MINISTRY SCIENCES AND HIGHER EDUCATION RUSSIAN FEDERATIONS
Federal State Autonomous Educational Institution
institution higher education
"NORTH CAUCASIAN FEDERAL UNIVERSITY"

Methodological guidelines
To practical classes on the
subject

Mathematical **Logic and Theory of Algorithms for** AI

For bachelors directions
11.03.02 Infocommunication technologies and communication systems

Stavropol, 2026

Content

Table of contents

1 LOGICAL TRANSFORMATIONS 3
2 MINIMIZATION BOOLEAN FUNCTIONS 10
3 SYNTHESIS LOGICAL SCHEME 19
4 APPENDICES CALCULATIONS STATEMENTS TO THEORIES EVIDENCE 28
5 METHOD RESOLUTIONS IN CALCULATION 37 STATEMENTS....
1. From methodological developments decide tasks With even numbers 50
6 KA LOGS PREDICATES 51
7 CONSTRUCTION CONCLUSIONS IN CALCULATION PREDICATES 57
8 MODELS FIRST-ORDER LOGIC 67
LITERATURE 81

1. Main concepts algorithmization And programming ....

1.1. Stages solutions tasks on Computer ....
1.2. Basics of Algorithmization ....
1.3. Theoretical basics programming ....
1.4. Rules records V C++ arithmetic expressions ....
Laboratory Job No. 1 Recording arithmetic expressions ....
Control questions ....

2. Programming algorithms linear structures ....

2.1. General characteristic language C++ ....
2.2. Main concepts language ....
2.3. Data And methods their organizations ....
2.4. Standard simple types data ....
2.5. Structure programs ....
2.6. Operators ....
2.7. Organization input/output data ....
Laboratory Job No. 2 Programming algorithms linear structures ....
Control questions ....
Example execution laboratory works ....

3. Operators branching ....

3.1. Simple And composite conditions ....
3.2. Composite operator ....
3.3. Conditional operation (?:) ....
3.4. Conditional operator if ....
3.5. Operator switch ....
3.6. Operator transition goto ....
Lab #3 Programming branching algorithms structures ....
Control questions ....
Example execution laboratory works ....

4. Cycles ....

4.1. Operator cycle With parameter (for) ....
4.2. Operator cycle With precondition (while) ....

4.3. Operator cycle With postcondition (dowhile)
4.4. Nested cycles
Laboratory Job No. 4 Programming algorithms cyclical structures 79
Control questions
Example execution laboratory works

5. Subroutines

5.1. Concept subroutines
5.2. Formal And factual parameters
5.3. Local And global variables
5.4. Functions
Laboratory Job No. 5 Usage functions For solutions applied 97 tasks
Control questions
Example execution laboratory works

6. Iteration And recursion

6.1. Concept iterative Process
6.2. Concept recursion
Laboratory Job No. 6 Programming recursive 103 algorithms
Control questions
Example execution laboratory works

7. One-dimensional arrays

7.1. Concept structured type data
7.2. Concept And description type array
7.3. One-dimensional arrays
7.4. Main actions over elements 112 arrays
Laboratory Job No. 7 Usage numerical one-dimensional 117 arrays
Control questions
Example execution tasks II laboratory works

8. Two-dimensional arrays

Laboratory Job No. 8 Two-dimensional arrays
Control questions
Example execution laboratory works

9. Algorithms for solving internal sorting problems and search algorithms information

9.1. Complexity 131 algorithms
9.2. Production tasks search
9.3. Consistent (linear) search
9.4. Binary search
9.5. Production tasks sorting data
9.6. Straight And fast methods internal sorting
9.7. Sorting inserts
9.8. Sorting With help direct choice
9.9. Sorting With help direct exchange
Laboratory Job No. 9 Tasks sorting And search
Control questions

10. Signposts And arrays

10.1. Concept static And dynamic variable

10.2. Pointers
10.3. Interrelationship between arrays and signs
10.4. Order announcements dynamic 144 arrays
10.5. Broadcast arrays as parameters functions
Lab #10 Using arrays and pointers to solve applied problems
Control questions
Example execution tasks III laboratory works

11. Peculiarities work With functions

11.1. Methods transmissions parameters V function
11.2. Broadcast names functions V as parameters
11.3. Overload functions
Laboratory Job No. 11 Study ways work With functions
Control questions

12. Lines How array elements type char

12.1. Methods of representing strings in C; implementation of some typical operations over lines
12.2. Main functions Si For work with lines
Laboratory Job No. 12 Processing symbolic arrays
Control questions
Example execution tasks II laboratory works

13. Lines How object special class string

13.1. Description lines V C++ And typical operations over there are 175 of them
13.2. Main functions class string For work with lines
Lab #13 Programming string data processing tasks type
Control questions
Example execution laboratory works

14. Structures

14.1. Order announcements And initialization structures
14.2. Programming With using structures
14.3. Usage functions For work With derivatives types data
Laboratory Job No. 14 Application structures For solutions applied problems
Control questions
Example execution tasks I laboratory works

15. Files

15.1. Approach With using opportunities language C
15.2. Approach With using opportunities language C++
Laboratory Job No. 15 Study methods access To files data
Control questions
Example execution tasks I laboratory works

16. List

16.1 Study formations elements related list
16.2 Study operations With elements related 215 lists
Laboratory Job No. 16 Study related 217 data
Control questions
Example execution tasks II

17. Trees .....

17.1 Concept trees .....
17.2 Classification 243 trees.....
17.3 Binary tree .....
17.4 Job With binary tree .....
Lab work No. 17 Research of features data processing at use binary tree .....
Control questions .....
Example execution tasks I .....

18. Vector .....

18.1 Concept vectors. Description .....
18.2 Main methods Vectors .....
18.3 Iterators .....
Laboratory Job No. 18 Usage vectors at decision tasks **. 257**
Control questions .....
Example execution tasks I And II laboratory work 17,263.....

19. Sets .....

19.1 Sets as computational structures: definition, description and restrictions .....
19.2 Operations over sets .....
19.3 Iterators .....
Laboratory Job No. 19 Usage sets at decision tasks **274**
Control questions .....
Example execution tasks I laboratory works .....

20. Modules .....

20.1 Modular programming .....
20.2 Modular programming means languages C And C++ .....
Laboratory Job No. 20 Usage vectors at decision tasks **. 279**
Control questions .....

21. Classes V C++ .....

21.1 set - functions And get - functions classes .....
21.2 Designers .....
21.3 Announcement class in a separate file .....
21.4 Department interface from sales of .....
Laboratory Job No. 21 Usage objects at decision tasks **293**
Option 1,294.....
Option 2,294.....
Option 3,294.....
Option 4,294.....
Variant 5 .....
Variant 6 .....
Variant 7 .....
Variant 8 .....
Variant 9 .....
Option 10 .....
Option 11 .....
Option 12 .....
Option 13 .....

Option 14 .....
Option 15 .....
Option 16 .....
Option 17 .....
Option 18 .....
Option 19 .....
Option 20 .....

Application 1 Order execution laboratory 300 works.....
Application 2 Basic functions 301

1 LOGICAL TRANSFORMATIONS

Learning objectives **:** To reinforce the definitions of logical operations on propositions, to learn how to compile truth tables for propositional algebra formulas, to reinforce the skills of using the laws of logic to prove equivalences, to master the principle of duality in propositional algebra.

1.1 Educational questions classes:

1. Execution logical transformations.
2. Solution tasks With using laws logic statements.

1.2 Questions For checks readiness students To practical lesson

1. All sorts of things whether offer natural language is a statement?
2. What means union "or" V non-exclusive what do you mean?
3. Provide verbal definitions of logical operations on statements and provide their designations.
4. Define concept formulas algebra statements.
5. Which formula it's called tautology?
6. Which formula it's called identically false?
7. Which formula it's called feasible?
8. Which formulas are called equivalent?
9. Write down main laws logic (tautologies).
10. Formulate law duality For Boolean formulas.

1. Execution logical transformations

Let's agree For further meaning "true" designate digital 1, A meaning "false" - the number 0 (in the literature the notations T (true) and F (false) are also used) etc.) Then The definitions of the five basic logical operations on statements (negation, conjunction, disjunction, implication and equivalence) can be summarized in one table (Table 1.1), which is called *a truth table* :

Table 1.1 - Table truth For definitions main logical operations

X	y	$\bar{x}$	x&y	$X \vee at$	$X \rightarrow at$	$X \leftrightarrow y$
0	0	1	0	0	1	1
0	1	1	0	1	1	0
1	0	0	0	1	0	0
1	1	0	1	1	1	1

Similar tables truth often use For proof logical identities and equivalences, laws of logic, in various applications of Boolean algebra. this practical lesson on basis definitions logical operations and laws logic We we will install, To which one class (tautologies, (identically false or satisfiable) belong to the given formulas of Boolean algebra. In addition, we will learn to express some logical operations through others, as well as find formulas dual to data formulas. Material this practical classes necessary for fastening And acquisitions protozoa skills For ghosts logical functions To SDNF (SKNF), their optimization And synthesis logical schemes.

Let us recall that any complex statement that can be obtained from elementary statements by applying logical operations and parentheses, regulating order their execution, it's called *formula algebra logic* .

Moreover , the elementary statements themselves, as well as the constants 0 and 1, are also the simplest formulas.

All possible logical values of a formula, depending on the values of the elementary statements included in it, can be described completely using a truth table.

By definition , the formula $f^*(x_1, x_2, ..., x_n)$ is called *the dual* formula $f(x_1, x_2, ..., x_n)$, if $f^*(x_1, x_2, ..., x_n) \equiv \overline{f(\bar{x}_1, \bar{x}, ..., \bar{x}_n)}$, at this column values f^* tables The truth value is obtained from the column of values of the function *f* by inverting the latter and negating it.

For Boolean formulas, the dual formula can be obtained by replacing the & operation with $\vee$, the $\vee$ οπερατιον with &, 1 with 0, and 0 with 1, all while preserving the formula's structure (i.e., the corresponding order of operations). If the formulas contain implication and equivalence operations, they are first expressed through the operations of negation, conjunction, and disjunction.

To perform identical transformations of propositional logic formulas, it is necessary *to know* :

A) definition logical operations over statements; b) the basic

laws of Boolean algebra;

be able to : make up tables truth For complex statements.

Let us list the main properties (laws) of Boolean operations of propositional algebra. In this case, we agree the equivalence relation on a set of statements (formulas) is denoted by the sign " $\equiv$ ".

1. $x \& x \equiv x$
2. $x \vee x \equiv x$

} - laws idempotency

3. $x \& 1 \equiv x$
4. $x \vee 1 \equiv 1$
5. $x \& 0 \equiv 0$
6. $x \vee 0 \equiv x$

} - zero laws and units;

7. $x \& \bar{x} \equiv 0$ - law contradictions;
8. $x \vee \bar{x} \equiv 1$ - law of the excluded middle;
9. $x \& (y \vee x) \equiv x$
10. $x \vee (y \& x) \equiv x$

} - absorption laws ;

11. $\bar{\bar{X}} \equiv X$ - law double denials;
12. $\overline{x \& y} \equiv \bar{x} \vee \bar{y}$
13. $\overline{x \vee y} \equiv \bar{x} \& \bar{y}$

} - laws de Morgan;

14. $x \,\&\, y \equiv y \,\&\, x$
15. $x \vee y \equiv y \vee x$ - commutative laws

16. $x \,\&\, (y \,\&\, z) \equiv (x \,\&\, y) \,\&\, z$
17. $x \vee (y \vee z) \equiv (x \vee y) \vee z$ - associative laws

18. $x \,\&\, (y \vee z) \equiv (x \,\&\, y) \vee (x \,\&\, z)$
19. $x \vee (y \,\&\, z) \equiv (x \vee y) \,\&\, (x \vee z)$ - distributive laws

20. $x \to y \equiv \bar{y} \to \bar{x}$ - law counterpositions.

Necessary Also be able to express alone logical operations through other. Among Among such equivalences, the following equivalences play a special role:

1. $x \leftrightarrow and \equiv (x \to and) \,\&\, (and \to x)$;
2. $x \to y \equiv \bar{x} \in\in y$; $x \to y \equiv \overline{x \,\&\, \bar{y}}$;
3. Laws where Morgana 12 and 13.
4. $x \,\&\, and \equiv \overline{\bar{x} \vee \bar{and}}$;
5. $x \vee and \equiv \overline{\bar{x} \,\&\, \bar{and}}$;
6. $\overline{x \to and} \equiv x \,\&\, and$

and dro.

Задачи

10.1. Compose tables truth For following formulas And install, which of them are tautologies. To what class do the remaining formulas belong?

10.1.1. $(x \vee y) \to ((x \,\&\, \bar{y} \vee x) \to \bar{y})$;

10.1.2. $(x \,\&\, \bar{y}) \to ((y \vee \bar{x}) \to \bar{z}))$;

10.1.3. $(\bar{x} \vee z) \,\&\, (y \to (\bar{z} \to x))$;

10.1.4. $\overline{\overline{x \vee y} \to \overline{x \,\&\, y}}$;

10.1.5. $((x \to y) \,\&\, \bar{y}) \to \bar{x}$;

10.1.6. $((p \,\&\, (q \to p)) \to \bar{p}$.

IN quality example let's consider formula 1.1.2

x	y	z	$\bar{x}$	$\bar{y}$	$\bar{z}$	$x \& \bar{y}$	$y \vee \bar{x}$	$(y \vee \bar{x}) \to \bar{z}$	f
0	0	0	1	1	1	0	1	1	1
0	0	1	1	1	0	0	1	0	1
0	1	0	1	0	1	0	1	1	1
0	1	1	1	0	0	0	1	0	1
1	0	0	0	1	1	1	0	1	1
1	0	1	0	1	0	1	0	1	1
1	1	0	0	0	1	0	1	1	1
1	1	1	0	0	0	0	1	0	1

Formula 1.1.2 is tautology, So How meaning this formulas on everyone sets equals 1.

10.2. Using the basic laws of logic, prove the following relationships with the help of equivalent transformations:

10.2.1. $x \vee y \equiv \overline{\bar{x} \& \bar{y}}$;

10.2.2. $(x \& y) \vee (x \& \bar{y}) \equiv x$;

10.2.3. $x \vee x \& y \equiv x$;

10.2.4. $\bar{x} \vee \bar{y} \equiv y \to \bar{x}$;

10.2.5. $(x \& y) \vee (\bar{x} \& \bar{y}) \vee (\bar{x} \& y) \equiv x \to y$;

10.2.6. $(x \to \bar{y}) \equiv (y \to \bar{x})$.

Let's prove, for example, 1.2.5.

$$(x \& y) \vee (\bar{x} \& \bar{y}) \vee (x \& y) \equiv (x \& y) \vee (\bar{x} \& (\bar{y} \vee y)) \equiv (x \& y) \vee \bar{x} \equiv (x \vee \bar{x}) \& (y \vee \bar{x}) \equiv y \vee \bar{x} \equiv x \to y.$$

10.3. Prove identical truth or identical falsity formulas, applying equivalent transformations:

10.3.1. $(x \vee \overline{(x \& y)}) \sim (y \vee x)$;

10.3.2. $(x \to y) \vee (x \to \bar{y})$;

10.3.3. $(x \vee y) \& \bar{x} \to y$;

10.3.4. $x \& (x \to y) \& (x \to \bar{y})$;

10.3.5. $(x \to y) \& \bar{y} \to \bar{x}$;

10.3.6. $(x \to y) \to \overline{(x \vee y)}$.

Let's prove it identical falsity formulas 1.3.4.

$$x \,\&\, (x \to y) \,\&\, (x \to \bar{y}) \equiv x \,\&\, (\bar{x} \vee y) \,\&\, (x \vee \bar{y}) \equiv x \,\&\, (x \vee (y \,\&\, \bar{y})) \equiv x \,\&\, \bar{x} \vee 0 \equiv x \,\&\, \bar{x} \equiv 0.$$

10.4. Next formulas transform So, to They contained only operations of conjunction and negation:

10.4.1. $x \vee y$;

10.4.2. $x \to y$;

10.4.3. $x \leftrightarrow y$;

10.4.4. $x \vee (x \leftrightarrow y)$;

10.4.5. $\overline{(x \,\&\, \bar{y})} \to (y \to x)$;

10.4.6. $(x \vee y) \to \overline{(x \to z)}$.

Let's consider, For example, 1.4.4:

$$x \vee (x \leftrightarrow y) \equiv x \vee ((x \to y) \,\&\, (y \to x)) \equiv x \vee (\bar{x} \vee y) \,\&\, (\bar{y} \vee x) \equiv$$

$$\equiv x \vee \overline{\left(\overline{(x \,\&\, \bar{y})} \,\&\, \overline{(y \,\&\, \bar{x})}\right)} \equiv x \,\&\, \overline{\left(\overline{(x \,\&\, \bar{y})} \,\&\, \overline{(y \,\&\, \bar{x})}\right)}$$

10.5. Find dual formulas to the formulas:

10.5.1. $x \,\&\, \overline{(y \vee z)}$;

10.5.2. $(x \,\&\, y) \vee (x \,\&\, z)$;

10.5.3. $\overline{(x \vee y)} \,\&\, (x \vee \overline{(y \,\&\, \bar{z})})$;

10.5.4. $((x \,\&\, y) \vee (y \,\&\, z) \vee (z \,\&\, x)) \,\&\, \overline{(x \vee y \vee z)}$;

10.5.5. $((x \vee y) \,\&\, \overline{(x \vee z)} \vee (x \,\&\, y)) \vee \overline{((x \vee y) \,\&\, z \vee x)}$;

10.5.6. $\overline{(x \,\&\, y \,\&\, z)} \vee (x \,\&\, y \,\&\, \bar{z}) \vee (x \,\&\, \bar{y} \,\&\, z) \vee \overline{(x \,\&\, y \,\&\, z)}$..

IN quality example let's build formula f^*, dual formula 1.5.3, based on from the definition of the dual formula:

$$f^*(x,y,z) \equiv \overline{\overline{f}(x,y,z)} \equiv \overline{\overline{(\bar{x} \vee \bar{y})} \,\&\, \bar{z}} \equiv \overline{(x \vee y) \,\&\, (x \vee (y \,\&\, z))} \equiv$$

$$\equiv \overline{(x \,\&\, y)} \vee \overline{(\bar{x}(y \,\&\, z))} \equiv \overline{(x \,\&\, y)} \vee (x \,\&\, \overline{(y \,\&\, z)}) \equiv \overline{(x \,\&\, y)} \vee (x \,\&\, (\bar{y} \vee \bar{z})),$$

which coincides with the formula obtained from formula 1.5.3 by replacing the operation & for surgery $\vee$ and vice versa.

At decision tasks 1.1 - 1.5 Can stick to next methods. One Go over the problem in detail at the board. Then divide the board into two parts and invite two students to it, who will solve two more problems simultaneously. Divide the entire group into two subgroups, each from which will track for solution tasks at boards your own

representative. Those the most will more wide coverage tasks, at boards More students will work and there will be time savings.

2. Solution tasks With using laws logic statements

Based on on law contrapositions $(x \to y) \equiv (\bar{y} \to \bar{x})$ For theorems Pythagoras formulate the inverse, opposite, and converse of the opposite proposition and establish the truth of each of them.

Solution . We formulate the Pythagorean theorem in the language of implication "if, ..., then ...": *if a triangle is right-angled* , *then the sum of the squares of the legs is equal to the square of the hypotenuse* - a true statement $(x \to y)$.

The opposite statement has view $at \to X$: *If V some triangle sum squares length any sides equal square lengths third sides* , *That This triangle is a right triangle* . This statement is also true.

The opposite statement: *If triangle Not rectangular* , *That V him sum squares length none two sides Not equal square lengths third* $(\bar{X} \to \bar{y})$.

Applying the law of contraposition in the form $(at \to \bar{X}) \equiv X \to at$, we are convinced of the truth of this statement.

Finally, expressing verbally formula $\bar{y} \to \bar{x}$, we get offer, the opposite opposite to the Pythagorean theorem: *if in a triangle the sum of the squares of the lengths If the length of any two sides is not equal to the square of the length of the third side* , *then this triangle is not a right triangle* . By the law of contraposition, written at the beginning of the problem, we see that this statement is also true.

Note, What law contrapositions lies V basis proof statements *method from contrary* .

2.1. Formulate four types of statements, as in problem 2.1, and establish the truth of each of them for theorems:

A) dot intersections diagonals parallelogram is his center of symmetry;

b) all sorts monotonous And limited subsequence has limit.

2.2. Find statement X, If $(X \vee A) \vee \overline{(X \vee A)} \equiv \overline{b}$.

Answer : $X \equiv \bar{b}$.

2.3. *The Sheffer prime* of two statements x and y is the statement denoted by $x \mid y$ (read " x is not compatible with y "), which is false if and only if When both statements are true. Compose table truth stroke Sheffer and express it through the operations &, $\vee$, $\bar{}$ and vice versa, the operations &, $\vee$, $\bar{}$ express through the Sheffer stroke.

Answer : $xy \equiv \overline{x \,\&\, and} \equiv \bar{x} \vee \bar{y}$; $x \equiv x \mid x$; $x \,\&\, and \equiv (x \mid and) \mid (x \mid and)$; $x \vee and \equiv (x \mid x) \mid (and \mid and)$.

2.4. *Pierce's Arrow* (Lukasiewicz's stroke) of two statements x and y is called the statement denoted by $x \downarrow y$ (read "neither x nor y "), which is true in that and only V volume case, When both data statements are false. Compose table

truth arrows Pierce And express her through &, $\vee$, $\bar{}$ and back, operations &, $\vee$, $\bar{}$ express through Pierce's arrow .

Answer : $x \downarrow y \equiv \overline{x \vee y} \equiv \bar{x} \,\&\, \bar{y};$

$\bar{x} \equiv x \downarrow x;$

$x \vee and \equiv (x \downarrow and) \downarrow (x \downarrow and);$

$x \,\&\, y \equiv (x \downarrow x) \downarrow (y \downarrow y).$

On lesson recommended decide from p.1 tasks With even numbers And tasks 2.1 And 2.4.

Conclusion

Let given arbitrary non-empty set $M = \{ a, b, with, \ldots x, at, z, \ldots \}$ And on him

Two binary algebraic operations, conventionally called multiplication and addition, are defined and have properties similar to those in 1–20. The set *M* in this case is called ***a Boolean algebra*** . This algebra is abstract. If we consider specific sets and specific operations on them, we obtain various interpretations of the Boolean algebra. Thus, the algebra of logic is one interpretation of the Boolean algebra. Other examples of interpretations of the Boolean algebra include the algebra of sets, the algebra of combinational circuits, and so on.

The duality principle allows us to almost halve the effort required to derive identities when examining the properties of elementary Boolean functions. There are other applications of the duality principle.

1.3 Exercise students For independent work

1. From this methodological developments decide tasks 1.1.1, 1.1.3, 1.1.5, 1.2.1, 1.2.3, 1.2.5, 1.3.1, 1.3.3, 1.3.5, 1.4.1, 1.4.3, 1.4.5, 1.5.1, 1.5.3, 1.5.5, 2.2, 2.3, 2.5.
2. Prepare for a practical lesson on the topic "Minimization of Boolean functions". In addition, prepare questions related to the reduction of Boolean functions to perfect disjunctive and conjunctive normal forms.

2 MINIMIZATION BOOLEAN FUNCTIONS

Learning objectives **:** Acquire skills in reducing Boolean functions with a small number of variables to a minimal form using analytical methods (Quine's method) and graphical methods (Karnaugh maps), including the minimization of partially defined functions. Learn how to transform a Boolean function to a determinate function (SDNF) using truth tables.

2.1 Educational questions classes:

1. Minimization Boolean functions method Quina
2. Minimization Boolean functions method kart Carnot
3. Minimization Not fully certain logical functions.

2.2 Questions For checks readiness students to class

1. What it's called implicant Boolean functions $f(x_1,\ldots,x_n)$? What such implicent?
2. Which implicant it's called simple?
3. Which implicant it's called significant?
4. IN how the essence minimization Boolean functions Quine's method ?
5. What it's called abbreviated disjunctive normal form (SokDNF) of a Boolean function? What does the algorithm for finding the SokDNF boil down to?
6. What it's called dead-end DNF (TDNF) Boolean functions? How her find?
7. What way from TDNF choose minimum DNF (MDNF) Boolean functions?
8. What represents by yourself map Carnot? To depict blanks kart Carnot for functions of 2, 3 and 4 variables.
9. Describe algorithm searches MDNF Boolean functions on basis Karnaugh maps .
10. Which Boolean function it's called Not fully (partially) certain?
11. How is the definition of a partially defined function carried out during its minimization?

1. Minimization Boolean functions method Quine

Abbreviated and Dead-End DNFs (KNFs)

Definition 1. **The reduced disjunctive normal form** is the disjunction of all prime implicants of a given function.

Therefore, finding the reduced DNF of a function is the problem of finding all its prime implicants.

Similar way Can enter abbreviated KNF.

Definition 2. Disjunction (conjunction) of prime implicants (implicents), none of which is not redundant, is called its **dead-end** disjunctive (conjunctive) normal form – DDNF (TCNF).

Logical function Maybe be presented one or several TDNF And TKNF. Definition 3. TDNF (TKNF) logical functions, containing the least possible number variables And their denials, it's called **minimal** disjunctive (conjunctive) normal form - MDNF (ICNF).

An exact solution to the Boolean function minimization problem involves finding their MDNF (MCNF). Solving this problem involves the following steps:

1. Performance Boolean functions V SDNF or SKNF.
2. Receipt abbreviated DNF (KNF).
3. Finding all TDNF (TKNF).
4. Choice from received dead-end forms MDNF (ICNF).

Minimization of Boolean functions by Quine's method is carried out in strict accordance with the main stages of minimization indicated above. Since the representation

The minimization of the function in the SDNF is a separate problem, independent of the minimization; for a better understanding of Quine's method, we will limit ourselves to functions already reduced to the SDNF.

Obtaining a reduced DNF by Quine's method is as follows. By successive conducting everyone possible operations incomplete gluing type $xy \vee x\bar{y} = x$ each minterma SDNF With each to others minterm, A then everyone possible acquisitions constituent units species $xy \vee x = x$, We we will receive All simple Implicants of the function being minimized. This step is most conveniently performed using a rectangular table whose row and column names represent all the minterms included in the DNF. Using such a table, pairs of minterms are selected that differ from each other by only one variable in their proper parts. The result of this partial concatenation is written into the table at the intersection of the concatenated minterms. The table will be symmetrical with respect to the main diagonal, so to avoid unnecessary implicants, the disjunction of implicants is taken only in the upper (or lower) part of the table. The resulting implicants yield *the reduced* DNF.

Finding all dead-end DNFs can also be produced with Using a rectangular table (*an implicant* matrix), the columns of which correspond to the constituents of unity (minterms), and the rows to the prime implicants. From the resulting matrix, the minimum numbers of the shortest prime implicants are selected that jointly absorb (overlap) all the minterms of the SDNF of the function being minimized, and none of which is redundant. The disjunction of the prime implicants thus selected is a dead-end. In the implicant matrix, the cells at the intersection of rows and columns in which a prime implicant occurs in a given minterm are labeled. Of all the dead-end forms, the minimal one is selected.

In more complex examples, when the number of variables is greater than 3, when finding a reduced DNF, it is necessary to perform gluing and absorption operations on the newly obtained implicants several times, since they may not be simple. Note at this, What implicant, which V the only one number overlaps one or another constituent of the unit (minterm) are called *essential* . Essential implicants are always included in the MDNF.

Tasks

1.1. Method Quina find MDNF And ICNF functions, given in SDNF:

$$f(x, y, z) = x\bar{y}z \vee \bar{x}\bar{y}z \vee \bar{x}\bar{y}\bar{z} \vee \bar{x}y\bar{z} \vee xy\bar{z} \vee xyz, \qquad (2.1)$$

Solution . Let's fill it out rectangular table 1 By specified higher algorithm.

Table 2.1 - Primary gluing minterms in task 1.1

Minter we	$x\bar{y}z$	$\bar{x}\bar{y}z$	$\bar{x}\bar{y}\bar{z}$	$\bar{x}y\bar{z}$	$xy\bar{z}$	xyz
$x\bar{y}z$		$\bar{y}z$				xz
$\bar{x}\bar{y}z$	yz		$\bar{x}\bar{y}$			
$\bar{x}\bar{y}\bar{z}$		$\bar{x}\bar{y}$		$\bar{x}\bar{z}$		
xyz			xz		yz	
xyz				yz		xy
xyz					xy	

Since all the implicants at the top of Table 1 are prime (that is, they are no longer glued together in pairs), the reduced DNF of this function will be the function

$$f_{\text{C ok DNF}}(x, y, z) = yz \vee xx \vee xy \vee xx \vee yz \vee xy. \qquad (2.2)$$

Let's make it up implicant matrix V view tables 2.2, using abbreviated DNF (2.2).

Table 2.2 - Implicant matrix To task 1.1

Minterms / Prime implicants	$\bar{x}yz$	$\bar{x}\bar{y}z$	$\bar{x}yz$	$x\bar{y}z$	$xy\bar{z}$	xyz
$\bar{y}z$	*	*				
$\bar{x}\bar{y}$		*	*			
$\bar{x}\bar{z}$			*	*		
yz				*	*	
xy					*	*
xz	*					*

From implicant matrices it is visible, What All minterms given functions turn out to be blocked two ways: implicants yz, xz And xy; $x\bar{y}$, $\bar{y}z$, $\bar{x}\bar{z}$; hence,

We we have two dead-end DNFs: $f_1(x, y, z) = xy \vee yz \vee xz$, (2.3)

$$f_2(x, y, z) = \bar{x}\bar{y} \vee yz \vee xz. \qquad (2.4)$$

Each one from dead-end forms (3) And (4) is minimal, So How contains the same number of letters – 6 each and the same number of negatives – 3 each.

ICNF this functions Can write down, using principle duality:

$f_3(x, y, z) = (x \vee y)(y \vee z)(x \vee z)$ *or* $f_4(x, y, z) = (x \vee y)(y \vee z)(x \vee z)$.

1.2. Minimize method Quina logical function

$$f(x_1, x_2, x_3, x_4) = \bar{x}_1\bar{x}_2x_3x_4 \vee x_1\bar{x}_2\bar{x}_3\bar{x}_4 \vee \bar{x}_1x_2\bar{x}_3x_4 \vee x_1\bar{x}_2x_3x_4 \vee x_1x_2\bar{x}_3x_4 \vee$$
$$\vee x_1\bar{x}_2\bar{x}_3x_4 \vee x_1\bar{x}_2x_3\bar{x}_4.$$

Function presented V SDNF. Necessary determine MDNF.

Answer : V process minimization we will receive three dead-end DNF:

а) $f_1(x_1, x2, x3, x4) = x_1x3x4 \vee x_1x2x4 \vee x2x3$;

б) $f_2(x1, x_2, x_3, x_4) = x_2x_3x_4 \vee x_1x_2x_4 \vee x_1x_2x_4 \vee x_2x_3$;

с) $f_2(x_1, x2, x3, x4) = x_2x3x4 \vee x_1x2x4 \vee x_1x3x4 \vee x2x3$.

Function $f_1(x_1, x_2, x_3, x_4)$ contains the smallest quantity conjunctions And variables, That's why she is MDNF.

1.3. Minimize function, given in SDNF:

$$f(x, y, z, v) = \bar{x}\bar{y}\bar{z}v \vee \bar{x}yzv \vee x\bar{y}zv \vee xyzv \vee xyzv \vee xyzv \vee xyzv \vee xyzv \vee xyzv,$$

Solution : Let's create a rectangular table 2.3 and perform a rank reduction. The original minterms are multiplied by 1. Minterm gluing and absorption are performed as before: pairs of minterms that differ only in a variable and its negation are merged and absorbed. Due to the symmetry of the table, it is sufficient to fill only the upper part.

Table 2.3 - Primary gluing minterms in task 1.3

Minterm	xyz	$\bar{x}yz$	$x\bar{y}z$	$xy\bar{z}$	$\bar{x}\bar{y}z$	$\bar{x}y\bar{z}$	$x\bar{y}\bar{z}$	$\bar{x}\bar{y}\bar{z}$
$\bar{x}\bar{y}\bar{z}v$		$\bar{x}\bar{z}v$				$\bar{y}\bar{z}v$		
$xyzv$	$\bar{x}\bar{z}v$			xyv				
$xy\bar{z}v$				xyz				yzv
$x\bar{y}zv$		xyv	xyz					yzv
$x\bar{y}\bar{z}v$					$x\bar{y}z$	$x\bar{y}\bar{z}$		
$\bar{x}yzv$	$\bar{y}zv$				$x\bar{y}z$		xyv	
$xyzv$					$x\bar{y}\bar{z}$	xyz	xzv	
$x\bar{y}zv$						xyv	xyz	xzv
$xyzv$			yzv				xzv	xyz
$xyzv$				yzv			xzv	xyz

From received 13 implicant let's make up new table 2.4, because Not All They are simple. IN That same time Not involved implicants $\bar{x}\bar{z}v$, $\bar{y}\bar{z}v$, $\bar{x}yv$ V operations gluing will be simple. Except Togo, simple turn out to be implicants yz, $x\bar{y}$ And xz, since they are not subject to further gluing are subject to.

Table 2.4 - Secondary gluing of minterms in problem 1.3

Импли-канты	xzv	yzv	$x\bar{y}v$	xyz	yzv	$\bar{x}\bar{y}\bar{z}$	$x\bar{y}z$	$xy\bar{z}$	$\bar{x}yv$	xyv	xyz	xzv	$\bar{x}\bar{y}\bar{z}$
$\bar{x}\bar{z}v$													
$\bar{y}\bar{z}v$													
xyv													
xyz													yz
yzv						yz							
yzv					yz								

$\bar{x}\bar{y}z$										xy			
$\bar{x}\bar{y}v$									x				
xyv								xy					
xyz							x						xz
xzv												xz	
xzv											xz		
xyz				y						xz			

Thus, the reduced DNF of the original function takes the form

$f_{\text{SokDNF}} = xyz \vee xz \vee yz \vee xyv \vee xzv \vee yzv$.

We'll find it Now All dead-end And minimum DNF this functions. We are building implicant matrix (table 2.5):

Table 2.5 - Implicant matrix To task 1.3

Minterms / Pros- the implicant	$\bar{x}\bar{y}\bar{z}v$	$\bar{x}yzv$	$x\bar{y}zv$	$xyzv$	$\bar{x}\bar{y}\bar{z}v$	$\bar{x}\bar{y}zv$	$x\bar{y}zv$	$xyzv$	$xyz\bar{v}$	$xyzv$
$\bar{x}\bar{y}es$					*	*	*	*		
xz							*	*	*	*
yz			*	*					*	*
xyv		*		*						
$\bar{x}\bar{z}v$	*	*								
$\bar{y}\bar{z}v$	*					*				

IN ours case, How shows implicant matrix, significant are two simple implicants $\bar{x}y$ (only she overlaps minterm $\bar{x}yzv$) And $\bar{y}\bar{z}\bar{v}$ (only she overlaps minterm $x\bar{y}zv$). That's why They Necessarily should log in V minimal form.

Together implicants $\bar{x}y$ And $\bar{y}z$ overlap All minterms, except first $\bar{x}yzv$ And $\bar{x}\bar{y}\bar{z}\bar{v}$. Absorption specified two minterms Maybe be received two ways: or

implicant $yz\bar{v}$, or two implicants xyv And $yz\bar{v}$. IN result we get two dead-end DNFs

$f_1(x, y, from, in) = xy\bar{z} \vee yz \vee xzv$, $f_2(x, y, from, in) = xyz \vee yzv \vee xyzv$.

Because first TDNF contains 7 letters, A second - 10, That minimal form this functions is first TDNF f_1.

1.4. Define method Quina All TDNF And MDNF logical function

$f(x_1, x_2, x_3, x_4) = \sum(0, 3, 7, 8, 9, 11)$.

On the rest sets given function accepts meaning 0.

Answer: we will receive two TDNF: $f_1(x_1, x2, \bar{x}3, x4) = \bar{x}_1 x3x4 \vee \bar{x}_1 x2x4 \vee x2x3x4$;

$f_2(x1, x_2, x_3, \bar{x}_4) = x_1\bar{x}_3x_4 \vee x_2\bar{x}_3\bar{x}_4 \vee \bar{x}_1\bar{x}_2\bar{x}_3 \vee x_2x_3x_4$.

Obviously, What MDNF this functions matches With TDNF $f_1(x_1, x_2, x_3, x_4)$.

2. Minimization Boolean functions method kart Carnot

One of the possible ways of graphical representation of a logical function of a small number of variables is Karnaugh maps (Veitch diagrams).

Cards Carnot consists of 2^m cells, where *m* is the number of function arguments, and represent transformed truth tables. Each cell corresponds to a specific set of arguments. The cells contain either zeros or ones, depending on the values the function takes on each set. The presence of a one V cage indicates on that which corresponding her constituent units must be included in the SDNF function.

Karnaugh maps for functions of two variables consist of four cells. The columns and rows of the map can be designated either by the numbers 0 and 1 (Table 2.6) or by a bar (Fig. 2.1). The bar denotes the value 1 for the corresponding variable.

Table 2.6 - Map Carnot for 2 changes

x \ y	1	0
1	xy	$x\bar{y}$
0	xy	$\bar{x}\bar{y}$

y

xy	$x\bar{y}$
xy	$\bar{x}\bar{y}$

Figure 2.1 - The line indicates the value, equal 1, the variable indicated near this line

In Fig. 1, the top row of cells corresponds to an argument *x* equal to 1, the bottom row to an argument *x* equal to 0. Similarly, the left column corresponds to an argument *y* equal to 1, the right column to an argument *y* equal to 0. If a specific function is given, the cells will contain the values of this function for the corresponding set of arguments.

Units located in adjacent (horizontal or vertical, but not diagonal) cells can stick together, What correspond gluing minterms V

SDNF for one of the arguments. In the case of two arguments, two or four units can be glued together.

The basis for minimizing Boolean functions using Karnaugh maps is the following reasoning. Two minterma, located V neighboring cells cards, can be are replaced by a single conjunction containing one less variable. If two pairs of minterms are adjacent, then such a group of four minterms can be replaced by a conjunction that contains two fewer variables. In general, the presence of minterms in 2^k neighboring cells allows us to exclude k variables. This is easy to verify if adjacent pairs of minterms are transformed by the method of successive elimination of variables, using the distributive laws of conjunction with respect to disjunctions And disjunctions with respect to conjunctions, rules absorption and adhesion.

When minimizing Boolean functions using Karnaugh maps, it's important to remember that adjacent cells include not only cells adjacent horizontally and vertically, but also cells on opposite edges of the Karnaugh map. Cells can be grouped in groups of 2, 4, 8, and so on, and the same cell can be part of multiple groups.

Karnaugh maps can be used to minimize Boolean functions defined in both SDNF and SKNF. In the latter case, each cell must be assigned a specific zero constituent (maxterm). The presence of a 0 in a Karnaugh map cell indicates that the corresponding zero constituent must be included in the SKNF of the function. Adjacent zeros can be merged according to the same rules as ones.

Finding the MDNF comes down to determining the most rational option in which all units of the Karnaugh map of a given function are covered by the smallest number of short conjunctions.

Tasks

WITH help kart Carnot minimize functions previous point:

2.1. $f(x,y,z) = \bar{x}\bar{y}\bar{z} \vee \bar{x}\bar{y}z \vee \bar{x}y\bar{z} \vee \vee xy\bar{z} \vee xyz \vee x\bar{y}z$.

$f(x_1, x_2, x_3, x_4) = \bar{x}_1\bar{x}_2x_3x_4 \vee x_1\bar{x}_2\bar{x}_3\bar{x}_4 \vee x_1x_2\bar{x}_3x_4 \vee x_1x_2x_3x_4 \vee x_1\bar{x}_2\bar{x}_3x_4 \vee$

2.2. $\vee x_1\bar{x}_2x_3x_4 \vee x_1x_2\bar{x}_3\bar{x}_4 \vee x_1x_2\bar{x}_3x_4$.

2.3. $f(x,y,z,v) = \bar{x}\bar{y}\bar{z}v \vee \bar{x}yzv \vee \bar{x}\bar{y}zv \vee xyzv \vee x\bar{y}\bar{z}v \vee x\bar{y}zv \vee xy\bar{z}v \vee xyzv \vee x\bar{y}zv \vee \bar{x}yzv \vee xyzv$.

2.4. $f(x_1,x_2,x_3,x_4) = \sum(0, 3, 7, 8, 9, 11)$.

Solution 2.1. We get the following cards Carnot (rice. 2a) and 2b)).

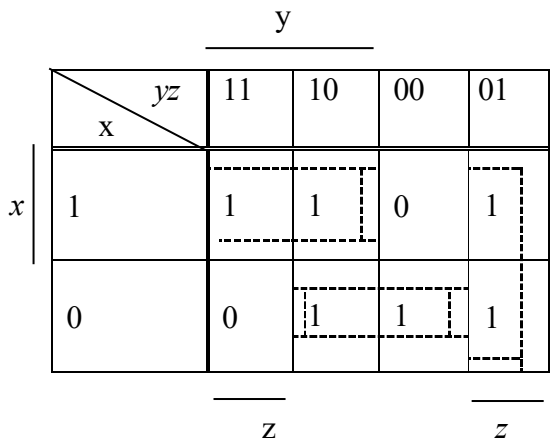


Figure 2.2a) - First gluing option

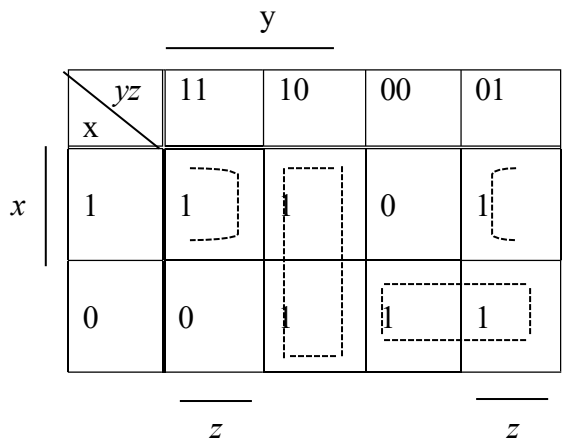
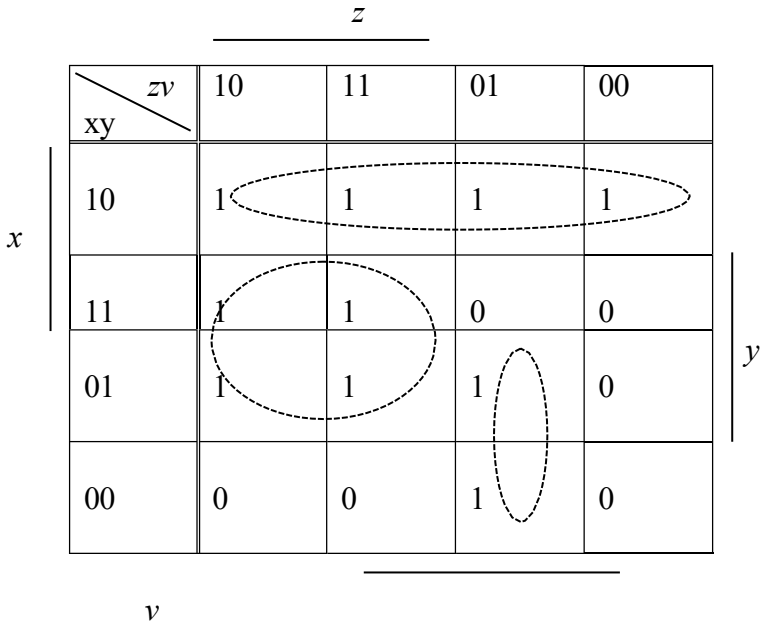


Figure 2.2b) - Second gluing option

IN map Carnot on Fig. 2a) All minterms form three groups: 1) xyz And $\bar{x}yz$, where $xyz \vee \bar{x}yz = xy$; 2) $xy\bar{z}$ и $\bar{x}y\bar{z}$, откуда $xy\bar{z} \vee \bar{x}y\bar{z} = y\bar{z}$; 3) $x\bar{y}z$ и $\bar{x}\bar{y}z$, откуда $x\bar{y}z \vee \bar{x}\bar{y}z = \bar{y}z$ и V result we get MDNF $f_1 = xy \vee xz \vee yz$, What matches With MDNF (3). Grouping units in a different way (Fig. 2b)), we get: 1) $xyz \vee x\bar{y}z = xz$; 2) $xyz \vee xyz = yz$;
3) $\bar{x}\bar{y}\bar{z} \vee \bar{x}\bar{y}z = \bar{x}\bar{y}$. Then MDNF will function $f_2 = xz \vee yz \vee x\bar{y}$, What matches With MDNF (4).

Solution 2.3. Grouping three groups minterms, How shown below on rice. 3, we get:
1) $\bar{x}\bar{y}zv \vee \bar{x}yzv \vee x\bar{y}zv \vee xy\bar{z}v = \bar{x}yz \vee x\bar{y}z = \bar{x}y$;2)
$\bar{x}yzv \vee xyzv \vee \bar{x}\bar{y}zv \vee xyzv = xyz \vee \bar{x}yz = yz$;
3) $\bar{x}\bar{y}zv \vee \bar{x}yzv = \bar{x}zv$, where $f_{\min} = xy \vee yz \vee \bar{x}zv$, What matches With previously received minimal form.



Drawing 2.3 - Map Carnot For functions 2.3

3. Minimization Not fully certain logical functions

Minimization of partially defined logical functions—function values for unspecified combinations of argument values—must be further defined to obtain the minimal DNF. Minimization of such logical functions can be performed using either Karnaugh maps or Quine's method.

Tasks

3. A partially defined function $f(x_1, x_2, x_3, x_4)$ is given by a truth table, and Namely, the numbers of the sets of arguments on which it takes on certain values are indicated. On the remaining sets of arguments, this function is undefined. Minimize the following functions.

3.1.

Set number N	1	2	3	4	6	7	8	9	11	12
Function value $f(x_1, x_2, x_3,$	0	0	1	0	1	1	0	1	1	0

x_4)									

3.2.

Set Number N	0	2	3	5	6	7	8	9	13	15
The value of the function $f(x_1, x_2, x_3, x_4)$	0	1	0	0	1	1	0	0	1	0

Conclusion

At a large number of variables ($n > 5$) method Quina as And method kart Carnot at Manual minimization methods become tedious and ineffective. However, using a given algorithm, it is possible to create a computer program that can automate the minimization process for a sufficiently large number of variables.

The material covered in this practical lesson serves as the basis for the synthesis of combinational circuits, which will be the subject of the next practical lesson.

2.3 Exercise students For independent work

1. From this methodological developments decide tasks 1.2, 1.4, 2.2, 2.4, 3.2.
2. Get ready To practical occupation on topic "Synthesis combinational circuits."

3 SYNTHESIS LOGICAL DIAGRAM

Learning objectives : To understand the problem of synthesizing combinational circuits in functionally complete sets of elements. To reinforce the methodology and main stages of the structural synthesis process, and to acquire basic skills and abilities in constructing combinational circuits in various functionally complete sets of elements.

3.1 Educational questions classes:

1. Sequential And parallel connections valves.
2. Main stages of constructing a combinational circuit in the selected functionally complete set.

3.2 Questions For checks readiness students to class

1. What such combinational scheme? Why We her we identify With "black "box"?
2. What logical operations correspond to serial and parallel connections of conductors?
3. What it's called basis Boolean functions?
4. Formulate production tasks synthesis logical schemes.
5. List the main stages of the process of structural synthesis of logical (combinational) circuits.

1. Sequential And parallel compound valves

Synthesis of a logic circuit is the design of this circuit if the law of its operation is given. In its most general formulation, the task of structural synthesis consists of constructing combinational circuits that implement given algorithms with using simple gates - *a conjunctor* , *a disjunctor* , *an inverter* , etc.

As a result of structural synthesis, a list of logical elements included in the combinational circuit and the order of their connection to each other must be determined. functionally complete systems functions initially Can take redundant system $\bar{}$, &, $\{\vee\ .\ \}$

This practical lesson is final course in a series of lectures and practical classes in which propositional algebra was studied.

Tasks

Draw the discrete signal circuits consisting of from contacts connected in series and in parallel.

Draw electrical and standard graphical representations (diagrams) of an inverter, disjunctor, conjunctor, Pierce arrow and Schaeffer stroke.

Bring examples functionally full systems functions (ligaments) algebra logic.

Bring examples bases.

1.4. Build combinational schemes, given functions:

1.4.1. $f(x,y,z) = xyz \vee \overline{xyz} \vee \bar{x}\bar{y}$;

1.4.2. $f(x,y,z) = x(yz \vee \overline{yz}) \vee \bar{x}(\bar{y}z \vee y\bar{z})$;

1.4.3. $f(x,y,z,u) = (\bar{x} \vee y) \vee (zy \vee x) \vee u$;

1.4.4. $f(x,y,z) = (x \to y)(y \to z)$.

Note, What at decision tasks 1.4.1-1.4.4 Not follows achieve optimality schemes, those. data functions Not follows on this stage to minimize.

As an example, let's consider function 1.4.2. Since the right-hand side of the function contains the expression $\overline{yz}$, we will use De Morgan's law and bring this function to the form

$f(x,y,z) = xyz \vee x\bar{y} \vee x\bar{z} \vee \bar{x}\bar{y}\bar{z} \vee \bar{x}yz$.

Then corresponding combinational diagram V functionally full set $\{\neg, \&, \vee\}$ Can depict drawing 3.1:

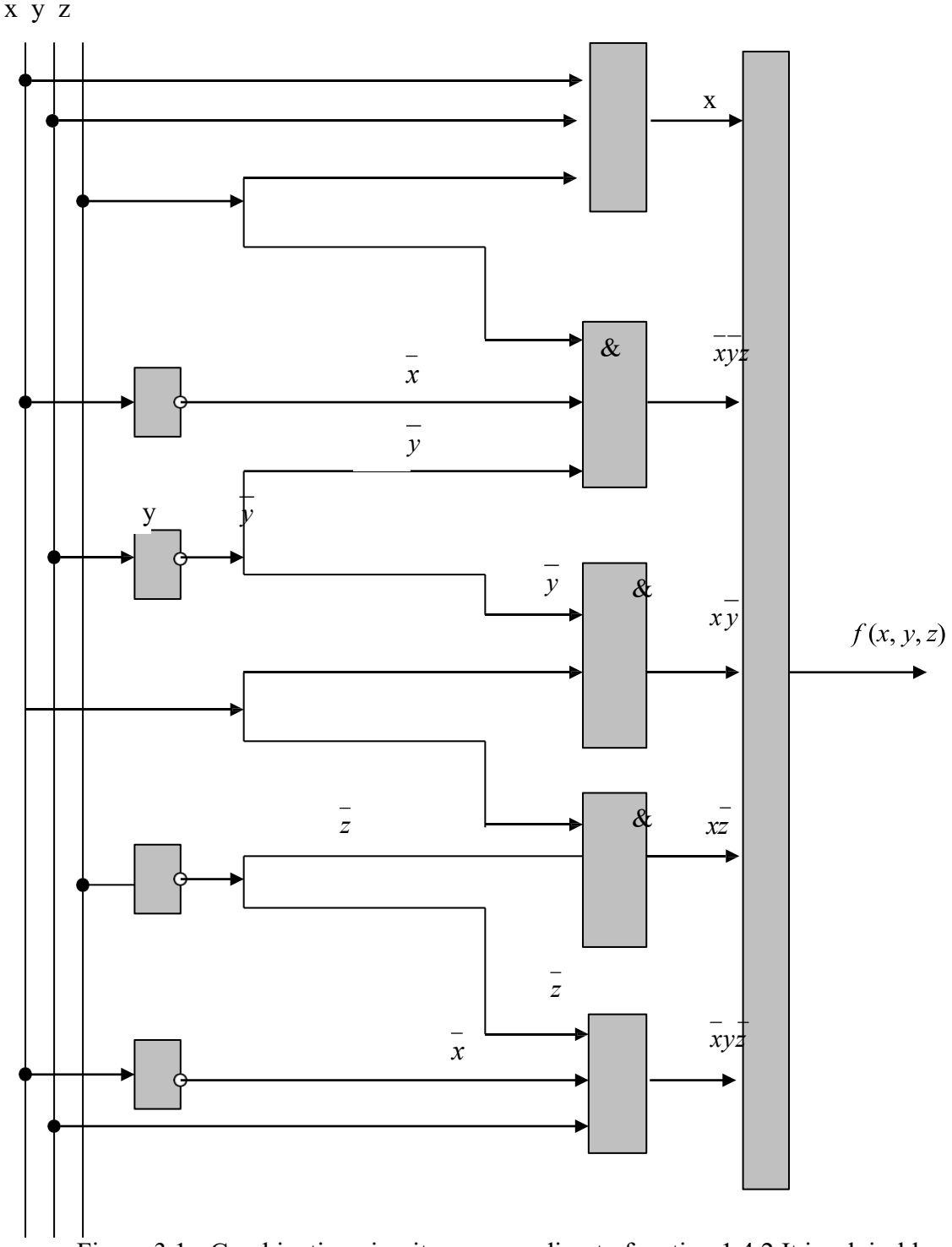


Figure 3.1 - Combination circuit corresponding to function 1.4.2 It is advisable on some sets check right work built combinational schemes.

On lesson recommended decide tasks 1.1-1.3, 1.4.2 And 1.4.4.

2. The main stages of constructing a combinational circuit in selected functionally complete set

Process structural synthesis logical schemes consists of from the following main stages:

1. Formalization verbal tasks schemes.
2. Compilation tables truth.
3. Recording logical functions V SDNF or SKNF.
4. Choice functionally complete systems logical elements.
5. Minimization logical functions.
6. Construction combinational schemes, corresponding logical functions.
7. Examination correctness work combinational schemes.

Although when synthesizing a combinational circuit, the formalization of a verbal task usually does not cause any particular difficulties and comes down to identifying sets of variables on which the function is equal to 0 or 1, in the lesson it is recommended to solve one of the problems using the first stage.

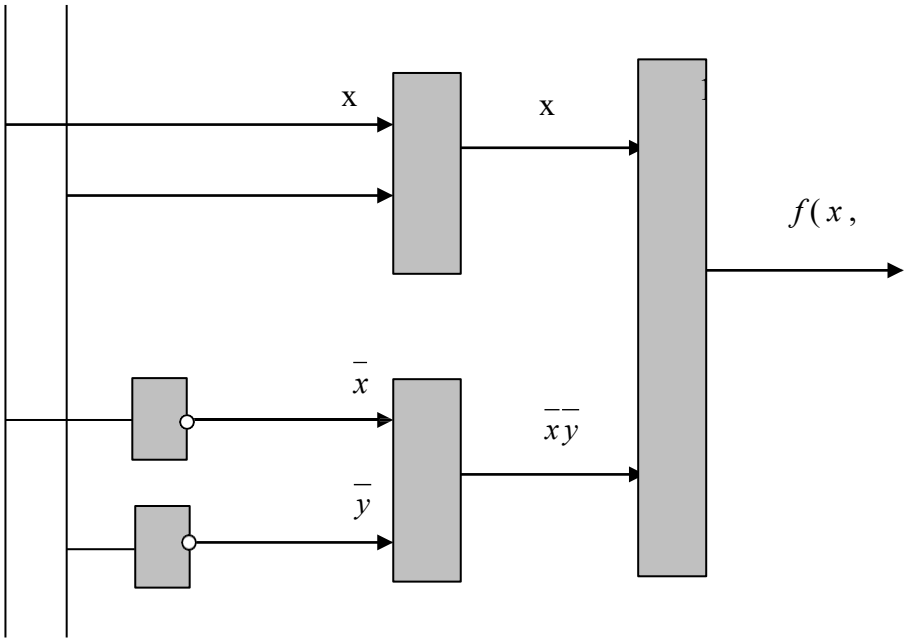
Tasks

2.1. At a given signal, each player **, A** and **B,** closes or opens the switch they control. If both do the same, then player **A wins** ; otherwise, player **B wins** . Construct a circuit so that if player **A wins** , the light bulb lights.

Solution . Let *x denote the state of player* **A's** switch , and *y denote the state of player* **B's** switch . Then, according to the problem statement, the function $f(x, y)$ takes the value 1 on sets (0, 0) And (1, 1). SDNF this functions The $f(x, y) = xy \vee \bar{x}\bar{y}$.

corresponding diagram looks like this (Fig. 3.2):

x y



Drawing 3.2 - Combinational scheme, feasible function tasks 2.1

Task 2.2. There is one light bulb in the stairwell of a two-story house. Design a circuit so that each floor's switch can turn the light bulb on and off independently of the other switch's position.

Answer $f(x,y) = \bar{x}y \vee y$.

:

From received V table 3.1 implicant let's make up new table 3.2, because Not All They are simple.

Table 3.2 - Secondary bonding

M ин-тер-мы	$\bar{a}\bar{c}d$	$\bar{b}\bar{c}d$	$\bar{a}bd$	$\bar{a}bc$	$b\bar{c}d$	bcd	$a\bar{b}\bar{c}$	$a\bar{b}\bar{d}$	$a\bar{b}d$	$ab\bar{c}$	$ac\bar{d}$	acd	abc
$\bar{a}\bar{c}d$	■												
$\bar{b}\bar{c}d$		■											
$\bar{a}bd$			■										
$\bar{a}bc$				■									bc
$b\bar{c}d$					■	bc							
bcd					b	■							
$a\bar{b}\bar{c}$							■			$a\bar{b}$			
$a\bar{b}\bar{d}$								■	$a\bar{b}$				
$a\bar{b}d$								a	■				
$ab\bar{c}$							$a\bar{b}$			■			ac
$ac\bar{d}$											■	ac	
acd											ac	■	
abc				bc						ac			■

Abbreviated DNF original functions takes the form:

$f(a,b,c,d) = \bar{a}\bar{c}d \vee \bar{b}\bar{c}d \vee \bar{a}bd \vee bc \vee a\bar{b} \vee ac$.

Now we'll find All dead-end And minimum DNF this functions With help construction of the implicant matrix (table 3.3).

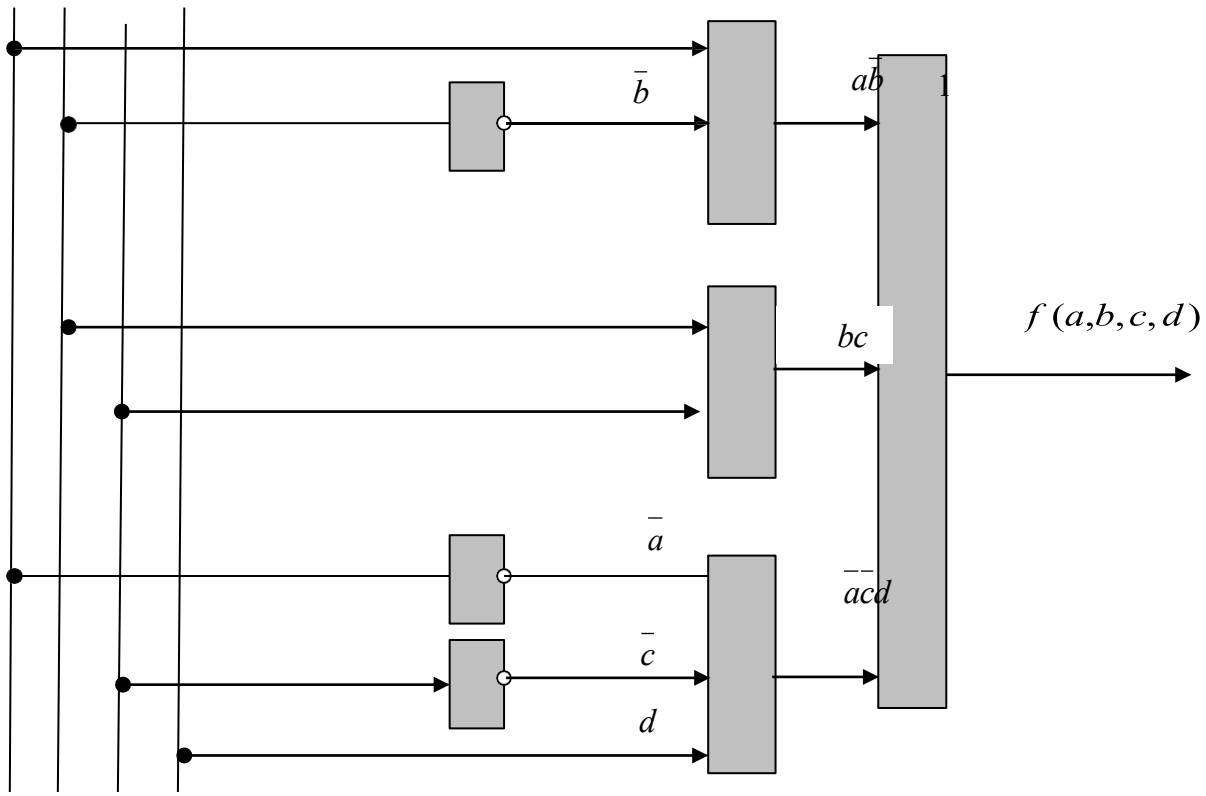
Table 3.3 - Implicant matrix

Минтермы / Импликанты	$\bar{a}\bar{b}\bar{c}d$	$\bar{a}b\bar{c}d$	$a\bar{b}\bar{c}d$	$ab\bar{c}d$	$\bar{a}\bar{b}cd$	$\bar{a}bcd$	$a\bar{b}cd$	$abcd$	$abc\bar{d}$	$ab\bar{c}\bar{d}$
$a\bar{b}$					*	*	*	*		
ac							*	*	*	*
bc			*	*					*	*
$\bar{a}bd$		*		*						
$\bar{a}\bar{c}d$	*	*								
$\bar{b}\bar{c}d$	*					*				

We have two dead-end forms $f_1 = a\bar{b} \vee bc \vee \bar{a}\bar{c}d$ $f_2 = ab \vee bc \vee \bar{a}bd \vee \bar{b}\bar{c}d$, And

at this implicants ab And $\bar{b}c$ are essential.

Let's build it combinational diagram, corresponding minimal form this functions.



Drawing 3.3 - Combinational scheme, feasible function tasks 2.3

Remains to carry out check correctness work this combinational schemes.

Boolean function given in video numerical works $f(x, in, z) = \prod (3.5)$ on 3rd and on 5th sets. Following the main stages of combinational circuit synthesis, synthesize this function in the set $\{\&, \vee, \bar{}\}$, as well as in the bases Pierce arrow $\{\downarrow\}$ and Sheffer stroke $\{|\}$.

Solution. A) Function $f(x, at, z)$ By definition numerical works on 3rd set <0, 1, 1> and on the 5th set <1, 0, 1> takes the value 0, on the remaining sets this function takes the value 1. b) The truth table for this function takes the form of table 3.4:

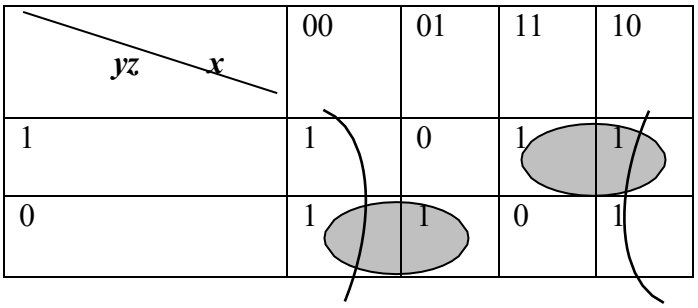
Table 3.4 - Table truth For functions $f(X, at, z)$

x	0	0	0	0	1	1	1	1
y	0	0	1	1	0	0	1	1
z	0							
$f(x, y, z)$	1							
Number folds	0							

c) The SDNF of this function is its entry in the form $f(x, y, z) = \Sigma(0, 1, 2, 4, 6, 7)$ (under the sum sign are the numbers of sets on which the function f takes the value 1). Then The SDNF of the function f takes the form $f(x, y, z) = xyz \vee hu\, z \vee xy\bar{z} \vee xy\bar{z} \vee x\bar{y}\bar{z} \vee \bar{x}\bar{y}\bar{z}$. From that same tables truth easily we get SKNF this functions: $f(x, y, z) = (x \vee \bar{in} \vee \bar{z})(\bar{x} \vee \bar{in} \vee z)$.

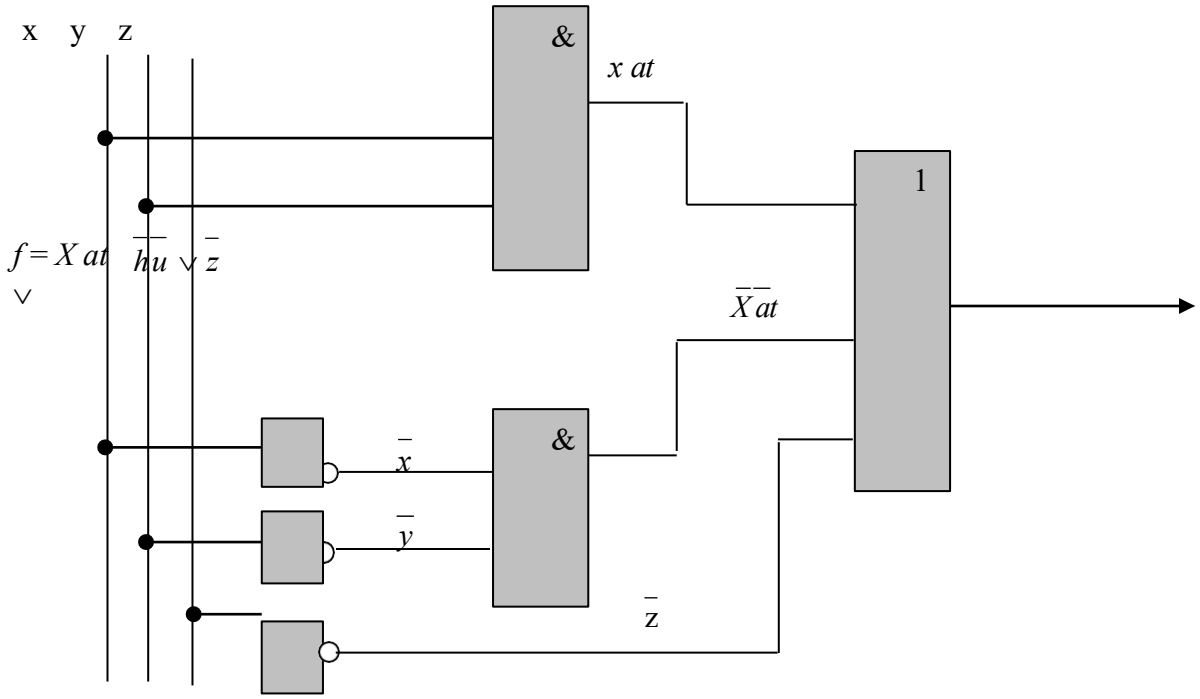
G) Let's consider at first excess functionally complete system $\{\&, \vee, \bar{}\}$. d) We will carry out minimization using the Karnaugh map.

By gluing the minterms as shown in Figure 3.4, we obtain the minimum disjunctive normal form functions f: $f = hu \vee \bar{h}u \vee z$.



Drawing 3.4 - Map Carnot For minimized functions f

The MCNF is obtained immediately in the form of a numerical product $f(\bar{x}, y, \bar{z}) = (X \vee \bar{at} \vee z)(x \vee at \vee z)$. e) We will construct the CS in two versions: for MDNF (Fig. 3.5) and for MCNF (Fig. 3.6).



Drawing 3.5 - Combination circuit built on the basis of the MDNF of this function

Comparing two received KS, Can notice, What V this case cost on construction Togo or other options schemes are practically identical.

g) We will verbally check the correct operation of the obtained KS on some sets.

Let us proceed to the synthesis of this function in the bases {|} and { ↓}. Let us recall the rules for the transition of this function to the specified bases. If the function f has the form DNF $f = d_1 \vee d_2 \vee \ldots \vee d_s$, then, taking from right parts this DNF double negation And having applied To him rule where $\overline{d_1} \mid \overline{d_2} \mid \ldots \mid \overline{d_s}$ Morgana, we will ... . From here we have receive $f =$

Rule 1. In order to move from the DNF of a Boolean function to its representation in the basis {|} , it is necessary to replace all disjunction and conjunction signs with Sheffer symbols | and enclose the groups of letters included in each elementary conjunction in brackets.

For compactness records can be former performance save inversions, having at this V view, What $\overline{X} = \overline{xx} = X \mid X$ And etc. Then finding minimal forms V basis {|} is performed according to the following algorithm: 1) determine the MDNF of the function; 2) replace all ∨ and & operations with a Sheffer stroke and we put in brackets the letters that are part of the elementary conjunctions MDNF functions; 3) we replace denials arguments $\overline{X}, \overline{at}$,…expressions $x \mid x$, $y \mid y$,… and etc.

For ours functions MDNF has view $f = hu \vee hu \vee \overline{\overline{z}}$. MDNF V basis {|} according to formulated algorithm accepts view $f = (x \mid y)((x \mid x) \mid (y \mid y)) \mid z$. Likewise we have

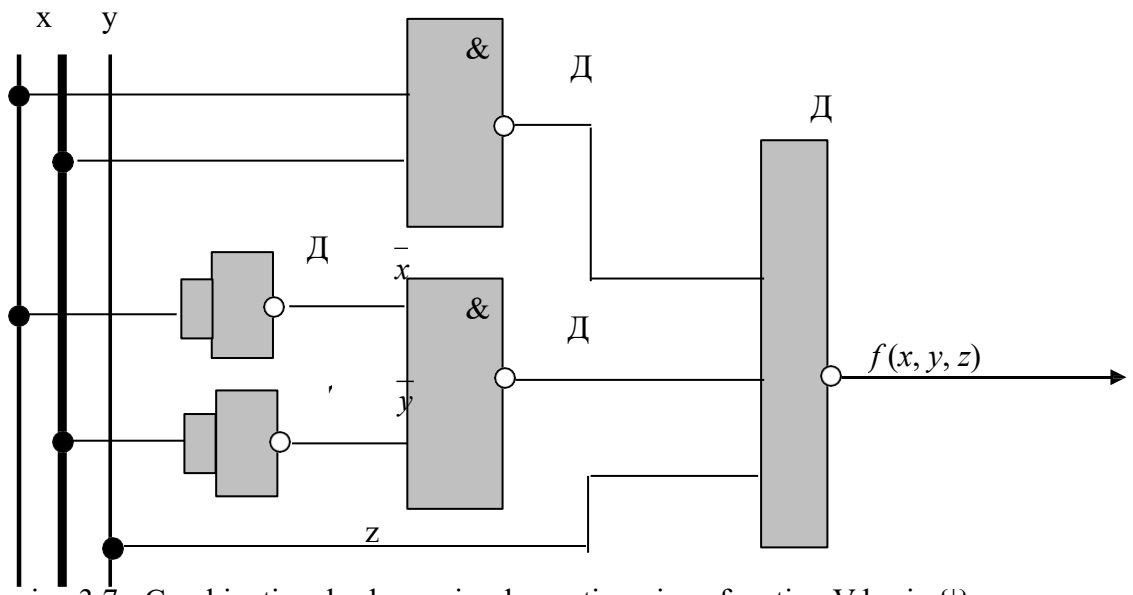
Rule 2. In order to obtain a representation of a logical function in the basis {↓}, it is sufficient to replace all the signs & and ∨ ιν ιτσ ΧΝΦ with a Peirce arrow and enclose in brackets the groups of letters included in each elementary disjunction.

The algorithm for finding the minimal representation of a function in the basis {↓} is based on rule 2 and includes the following steps: 1) determining the minimal finite-valued form of the function; 2) replacing everyone signs & And ∨ familiar ↓ And conclusion V brackets letters, incoming V elementary

disjunctions ICNF functions; 3) replacement denials arguments $x, y, \ldots$ expression species

$x \downarrow x$, $y \downarrow y$ And etc. For ours functions ICNF has view $f(x, y, z) = (X \vee \bar{a}t \vee \bar{z})(x \vee at \vee \bar{z})$) . The MCNF in the basis {↓} according to the above algorithm takes the form $f = (x \downarrow (y \downarrow y)) \downarrow ($ $z \downarrow z) \downarrow ((x \downarrow x) \downarrow y \downarrow (z \downarrow z))$.

KS this functions V basis {|} has view:



Drawing 3.7 - Combinational scheme, implementing given function V basis {|}

In Fig. 3.7, microcircuits D1-D2 are used to obtain the inversions of the signals x , y . Often, in the circuit preceding the synthesized circuit, not only variables are formed, but also their negations. In these cases, microcircuits D1-D2 can be excluded from the circuit. The expression, standing in the first in brackets is being implemented D3 microcircuit (2AND-NOT). On her inputs are supplied signals X And at . Expression, standing in second brackets, is being implemented

microcircuit D4 (2NAND). Signals are fed to its inputs X And y . Finally, the D5 (3I-NOT) microcircuit implements the entire function (the number in front of the letter I indicates the number of inputs V diagram).On drawing 3.8 shown synthesis KS V basis {↓}, implementing ours

function.

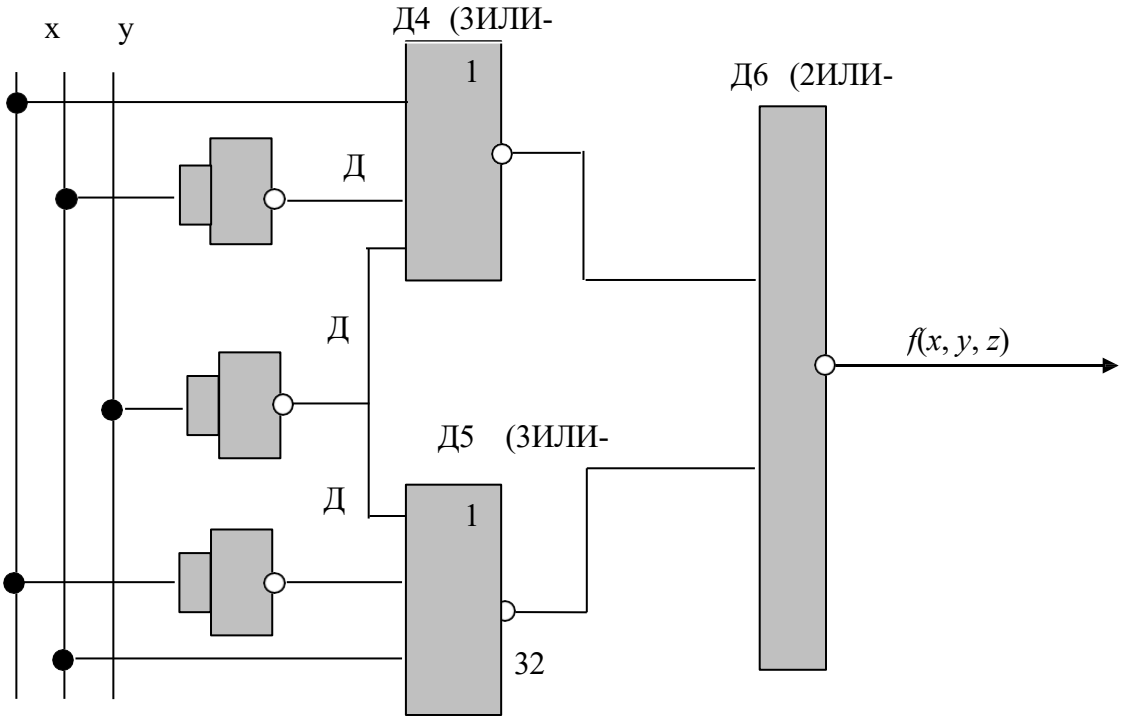


Figure 3.8 - Combination circuit implementing a given function in the basis {↓} Remains check right work schemes, presented drawings 3.7 And 3.8.

2.5. For these Boolean functions, construct the corresponding combinational circuits in redundancy functionally complete set {&, $\vee$, $\bar{}$ }, as well as in bases Pierce arrow { ↓} and Sheffer stroke {|} .

2.5.1. $f(x, y, z)$ defined so that $f(1,0,1) = f(1,1,0) = 1$, rest values functions f equal zero.

2.5.2. $f(x, y, z) = (\bar{x} \vee y)(zy \vee x)$.

2.5.3. $f(x, y, z) = (x \to y) \to (\bar{x}(y \vee z))$.

2.5.4. $f(x, y, z) = \bar{x}\bar{z} \vee \bar{x}yz \vee \bar{y}$.

2.5.5. $f(x, y, z) = \overline{xyz} \vee \overline{((x \vee y) \vee z)}$.

From point 2, in class it is recommended to solve problems 2.1, 2.3, 2.5.1, 2.5.2, 2.5.4 and analyze in detail the solution to problem 2.4.

Conclusion

We've examined the problem statement for synthesis of a control system and developed an algorithm for the structural synthesis process. The technology for synthesis of a control system itself depends on the selection of a functionally complete system of logical operations.

We did not touch on the synthesis of logic circuits with multiple outputs, that is, circuits defined by a system of Boolean functions. This material, like the synthesis of incompletely defined functions,

can be found in many books on discrete mathematics and digital circuit design. However, the material covered in this practical lesson allows us to solve synthesis problems for a fairly wide class of circuits.

It's worth noting that there are a number of circuit simulation systems designed to automate the modeling and analysis of fairly complex analog, digital, and digital-to-analog circuits on personal computers. Among these, the Electronics Workbench circuit simulation system occupies a special place, boasting a large library of common electronic components whose parameters can be adjusted over a wide range of values.

3.3 Exercise students For independent work

1. From this methodological developments decide tasks 1.4.1, 1.4.3, 2.2, 2.5.3 And 2.5.5.
2. Get ready To practical occupation By topic Applications "Propositional calculus to proof theory."

4 APPENDICES CALCULATIONS STATEMENTS TO THEORY EVIDENCE

Learning objectives: To master the basic principles of the axiomatic construction of formal theories using propositional calculus as an example; to learn to use the axioms and inference rules of propositional calculus to establish the provability of simple formulas of propositional calculus; and to acquire the skills of constructing derivations of simple formulas.

4.1 Educational questions classes:

1. Provability formulas
2. Construction conclusions from aggregates formulas

4.2 Questions For checks student readiness to class

1. Formulate main principles axiomatic constructions formal theory (calculus).
2. From what consists of alphabet propositional calculus ?
3. Give inductive definition formulas And subformulas propositional calculus .
4. Write down system axioms calculus statements. What such axiom scheme ?
5. Formulate main rules conclusion calculus statements (inference rule and substitution rule).
6. Give definition output formulas, theorems.
7. What such derivatives rules conclusion. Formulate some such rules.
8. Concept conclusion from aggregates formulas – This definition or theorem?
9. Formulate theorem deductions V calculus statements.
10. What kind theorems is expressed connection between algebra statements And propositional calculus ?

Historically, the concept of a formal theory was developed during the period of intensive research in the field of foundations of mathematics for the formalization of the proper logic and proof theory. This apparatus is now widely used in the development of specialized calculi for solving specific applied problems.

Truth tables in propositional algebra allow us to answer questions many questions concerning logic statements. However more complex questions logic statements can no longer be solved using truth tables. In describing propositional algebra, we used the logical values of statements (true, false). But the concepts of truth and falsehood are not mathematical. These concepts in In many cases, they are subjective and relate more to the methodology of knowledge. In this regard, it is desirable to construct a theory without using the concept of truth and falsity without applying the laws of logic. Such a theory, in particular, is calculus statements that are constructed according to general rules construction of formal axiomatic (deductive) systems (or calculi). Constructing such a formal system or theory is one way to model our reasoning in many practically important applications.

Formal theory is being built next way.

1. A set of formulas, or well-formed expressions, is defined, forming the language of the theory. This set is defined by constructive means (usually by inductive definition) and, therefore, is enumerable, and in most cases, decidable.
2. A subset of formulas, called the axioms of the theory, is identified that must be decidable.

3. They are asked rules of inference of the theory, that is, a computable relation on the set of formulas.

Process constructions theorems V such theories it's called *formal conclusion* or *proof* V formal theory.

One such formal theory is propositional calculus. Propositional calculus is an axiomatic a logical system whose interpretation is propositional algebra.

The description of any calculus includes a description of the symbols of this calculus (alphabet), formulas that are finite configurations of symbols, and a definition of the formulas to be derived.

The provability of propositional calculus formulas is based on its axioms and fundamental and derived rules of inference. Therefore, mastering the rules of inference in practice, especially the inference rule and the substitution rule, is of great importance.

The second important component of propositional calculus is the ability to construct a derivation for various formulas and to establish the derivability of formulas from a set of formulas.

Of particular importance for propositional calculus is the deduction theorem, which, on the one hand, serves as the basis for one of the methods of proving theorems of propositional calculus, and on the other hand, it is also used in other calculi.

1. Provability formulas

Alphabet, formulas And subformulas calculus statements

Alphabet calculus statements consists of from symbols three categories:

1. Symbols, which denote variables statements: lowercase letters of $x, y, z, \ldots, x_1, x_2, \ldots$ - the Latin alphabet (with or without indices).
2. Logical ligaments: &, $\vee$, $\rightarrow$, $\overline{\ }$.
3. Round brackets (,).

Others symbols alphabet calculus statements Not has.

Formulas calculus statements represent by yourself final Sequences of symbols of the propositional calculus alphabet. We will use capital letters of the Latin alphabet to denote formulas. These letters are not symbols of the propositional calculus alphabet, but merely represent conventional notations for formulas—that is, they are symbols of the so-called *metatheory* , which is used to describe propositional calculus.

A lot of formulas is usually given by an inductive definition using formal grammar. As a rule, this set is infinite.

Definition formulas:

1. All sorts of things variable statement $x, y, z, \ldots$ is formula.
2. If A And B - formulas, That formulas is And next combinations description : $(A \& B)$, $(A \vee B)$, $\overline{(A \rightarrow B)}$, (A) . . . .
3. No other combination of symbols is a formula. Variables statements

we will call *elementary formulas* .

Simultaneously With concept formulas is introduced concept *subformulas* or parts of the formula.

Definition subformulas:

1. Subformula elementary formulas is only she herself.
2. If formula has view A , That $\overline{\text{her}}$ subformulas are: she herself, formula A And All subformulas formulas A .

3. If formula has view $(A * B)$, Where under symbol $*$ understood any from logical ligaments $\vee$, &, $\rightarrow$, then her subformulas are: she herself, formulas A and B and that's all subformulas formulas A And IN.

Let's introduce V recording formulas some simplifications, A exactly: we will let down outer brackets And some internal, If subsequence execution of logical operations Not causes doubts in view of their priority: $\overline{\ }$, &, $\vee$, $\rightarrow$.

The next step in describing propositional calculus will be to identify a certain class of formulas, which we will call *initial provable ones.* (*derivable*) *formulas* or *axioms* of propositional calculus.

One of the possible systems of axioms for propositional calculus consists of 11 axioms, which are divided into four groups.

System axioms calculus statements

First group axioms:

I 1. $x \rightarrow (y \rightarrow x)$;

I 2. $(x \rightarrow (y \rightarrow z)) \rightarrow ((x \rightarrow y) \rightarrow (x \rightarrow z))$.

Second group of axioms:

II1. $x \& y \& x$;

II2. $x \& y \rightarrow y$;

II3. $(z \rightarrow x) \rightarrow ((z \rightarrow y) \rightarrow (z \rightarrow x \& y))$.

Third group axioms:

III 1 $x \rightarrow x \vee and$;

. III $and \rightarrow x \vee and$;

2.

III 3. $(x \rightarrow z) \rightarrow ((y \rightarrow z) \rightarrow (x \vee y \rightarrow z))$.

Fourth group axioms: IV 1.

$(x \rightarrow y) \rightarrow (\overline{y} \rightarrow \overline{x})$;

IV 2 $x \rightarrow \overline{\overline{x}}$;

. IV $x \rightarrow x$.

3.

All axioms first groups from logical ligaments contain only implication. This bunch enters And in All rest groups, But in the second group To implications are attached conjunction, V third group - disjunction, A V fourth - denial.

Main inference rules

Revealing the fact that from a set of statements (formulas of propositional calculus) some other statement logically follows (formula) and is, in essence, one of the main problems of propositional calculus.

The definition *of provable formulas* has the same character as the definition of a formula. After the introduction of the initial provable formulas (axioms), *the basic rules of inference are defined* , which allow new provable formulas to be obtained from existing provable formulas. Formation of a provable formulas from initial provable formulas by applying the rules of inference are called *the logical deduction* of a given formula, *proof* or *logical consequence* .

Rule conclusions. If formulas A and statements, then the formula B is also provable. Sc he m atically This rule are writing down So:

$A \to B$

provable V calculus

$$\frac{A,\ A\to}{B\ B}\ . \qquad (4.1)$$

IN literature provability formulas F more often total denote symbol $\vdash F$ or symbol $\models F$ (We we will use designation $\vdash F$). Then rule the conclusions can be written as:

$$\begin{array}{l}\vdash A,\ \vdash A\to B\\ \vdash B\end{array}\ .$$

Comma over feature means union "And".

Rule conclusions accepted call rule ***modus ponens***. IN this rule We we accept procedure transition from two formulas species A And $A\to B$ To one formula IN. IN conclusion By to this rule (That There is V application this rules) formulas A And $A\to B$ are *parcels*, A formula *IN* - *conclusion*.

In some sources the provability symbol is denoted by the symbol logical consequence «$\Rightarrow$» (record $P\Rightarrow Q$ means that formula Q *logically follows from formula* P, that is, that the implication $P\to Q$ is a tautology). The concept of logical consequence is commonly used in logic programming and artificial intelligence to describe logical inference methods. Note that implication is sometimes denoted by the symbol «$\supset$".

In proof theory, in most cases the rule *modus ponens* (or the separation rule, as it is also called) is accepted as the only basic rule of inference. However, often another basic rule of inference is introduced *substitution* rule.

Substitution rule. If a formula A is provable in propositional calculus, x is a variable, B is an arbitrary formula in propositional calculus, then the formula, obtained by replacing the variable x in formula A everywhere where it appears with the formula IN, is also a provable formula.

Symbolically This rule Can write down V view

$$\frac{\vdash A(x)}{\vdash A(B)}\ . \qquad (4.2)$$

The formulas used in rules (4.1) and (4.2) are called *metaformulas* or *formula schemes*. The meaning of metaformulas is that they denote the set of all formulas of a calculus that are obtained if its metavariables are considered formulas of the calculus. The use of formula schemes can be extended to axioms. For example, if in the first group of axioms the propositional variables x, y, z replacing *X with metavariables*, Y, Z, That it turns out two schemes axioms, asking three endless sets axioms. As a result, another way of constructing a propositional calculus with an infinite set of axioms (defined by a finite number of schemata) arises, but without a substitution rule, since it is implicitly contained in the interpretation of axiom schemes. The first method, that is, the method we used, is more constructive: all its means are clearly fixed and finite. On the other hand, the second method is more in line with mathematical traditions interpretations formulas: For example, algebraic identity $(x+y)^2=x^2+2xy+y^2$ or identities Boolean algebra are interpreted exactly as schemes of identities, rather than specific identities that are true only for specific letters.

Now some let's clarify rule substitutions.

A) If the formula A There is variable x, That substitution (4.2) gives formula B.

b) If formula A There is variable $y\neq x$, then the substitution (4.2) gives formula A.

V) If A - formula, For which substitution already determined, That substitution *IN* instead of x V negation formulas A there is a negation of the substitution.

G) If A_1 And A_2 - formulas, For which substitutions already defined, That

$$\frac{\vdash A_1 * A_2}{\vdash A_1(B) * A_2(B)},$$

Where under symbol $*$ understood any from logical ligaments $\vee$, &, $\rightarrow$

Definition of provable (deducible) formulas

1. Each one axiom is provable formula.
2. Formula IN, received from provable formulas A and $A \rightarrow B$ by way applications rules conclusions, There is provable formula.
3. The formula obtained from the provable formula by applying the substitution rule instead of the variable x of an arbitrary formula B is provable formula.
4. No other formula of propositional calculus is considered provable.

Process receipt provable formulas it's called *proof*.

Derivatives rules conclusion

Let us denote by **T** is a formal theory of propositional calculus. Then, by a proof of formula B in theory **T** we will mean a derivation in which only axioms are used as initial formulas. A formula B for which a proof exists is called a formula *provable* in theory **T** or a theorem of theory **T**, and the fact that formula B *is provable* we will, as before, denote by the symbol $\vdash B$.

However, the derivation of formula B can be carried out Not only from axioms but also from other formulas, that is, from a set of a certain number of formulas that are not necessarily just axioms. Thus, the concept of proof is a special case of the concept of derivability. Along this path, with the further development of the theory of propositional calculus, we can obtain a number of new rules of inference, distinct from the rules of inference and substitution, which in turn can be used to prove most of the laws of logic and establish a connection between propositional algebra and propositional calculus. These rules (that is, rules distinct from axioms) are called *derived rules of inference* . Let us consider the most important derived rules of inference.

Rule simultaneous substitutions : let A - provable formula; $x_1, x_2, ..., x_n$ - variables statements, A $B_1, B_2, ..., B_n$ - any formulas theories **T**. Then result simultaneous substitutions V formula A instead of variables $x_1, x_2, ..., x_n$ respectively formulas $B_1, B_2, ..., B_n$ is provable formula theories **T**.

Rule complex conclusions:

$$\vdash A_1, \vdash A_2, \ldots, \vdash A_n, \vdash A_1 \rightarrow (A_2 \rightarrow (A_3 \rightarrow (\ldots A_n \rightarrow Z)\ldots)))$$
$$\vdash Z.$$

The rules of simultaneous substitution and complex inference are a natural generalization of the rules of substitution and inference, respectively. in the case of n formulas.

Rule syllogism:

$$\frac{\rightarrow B, \vdash B \rightarrow C}{\vdash A \rightarrow C}.$$

Rule contrapositions:

$$\vdash A \rightarrow B$$
$$\vdash \overline{IN} \rightarrow \overline{A}.$$

Rules removals double denials:

$$\frac{\vdash A \to \overline{\overline{B}}}{\vdash A \to B}, \qquad \frac{\overline{\overline{A}} \to B}{\vdash A \to B}.$$

Tasks

1.1. Which from following expressions are formulas calculus statements:

1.1.1. $(\overline{P}_1 \& \overline{P}_2) \to (P_1 \vee P_2)$;

1.1.2. $((P_1 \vee P_2) \vee (P_1 P_2)) \to \overline{P_2}$;

1.1.3. $(P_1 \to (P_2 \vee P_3)) \to P_3$;

1.1.4. $(P_1 \to P_2) \to ((P_1 \to \overline{P}_2) \to P_1)$;

1.1.5. $(P_1 \& (\to P_2)) \to (P_2 \to \overline{P}_1)$;

1.1.6. $(P_1 \to P_3)((P_2 \to P_3) \to ((P_1 \vee P_2) \to P_3))$;

1.1.7. $((P_1 \to \overline{P}_2) \& (P_1 \to P_3)) \to (P_1 \to (P_2 \& P_3))$;

1.1.8. $((P_1 \& P_2) \to (\overline{\overline{P}}_1 \vee \vee P_2)) \leftrightarrow (\vee P_1 \vee P_2)$.

Answer : expressions 1.1.2, 1.1.5 And 1.1.8 formulas Not are.

1.2. Write out All subformulas formulas:

1.2.1. $x \to (y \to x)$;

1.2.2. $a \& \overline{c \vee b}$;

1.2.3. $x \vee y \& z \to x$;

1.2.4. $\overline{x \to y \vee x \& y}$;

1.2.5. $\bar{t}$;

1.2.6. $(x \to y) \to (x \to \overline{y}) \to \overline{y}$.

1.3. Install provability formulas:

1.3.1. $A \& B \to A \& B \vee C$;

1.3.2. $(A \& B \to (C \to B \& C)) \to ((A \& B \to C) \to (A \& B \to B \& C))$;

1.3.3. $A \vee A \to A$;

1.3.4. . $A \& B \to B \& A$;

1.3.5. . $A \vee B \to B \vee A$

Instructions . 1.3.1. B axiom III $_1$ perform a substitution instead of x $A \& B$, instead of y – C.

1.3.2. TO provability leads production V axiom I $_2$ instead of x, y, z respectively formulas $A \& B$, C, $B \& C$.

1.3.3. In axiom III $_3$ substitute formula A *for* x, y, z. We obtain a provable formula $(A \to A) \to ((A \to A) \to (A \vee A \to A))$. Then apply the rule of complexity conclusions

1.3.4. IN axiom II $_3$ substitute instead of x, y, z respectively formulas B, A, $A \& B$. Then apply the rule of complex inference again.

1.3.5. IN axiom III $_3$ substitute instead of x, y, z respectively formulas A, B, use $B \vee A$ And the rule of complex inference.

1.4. Using derivatives rules conclusion show, What provable the following formulas:

1.4.1. $\overline{A} \vee \overline{B} \to \overline{A \& B}$;

1.4 2. $(A \to B) \& \overline{B} \to \overline{A}$;

1.4.3. $A \& B \to \overline{A \vee B}$.

Instructions . 1.4.1. IN axiom III $_3$ instead of x , y , z substitute respectively $\overline{A}$, $\overline{B}$, $\overline{A \& B}$ And further take advantage of axioms II $_1$, II $_2$, rules contrapositions and a complex conclusion.

1.4.2. To implement sequentially substitutions: V axiom I $_2$ instead x , y , z respectively $(A \to B) \& B$, $\overline{B}$, $\overline{A}$, V axiom II $_2$ - instead of x , y respectively $A \to B$ And $\overline{B}$, V axiom II $_1$ - instead of x , y respectively $A \to B$ And $\overline{B}$ And take advantage of rule of complex inference.

1.4.3. IN axiom III $_3$ do substitution, replacing V her z on $\overline{\overline{x} \& \overline{y}}$ and further take advantage of axioms II $_1$, II $_2$ And rules contrapositions, removals double negation and complex inference.

1.5. Applying rule substitutions, install provability formulas:

1.5.1. $(A\&B) \to ((A\&B) \vee C)$;

1.5.2. $\overline{\overline{P \vee Q}} \to P \vee Q$.

1.6. Applying the substitution rule and *the modus ponens rule* , establish the provability of formulas:

1.6.1. $(A \to B) \to (A \to A)$;

1.6.2. $P \to P \& Q$.

1.7. Show, What next formulas are provable:

1.7.1. $a \to (\overline{a} \& b)$; 1.7.2. $p \vee \overline{p}$; 1.7.3. $\overline{r} \& \overline{s} \to \overline{r \vee s}$.

1.8. Prove derivatives rules conclusion:

1.8.1. $\dfrac{\vdash A \to B, \ \vdash \overline{B}}{\vdash \overline{A}}$;

1.8.2. $\dfrac{\vdash A, \vdash \overline{B}}{\vdash \overline{\overline{A \to B}}}$;

1.8.3. $\dfrac{\vdash A \to B, \ \vdash A \to \overline{B}}{\vdash \overline{A}}$;

1.8.4. $\dfrac{\vdash P \to Q, \vdash \overline{P} \to Q}{\vdash Q}$.

1.9. Prove:

1.9.1. Rule syllogism.

1.9.2. Rule contrapositions.

1.9.3. Rules removals double negative.

Let's prove it V quality example rule contrapositions, which lies V basis a powerful method of proving theorems - *the method by contradiction* . To prove the axiom IV $_1$ $((X \to y) \to (\overline{at} \to X))$ let's do it simultaneous substitution: instead of X let's substitute formula A , A instead of $\overline{at}$ - formula IN . We will receive provable formula $\vdash (A \to IN) \to (\overline{IN} \to A$

)

. From parcels rules contrapositions Formula $A \to In$ provable, That's why, By applying the rule of *modus ponens* , we obtain the conclusion of the rule of contraposition.

1.10. Using derivatives rules conclusion show, What provable the following formulas:

1.10.1. $\overline{A} \vee \overline{B} \to \overline{A \,\&\, B}$;

1.10.2. $(A \to B) \,\&\, \overline{B} \to \overline{A}$;

1.10.3. $A \,\&\, B \to \overline{A \vee B}$.

Instructions . 1.10.1. IN axiom III $_3$ instead of x , y , z substitute respectively formulas $\overline{A}$, $\overline{B}$, $\overline{A \,\&\, B}$. Then apply To axioms II $_1$ And II $_2$ rule contrapositions, after what take advantage of rule complex conclusions.

1.10.2. IN axiom I $_2$ substitute instead of x , y , z according to the formula $(A \to B) \,\&\, \overline{B}$, $\overline{B}$, $\overline{A}$, V axiom II $_2$ substitute instead of x , y respectively formulas $A \to B$ And $\overline{B}$, V axiom II $_1$ substitute instead of x , y respectively formulas $A \to B$ And B , then apply the rule We will receive a complex conclusion.

1.10.3. IN axiom III $_3$ replace z on $x \,\&\, y$ And apply axioms II $_1$ And II $_2$. After this apply consistently contraposition rule, double removal rule negation, the rule of complex inference and again the rule of contraposition and the rule of removing double negation.

2. Construction conclusions from aggregates formulas

Let's consider a finite set of formulas $\Gamma = \{ A_1, A_2, \ldots, A_n \}$.

The derivation of formula B from the formulas of the set Γ is called the sequence of formulas $F_1, F_2, \ldots, F_m$ such that F_m = B, and any of the formulas $F_1, F_2, \ldots, F_m$ either an axiom or one from the original formulas $A_1, A_2, \ldots, A_n$, or directly derivable from the formulas preceding it, according to one of the rules of derivation.

If there is a derivation of formula *B* from a set of formulas Γ , then they say that the formula *IN deducible* from aggregates formulas Γ And are writing down $\Gamma \vdash In$ or $A_1, A_2, \ldots, A_n \vdash V$. Formulas $A_1, A_2, \ldots, A_n$ are called *hypotheses* or *premises of the inference* . The transition in the inference F_{i-1} To F_i is called *the i* -th step of inference. Obviously, adding formulas to hypotheses does not violate derivability. Therefore, if $\vdash B$, then $A \vdash B$, and if $A_1, A_2, \ldots, A_n \vdash B$, then $A_1, A_2, \ldots, A_n, A_{n+1} \vdash B$.

At an intuitive level, in relation to some concrete and meaningful theory **T,** the conclusion is nothing more than obtaining V in the process of proving some statement (theorem, lemma, formula, etc.) intermediate statements, based either on axioms or on previously obtained statements according to existing rules of inference (inferences).

Theorem deductions . Let Γ - set formulas, A and B - formulas. Then

$$\Gamma, A \vdash \text{IN}$$
$$\Gamma \vdash A \to IN.$$

Rule introduction conjunctions:

$$\Gamma \vdash A, \Gamma \vdash IN$$
$$\Gamma \vdash A \,\&\, IN \ .$$

Rule introduction disjunctions:

$$\Gamma, A \vdash WITH, \Gamma, IN \vdash WITH$$
$$\Gamma, A \vee IN \vdash WITH \ .$$

Law permutations parcels : $\vdash (x \to (y \to z)) \to (y \to (x \to z))$.

Law connections parcels: $\vdash (x \to (y \to z)) \to (x \& y \to z)$.

Law disconnection parcels : $\vdash (x \& y \to z) \to (x \to (y \to z))$.

Tasks

2.1. Write down conclusion For the following formulas:

2.1.1. N = $\{ A, B \}$, $\mid$ N - $A \& B$;

2.1.2. G = $\{ A \to B, B \to C \}$, G / $AND \to C$;

2.1.3. G = $\{ \overline{\overline{A}}nd, AND \to \neg B \}$, G $- B$;

2.1.4. H = $\{ A \to (B \to C) \} \mid$, H $- B \to (A \to C)$.

IN quality example let's write it down conclusion For tasks 2.1.1.

$F_1 \equiv A$; $F_2 \equiv B$; $F_3 \equiv (A \to A) \to ((A \to B) \to (A \to A \& B))$ (substitution in axiom II $_3$ instead of x, y, z respectively formulas A, IN, A); $F_4 \equiv A \to A$ (How provable formula); $F_5 \equiv (A \to B) \to (A \to A \& B)$ (By rule conclusions from F_3 And F_4; $F_6 \equiv B \to (A \to B)$ (substitution V axiom I $_1$ instead of x, y respectively IN And A); $F_7 \equiv A \to B$ (By rule conclusions from F_2 And F_6); $F_8 \equiv A \to A \& B$ (By rule conclusions from F_5 And F_7); $F_9 \equiv A \& B$
(By rule conclusions from F_1 And F_8).

2.2. Prove, What If G = $\{ A \to B \}$, That $\mid\!-_G (B \to C) \to (A \to C)$.

Instruction. Use rule syllogism And theorem deduction.

2.3. Prove rules introduction conjunctions, disjunctions And laws permutations, connections and disconnections of parcels

Conclusion

The problems considered, related to propositional calculus, allow one to master individual logical rules of inference for proving formulas and constructing conclusions for various formulas.

Exercise students For independent work

1. Decide tasks With odd numbers.
2. By alone from literary sources disassemble proof theorem deduction.
3. Get ready To practical occupation By topic "Method resolutions V "propositional calculus ."

5 METHOD RESOLUTIONS IN CALCULATION STATEMENTS

Learning objectives **:** to expand and deepen students' knowledge gained in lectures, which can be used in solving problems using the resolution method in propositional calculus; to bring the symbolism and terminology of the resolution method as close as possible to the language of logical programming.

5.1 Educational questions classes:

3. Proof truth logical Clause method resolutions.
4. Main principles logical programming.

5.2 Questions For checks student readiness to class

1. What kind properties possesses attitude order?
2. What understood under *letter* And *atom* ?
3. What V logical conclusion understand under *clause* and *a legend* ?
4. How is different *Horn* clause from others types of clauses?
5. Give definitions *Khornovsky disjunct* , sets disjuncts.
6. What such Erbranovsky *universe* » this sets disjuncts?
7. Give definition *interpretations* sets disjuncts over Herbrand's universe.
8. Formulate theorem Herbrana O impracticability sets disjuncts.
9. What it's called *resolvent* two disjuncts?
10. Formulate rule resolutions For calculus statements.
11. How is determined *logical investigation* disjuncts through their resolution?
12. How it's called conclusion empty disjunct from this sets disjuncts?
13. From what kind types proposals consists of program on language programming Prologue?

In Boolean algebra, all proofs were based on *the equivalence relation* . Two logical functions were considered equivalent if they yielded absolutely identical values of zeros and ones on the corresponding sets of arguments. When using the formal notation of logical expressions, individual links in the chain of any proof were connected by the equality symbol "=". In propositional calculus, all proofs are based on *the order relation* , which is established between cause and effect. Here, individual links in the chain of evidence are connected by the implication symbol "→". As in Boolean algebra, in propositional calculus there are axiomatic and constructive approaches to proving logical expressions. *Axiomatic* construction calculus statements consists of volume, to attempt to isolate from the infinite number of true formulas an independent system of axioms by which the validity of any other formulas could be established. Propositional calculus, as we already know, is an extension of propositional algebra. In other words, propositional algebra serves as one interpretation of propositional calculus.

The constructive approach to proving logical expressions includes the method of compiling truth tables.

In addition to the axiomatic and constructive methods of proving logical formulas, there are also *mixed* proof methods, among which *the resolution method occupies a special place* . A resolution rule is an inference rule that generates resolutions for a set of clauses.

To prove theorems using the resolution rule, the original is transformed formula To set Hornovskys disjuncts S_0. Next are building all sorts

resolvents $R = \{ R_1, R_2, ..., R_n \}$ from disjuncts, belonging set S_0. Next we get set $S_1 = S_0 \cup R$ And let's repeat All at first to those pores, Bye Not we will receive set S_m, containing empty disjunct.

1. Proof truth logical Clause method resolutions

In logical inference, we will replace the implication symbol "→" symbol " ⇒ ". The symbol "→" is *objective* , and the symbol " ⇒ " is *subjective.* Thus, it is necessary to distinguish language logic statements and *metalanguage* researcher. In avoid To avoid confusion, we will introduce two more *metasymbols* : instead of *the objective conjunction* "&", we will use *the subjective metaconjunction* symbol ",", and instead of *the objective disjunction* " ∨ " - *the subjective metadisjunction* " ; ". Then the statement that needs to be proved is formalized in the propositional calculus as the following *cause-and-effect relation* :

$$P_1, P_2, ..., P_{n-1}, P_n \Rightarrow C, \qquad (5.1)$$

Where P_i - *package* (*reason*), C - *conclusion* (*consequence*). Reads: "If parcels $P_1, P_2, ..., P_{n-1}, P_n$ true, That conclusion C Same truly" or, in another way: "If reasons $P_1, P_2, ..., P_{n-1}, P_n$ had place, That will have place And investigation C ".

To avoid confusing *the objective statement* (sentence) with *the subjective statement* , the validity of which we intend to establish, we will agree to call sentences of type (5.1) ***a clause*** . A clause is *a meta-sentence* in which an order relation is used, formalized through the symbol of meta-implication. " ⇒ " Like the equivalence relation, the order relation satisfies three axioms (laws):

reflexivity : $A \Rightarrow A$;

antisymmetry: if $A \Rightarrow B$, That $\overline{B} \Rightarrow \overline{A}$;

transitivity: if $A \Rightarrow B$ And $B \Rightarrow C$, That $A \Rightarrow C$.

Law of antisymmetry you can write it down And So: If $A \Rightarrow B$ And $B \Rightarrow A$, That $A = B$.

Clause There is exactly formal recording provable offers. Instead of letters V It is possible to substitute objective statements into it, and then the clause is filled with specific content, which is called *semantics* or ***legend*** . An example of a clause:

$A \to B$, $A \Rightarrow B$.

If we accept that A = lightning flashed, B = thunder struck, then the following legend can be made: it is known that if lightning flashes, thunder will follow. Lightning flashed. Therefore, thunder must also strike.

Above the subject that formulates meta-sentences there may be another subject, for whom already offers first subject will turn out to be objective. Then clause (5.1) the second subject or metasubject will write for itself the following logical expression:

$$(P_1 \& P_2 \& ... \& P_{n-1} \& P_n) \to C .$$

Having transformed This expression V disjunction, we get:

$$\overline{P}_1 \vee \overline{P}_2 \vee ... \vee \overline{P}_{n-1} \vee \overline{P}_n \vee C .$$

From here easily we find:

$$(P_1 \& P_2 \& ... \& P_{n-1}) \to (\overline{P}_n \vee C).$$

That's why Clause (5.1) Maybe be presented V another equivalent form:

$$P_1, P_2, ..., P_{n-1} \Rightarrow \overline{P}_n ; C. \qquad (5.2)$$

IN force commutativity conjunctions on place parcels P_n can turn out to be any other, and not just one. For example, a clause

$$P_1, P_2, P_3, P_4 \Rightarrow C_1; C_2; C_3$$

Maybe be transformed V another, equivalent form:

$$P_4, \overline{C_2}, P_1, \overline{C_1} \Rightarrow \overline{P_1}; C_3; \overline{P_2}. \qquad (5.3)$$

However, clause (5.1), compared to (5.2) and other similar forms like (5.3), has certain advantages and is used, in particular, in the logical programming language PROLOG. It is called *a Horn clause* . Any clause can always be reduced to a Horn clause by equivalent transformations.

If the symbol of metaimplication I f t h e " ⇒ " clause (5.2) is shifted to the extreme left, it will become a *tautology* ; if it is shifted to the extreme right, it will become a *contradiction* :

$1 \Rightarrow \underline{P}_1; \underline{P}_2; ...; \underline{P}_{n-1}; P_n; C$ – tautology,

$P_1, P_2, ..., Pn_{-1}, \overline{Pn}, C \Rightarrow 0$ – contradiction.

Let us remind you Now rule (principle) resolutions For calculus statements:

If for any two clauses C_1 and C_2 there is a letter L_1 in clause C_1 , which is inverse (contrary) to the letter L_2 in C_2 , then, having crossed out L_1 and L_2 from C_1 and C_2 respectively, построим дизъюнкцию оставшихся дизъюнктов. Построенный дизъюнкт называется **резольвентой** дизъюнктов C_1 and C_2 .

The following theorem holds: let two clauses $C_{1 \text{ be given}}$ and C_2 . Then the resolution C of clauses C_1 and C_2 there is a logical consequence of the clauses C_1 and C_2 .

The method of proving the truth of logical clauses using *the resolution principle* is a semi-constructive proof method. The resolution principle plays the role of an axiom of order and at the same time generates a very effective constructive procedure. Thus, the essence of this principle comes down to the fact that two parcels disjunct With opposite letters (terms) Always Can glue in one final disjunct (resolution) in which there will no longer be opposite letters:

$$X \vee TO, AND \vee A \Rightarrow X \vee AND,$$

Where X And Y - arbitrary baths or disjuncts With any set terms, including empty disjunct $\oslash$ (V further We his we will designate zero "0"); A And $\overline{A}$ - arbitrary terms.

With the consistent application of the resolution method, the number of letters decreases, eventually disappearing entirely. In this case, the initial clause is constructed in the form of a conjunctive contradiction:

$P_1, P_2, ..., P_n \Rightarrow 0.$

Some of the inference rules of the propositional calculus we've considered are special cases of the resolution rule. For example, using the resolution method, we'll prove the *modus ponens rule* , which, in the notation we've introduced here, is written as follows:

$$Oh, A \to B \Rightarrow B \text{ or } 0 \vee Oh, \underline{A} \vee B, \underline{B} \vee 0 \Rightarrow 0.$$

There are three clauses here. By combining them sequentially, we obtain an empty clause (0), which indicates the incompatibility of the conclusion and premises. This is precisely what confirms the truth of the original clause.

The resolution principle completely replaces the axiom of order, since this axiom itself can be proven within the framework of the resolution method. Indeed,

$$A, B \Rightarrow A, \quad A, B, \overline{A} \Rightarrow 0, \quad 0, B \Rightarrow 0.$$

It's worth noting that premise B isn't used at all here. This is important to keep in mind: it's not necessary to use all the premises (their number is often excessive) – the main thing is to obtain an empty disjunct.

Let given Clause $\overline{A} \to B,\ C \vee A,\ B \to C$ Proof her justice $\Rightarrow A$.

follows begin With ghosts her V NKF: $A \vee B,\ C \vee A,\ B \vee \overline{C},\ \overline{A} \Rightarrow 0.$

Let's write out all the premises in order and then begin to glue them together in order. To the right of each new clause, we'll write the numbers of the clauses used. We'll get the following chain of derivation for the empty clause:

1. $A \vee B$,
2. $C \vee A$,
3. $\overline{B} \vee \overline{C}$,
4. $\overline{A}$,
5. $A \vee \overline{C}$, (1, 3)
6. B, (1, 4)
7. $A \vee \overline{B}$, (2, 3)
8. C, (2, 4)
9. A, (2, 5)
10. $\overline{C}$, (3, 6)
11. $\overline{B}$, (3, 8)
12. $\overline{C}$, (4, 5)
13. $\overline{B}$, (4, 7)
14. 0. (4, 9)

This strategy for searching for an empty clause is very unproductive. If you approach the same problem more creatively, the empty clause will be discovered in the fourth step of the search:

5. B (1, 4), 6. C (2, 4), 7. $\overline{B}$ (3, 6), 8. 0 (5, 7).

So far, we've used the *conjunctive contradiction form* . Based on *the duality principle* , the resolution method can be formulated relative to *disjunctive tautology* , with the resolution principle naturally changing:

$X \,\&\, Y \Rightarrow X \,\&\, A;\ Y \,\&\, \overline{A}$.

Let's prove it ode And that same clause two ways - V form contradictions And V form tautologies. Let given Clause: $A,\ A \to B,\ B \to (A \to C) \Rightarrow C$.

Proof V form contradictions looks like next way:

$A,\ \overline{A} \vee B,\ \overline{B} \vee \overline{A} \vee C,\ \overline{C} \Rightarrow 0.$

1 2 3 4

Gluing : 5. B (1, 2), 6. $\overline{A} \vee C$ (1, 6), 7. C (3, 5), 8. 0 (4, 7).

Proof V form tautologies looks like similarly:

$1 \Rightarrow \overline{A};\ A \,\&\, \overline{B};\ B \,\&\, A \,\&\, \overline{C};\ C.$

1 2 3 4

Склейки: 5. $\overline{B}$ (1, 2), 6. $A \,\&\, \overline{C}$ (1, 6), 7. $\overline{C}$ (3, 5), 8. 1 (4, 7).

One of the disadvantages of the resolution method is that the original clause is necessarily should be presented V normal disjunctive or conjunctive form.

Tasks

1. Given below clauses prove method resolutions:

1.1. $A \to B,\ C \to D,\ B \to E,\ D \to F,\ E \to \overline{F},\ A \to C \Rightarrow \overline{A}.$

1.2. $(A \to C) \to (\overline{A}\ \&\ B) \Rightarrow A \vee B.$

1.3. $A \vee D,\ B \vee E,\ D \to C,\ D \vee C \Rightarrow A\ \&\ C;\ E\ \&\ D;\ B.$

1.4. $A \to B, C \to D, A \vee C, A \to \overline{D}, C \to \overline{B} \Rightarrow (A \vee B) \to (A\,\&\,B).$

Example proof Clause V form contradictions And V form tautologies given above.

2. Compose legends For following Clause:

2.1. $A \to B,\ B \to E,\ A \to C,\ C \to D,\ D \to F,\ \overline{E\,\&\,F} \Rightarrow \overline{A}.$

2.2. $J \to H,\ K \to H,\ I \to J,\ H \to I, \overline{H} \Rightarrow \overline{J}\ \&\ \overline{K}.$

Let us give example legends For clauses 2.1:

If a person is involved in sports (A), then he wants to be healthy (B). Good health (B) leads to a happy life (E). In addition, if a person is involved in sports (A), then, as a rule, he strives to achieve high athletic results (C). Having high results (C) allows one to win competitions (D). Victories in competitions (D) attract for itself is universal confession (F). However Human Not wants live happily and have universal recognition ($\overline{E\,\&\,F}$). Therefore, he will not do sports. (A) ».

3. For given legends make up clauses, responsible text or context these legends, for which it is necessary to formulate the necessary premises and two consequences: one true, the other false.

If there's a large voltage drop in the circuit, the fuse will blow, requiring replacement. If the fuse is intact, the TV will still work, but only if it's plugged in. If the TV works Fine, That I I will see today's "News" So, I I'm watching television "News" provided there is no voltage drop and the TV is connected to power supply networks.

Solution . If enter designations: A - emergence difference voltage V networks; B - the fuse has blown; C - the fuse needs to be replaced; D - the TV is working properly; E - the TV is connected to the power supply; F - I am watching the News, then for legend 3.1 you can create the following clause:

$$A \to B,\ B \to C,\ E \to (\overline{B} \to D),\ D \to F \Rightarrow (\overline{B}\,\&\,\overline{A}\,\&\,E) \to F.$$

In this clause the conclusion is true, while the conclusion given in the legend is $((A\,\&\,E) \to F)$, is false.

A decrease in temperature leads to a decrease in pressure and a decrease in volume. An increase in volume leads to an increase in flow velocity. An increase in pressure leads to a decrease in level, if the temperature is also decreased. A decrease in velocity leads to a decrease in pressure. or an increase in temperature. Technologist Ivanov reasoned: "I need to increase the pressure while simultaneously reducing the flow rate, so I must increase the volume and temperature."

I didn't notice the motorcycle at first, because it was hidden by a fuel truck, and the Volga turned the corner when the Zhiguli was already near the traffic light. The foreign car ran the red light. light And appeared, How to me Seems, reason all this accidents. Because of her Volga

braked sharply, and the motorcyclist ended up on the asphalt. The Zhiguli, in order to avoid running over the motorcyclist, turned away on sidewalk, A fuel tanker V This time crashed V Volga. If would If there had been no motorcyclist, the dangerous situation might not have occurred. Although the driver is also at fault.
Volga, because He obviously exceeded the speed limit.

3.4. Increasing the complexity of a device's circuitry will increase its performance, and using new components will extend its service life. Only when both its performance and service life increase will the device sell well. However, demand for the device is low.

3.5. An increase in money in circulation leads to inflation. But the money supply grows for two reasons: through money emission or a decline in trade turnover. A decline in trade turnover leads to unemployment and a decline in production. Inflation causes the exchange rate of the currency to fall. Economist Ivanov's recommendation: increase money emission and boost production, then we'll avoid unemployment, and the exchange rate will remain unchanged.

3.6. If something decreases in one place, something will increase in another—that's a self-evident truth. But there's a theory that claims that somewhere in deep space, there are "black holes" into which everything falls, but nothing comes out. does not appear. This theory says nothing about the existence of "white holes" that would act opposite to "black holes." One foreign astronomical journal reported coordinates "black hole." Russian astronomer Petrov directed He pointed his powerful telescope there and found nothing. "Well, well," said Petrov, "but I'll still discover the 'white hole'."

2. Main principles logical programming

At the initial stage of artificial intelligence (AI) development, languages and systems specifically designed for the creation of knowledge-based applications did not exist. On the one hand, the approach itself, in which knowledge was central, had not yet taken shape. place was allocated would presentation theories in form programs, and with Another was that the field of AI itself was just emerging as a scientific field. It was also significant that the high-level universal programming languages that had emerged by that time seemed adequate. a tool for creating any, including and intelligent systems. However, the complexity and labor-intensive nature of development here are so great that practically useful intelligent systems become unachievable. Given this, symbolic information processing languages and systems were developed, which became the primary programming tool for intelligent systems for several decades.

Until recently, *Lisp was the most popular base language for implementing AI systems in general and expert systems (ES) in particular* . This language was developed at Stanford under the leadership of J. McCarthy in the early 1960s and was not initially intended for programming AI tasks. It was a language that was supposed to be the next step after *Fortran* in the automation of programming. By the time the first *Lisp* interpreters were created, interactive mode was becoming commonplace on computers, and the interpretation mode naturally fit into the general structure of interactive work. Around the same time, the principles underlying the *Lisp language were finally formed* : the use of a unified list representation for programs and data; the use of expressions to define functions; and the language's bracket syntax. The development of the language culminated in the creation of version *Lisp 1.5, which determined the path of its development for many years. In parallel with the development of the Lisp* language and its various versions, other programming systems for AI were also being developed, among which we note the languages *Snobol* and *Refal* .

In the early 1970s, another new language emerged that could compete with *Lisp* in the implementation of systems oriented towards knowledge, is the *Prolog language* . It does not provide new super-powerful programming tools compared to *Lisp* , but supports a different model for organizing computations. Just as *Lisp* hid the structure of computer memory from the programmer, PROLOG allowed him not to worry (unnecessarily) about the flow of control in the program. PROLOG offers a paradigm of thinking in which the description of the problem being solved is presented in the form of a weakly structured set of relations. This is convenient if the number of relations is not too large and each relation is described by a small number of alternatives. Otherwise, a PROLOG program becomes very difficult to understand and modify. Problems with the efficiency of language implementation also arise, since in the general case the inference mechanisms built into PROLOG provide a solution search based on enumerating possible alternatives and declarative return from dead ends. It belongs more to *relational* or *declarative* (*descriptive*) languages, rather than to functional languages, and is a programming language of logical type. The logical basis of the PROLOG language is *unification* and *the resolution method* .

Let us explain the application of the principle of resolutions using an example and unification of the inference procedure in the PROLOG language.

Let us write the expression " X is a footballer" as footballer (X). Here, footballer is a predicate letter, and X is a variable term. Furthermore, we write the expression " *X is an athlete* ." in the form of sportsman (X). Thus, the expression "a football player is an athlete" can be written as footballer (X) → sportsman (X).

Let us now try, given the expression "Pele is a footballer", that is, *footballer* (Pele), where Pele is a constant term (or simply a constant), to derive the expression *sportsman* (Pele) using the resolution method.

Before in total, let's add To original set formulas negation statements, which We we want prove, That There is $\overline{sportsman(\text{Pele})}$:

$$\overline{footballer(X)} \vee sportsman(X) \qquad (5.4)$$

$$\text{footballer (Pele)} \qquad (5.5)$$

$$\overline{sportsman\ (\text{Pele})}. \qquad (5.6)$$

Formula $\overline{footballer(X)} \vee sportsman(X)$ logically equivalent formula footballer (X) → sportsman (X).

That's why necessary to carry out unifying substitution X = Pele into disjunction formulas (5.4) And (5.5). IN result is being removed identically true formula species $a \vee \bar{a}$ And remains only expression *sportsman* (Pele). After this, having joined To expression *sportsman* (Pele) With help operations disjunctions formula (5.5), we again obtain an identically true expression

$$\overline{sportsman\ (\text{Pe le})} \vee sportsman\ (\text{Pele}),$$

which Also is removed, giving at this empty disjunct. This shows impracticability formulas (5.4) - (5.6) And, hence, hatchability method refutations formulas *sportsman* (Pele), That There is truth theorems *footballer* (X) → *sportsman* (X).

On language *Prologue* given example proof Maybe look next way :

```
sportsman (X) : - footballer (X). footballer(
Pele ).
? - sportsman (Pele).
```

This short program contains three sentences. The first one is a rule, the second one is a fact, and third - a question. In general, *Prologue* sentences are only three types: *facts* , *rules* and *questions* (or *requests*). We appeal Note that every sentence in a *Prolog program* ends with a period.

Prologue programs Can expand, adding V them new offers.

Facts contain statements that are always unconditionally true. In our example, the fact is the sentence *"footballer"* (Pele).

Rules contain statements whose truth depends on certain conditions. Therefore, rules have a conditional part (the right half of the rule after the symbol : - , which is read as "if") and the conclusion part (the left half of the rule). The conclusion is also called ***the head*** sentence, and the conditional part is its ***body*** . It follows that facts are sentences with an empty body, and rules have a head and a non-empty body. In our example, there is only one rule—the first sentence. It reads: If *X* is a soccer player, then he is an athlete. Generally, the body of a rule is a list of ***goals*** , separated by commas. *A comma* is understood as *a conjunction* . For example, a rule for the relation "to be a sister" might look like this:

Sister (X, Y): - parent (Z, X), parent (Z, Y), woman (X), different (X, Y).

During the course of calculations, another object may be substituted into the predicate instead of a variable. In this case, the variable is said to *be instantiated* . It is assumed that All variables are bound by a universal quantifier, although for variables appearing only in the body of rules, binding by an existential quantifier is also possible.

With the help of ***questions*** The user can ask the system which statements are true. The question is the target statement. Typically, the answer to the question is either affirmative or negative. "yes" (yes), or the negative "no" (no). In the *Prolog programming language* , as in the resolution method, taking the negation of a given formula as a starting point, the refutation method can be used to prove the formula itself.

One of the most important aspects of *Prolog programming is* are the concepts *of unification* (identification) and *concretization of variables* . *Prolog* attempts to identify terms when proving or matching a target sentence. For example, to match the query ?- *dog* (*X*), the target sentence *dog* (*X*) can be identified with the fact *dog* (rex), as a result of which the variable *X* has become concretized: *X* = rex. Variables included in sentences are identified in a special way - they are compared. A fact is proved for all values of the variable (variables). A rule is proved for all values of the variables in the head target sentence, provided that the target statements in the rule body are proved. Variables take on specific values for the duration of the proof of the target sentence. In the case where variables are contained only in the target sentences of the rule body, the rule is considered proven if The target proposition of the body is true for one or more values of the variables. Thus, they take on specific values at the time the target proposition in which the variables were agreed upon remains proven.

A term *X* is matched with a term *Y* according to the following rules. If *X* and *Y* are constants, they are comparable if and only if they are the same. If *X* is a constant or structure and *Y* is an uninstantiated variable, then *X* and *Y* are comparable and *Y* takes the value *X* (and vice versa). If *X* and *Y* are structures, then they are comparable if and only if they have the same principal operator and arity, and each of their corresponding components is comparable. If *X* and *Y* are uninstantiated (free) variables, then they are comparable; in this case, they are said to *be concatenated* . Note that *Prolog* finds the most general unifier of terms. Let us consider a concrete example of a *Prolog program* .

Example : The following legend exists: Ivan is interested in computers, books, and cars. Peter is interested in something that interests Ivan, but only if that something is technology and if it is produced in Russia. It is known that computers and cars are technology. Furthermore, it is known that computers Machines are produced in America, and cars are produced in the US and Russia. The question is, "What interests Peter?" For ease of analysis of the program below, all predicates and specific variable values will be denoted not by individual letters, but by words that convey their semantics. Furthermore, program objects (facts, rules, and queries) will be numbered, and symbolic expressions will be provided to the right of the program text so that we can further demonstrate in analytical form *the resolution method* , which underlies *Prolog's functioning* .

Program

1)	*interest* (Ivan , computers).	$A\,(\,i,\,a\,)$
2)	*interest* (Ivan, books).	$A\,(\,i,\,b\,)$
3)	*interest* (Ivan, cars).	$A\,(\,i,\,c\,)$
4)	*interest* (Peter, x) $\Leftarrow$	$A\,(\,p,\,x\,)$
5)	$\Leftarrow$ *interest* (Ivan, x),	$A\,(\,i,\,x\,)$
6)	*technique* (x),	$T\,(\,x\,)$
7)	*produced* (x , Russia).	$P\,(\,x,\,r\,)$
8)	*technique* (computers).	$T\,(\,a\,)$
9)	*technique* (cars).	$T\,(\,c\,)$
10)	*produced* (computers, USA).	$P\,(\,a,\,s\,)$
11)	*produced* (cars, USA).	$P\,(\,c,\,s\,)$
12)	*produced* (cars, Russia).	$P\,(\,c,\,r\,)$
Target **:**	*interest* (Peter , x).	$A\,(\,p,\,x\,)$

This program allows you to create the following contradiction, which can be resolved using the resolution method:

$A\,(\,i\,,\ a\,),\ A\,(\,i\,,\ b\,),\ A\,(\,i\,,c\,),\ T\,(\,a\,),\ T\ A(p,x) \vee \overline{A(i,x)} \vee \overline{T(x)} \vee \overline{P(x,r)},\ P(a,s),\ P(c,s),\ P(c,r),\ \overline{A\,(\,p\,,x\,)} \Rightarrow 0.$

Zero Can get only V volume case, If $x = c$. Then the disjunct

$A(p,c) \vee \overline{A(i,c)} \vee \overline{T(c)} \vee \overline{P(c,r)}$

is neutralized by predicates numbers 3, 9, 12, and 13. The search for the required value is carried out through the unification procedure, which can be tracked by program *tracing* . ***Tracing*** is the step-by-step recording of program execution.

Let us introduce the following notations: **B** - call of a new predicate; **P** - repeated call of the predicate; **Y** - successful completion of the unification procedure, i.e. the called The predicate corresponds to some fact; **H** is the unsuccessful completion of unification; **Y** is a pointer indicating that at least one more fact with a suitable unification exists. The underscore in place of x is *an anonymous variable* and completely replaces x in the program trace. The trace line ends with a number corresponding to the program listing number. Here is the trace of our program:

1. **IN** : *target* ( ) - 13,
2. **IN** : *interest* (Peter, _) - 1,
3. **P** : *interest* (Peter, _) - 2,
4. **P** : *interest* (Peter, _) - 3,
5. **U** : *interest* (Peter, _) - 4,
6. **IN** : *interest* (Ivan, _) - 5,
7. **U** : *interest* (Ivan, computers) - 1 **y** ,

8. **IN** : *equipment* (computers) - 6,
9. **U** : *equipment* (computers) - 8,
10. **IN** : *produced* (computers, Russia) - 7,
11. **P** : *produced* (computers, Russia) - 10,
12. **P** : *produced* (computers, Russia) - 11,
13. **H** : *produced* (computers, Russia) - 12,
14. **P** : *interest* (*Ivan* , _) - 5,
15. **U** : *interest* (Ivan, books) - 2 **at** ,
16. **IN** : *technique* (books) - 6,
17. **P** : *technique* (books) - 8,
18. **H** : *technique* (books) - 9,
19. **P** : *interest* (Ivan, _) - 5,
20. **U** : *interest* (Ivan, cars) - 3,
21. **IN** : *technique* (cars) - 6,
22. **P** : *technique* (cars) - 8,
23. **U** : *technique* (cars) - 9,
24. **IN** : *produced* (cars, Russia) - 7,
25. **P** : *produced* (cars, Russia) - 10,
26. **P** : *produced* (cars, Russia) - 11,
27. **U** : *produced* (cars, Russia) - 12,
28. **U** : *target* ( ) – 13.

To answer the question "What interests Peter?" or, in predicate form, *interest* (Peter, *x*), *the Prolog* system searches for facts and rule headings that match the goal. The search always begins with the first line of the program. The first three facts are not matched with the goal; then follows a rule whose heading is replaced. three subgoals. In the line 6 the first subgoal is called. Then *Prolog* - *the system* returns to the beginning programs, where is her *success* awaits . Exhibited pointer " ***U*** **"** , To to whom the system will return in case of *failure* , which may appear elsewhere in the program. In Line 8 calls the second subgoal, *equipment* (computer), for which a suitable fact is immediately found (line 9). However, for the third subgoal *(* computers, Russia), the required fact is not found (line 13). Then the variable *x* is released from its previous specification (computers) and takes on a new value, books. This specification does not satisfy the second subgoal (line 18). The variable is released again and takes on the value cars. With this specification, all three subgoals are satisfied, and therefore the search for an answer to the main question ends with success: Peter is interested in cars, that is, *interest* (Peter, cars) is the a specified predicate that ensures the truth of the entire clause.

Tasks

2.1. *A Prolog program called "Family Relationships"* is given . It is necessary to trace the program separately for two targets, having first introduced the missing predicates and rules. Since there may be multiple values for the variables *x* and *y* that satisfy the same target, it is permissible to limit the tracing to one. true meaning *x* or *y* . In addition, due to the large number of repeated calls, only the first and last lines of repeated calls are allowed, for example:

…

81. **IN** : *husband* (Felix) - 85,
82. **P** : *husband* (Felix) - 1,

…

99. **P** : *husband* (Felix) - 18,

100. **U** : *husband* (Felix) - 19.

Prologue program "Related relationship"

husband (x) - x male,
female (x) - x woman,
father (x, y) - x father y, mother (x, y) - x mother y, daughter (x, y) - x daughter y, son (x, y) – x son of y,
genus (x, y) - x parent y,
brother (x, y) – x brother y,
sister (x, y) - x sister y.

Clauses:

1) *husband* (Nikolai).
2) *husband* (Ivan).
3) *husband* (Stepan).
4) *husband* (Sergey).
5) *husband* (Pavel).
6) *husband* (Igor).
7) *husband* (Anatoly).
8) *husband* (Joseph).
9) *husband* (Roman).
10) *husband* (Kirill).
11) *husband* (Dmitry).
12) *husband* (Maxim).
13) *husband* (Eugene).
14) *husband* (Peter).
15) *husband* (Efim).
16) *husband* (Yuri).
17) *husband* (Vadim).
18) *husband* (Oleg).
19) *husband* (Felix).
20) *wives* (Maria).
21) *female* (Olga).
22) *wives* (Tatiana).
23) *wives* (Jeanne).
24) *wives* (Irina).
25) *female* (Alice).
26) *wives* (Catherine).
27) *female* (Elena).
28) *wives* (Victoria).
29) *female* (Polina).
30) *wives* (Louise).
31) *female* (Natalia).
32) *wives* (Louise).
33) *wives* (Bella).
34) *wives* (Anna).
35) *father* (Nikolai, Ivan).
36) *father* (Stepan, Maria).
37) *father* (Ivan, Olga).
38) *father* (Ivan, Sergey).
39) *father* (Ivan, Tatyana).
40) *father* (Sergey, Victoria).
41) *father* (Pavel, Roman).
42) *father* (Sergey, Dmitry).
43) *father* (Joseph, Irina).
44) *father* (Efim, Anatoly).
45) *father* (Efim, Polina).
46) *father* (Yuri, Vadim).
47) *father* (Peter, Efim).
48) *father* (Peter, Yuri).
49) *father* (Peter, Anna).
50) *father* (Maxim, Peter).
51) *father* (Dmitry, Natalia).
52) *father* (Felix, Oleg).
53) *father* (Oleg, Eugene).
54) *mother* (Alice, Peter).
55) *mother* (Catherine, Elena).
56) *mother* (Elena, Efim).
57) *mother* (Elena, Yuri).
58) *mother* (Elena, Anna).
59) *mother* (Maria, Olga).
60) *mother* (Anna, Victoria).
61) *mother* (Louise, Eugene).
62) *mother* (Tatiana, Polina).
63) *mother* (Maria, Sergey).
64) *mother* (Anna, Oleg).
65) *mother* (Tatiana, Anatoly).
66) *mother* (Anna, Dmitry).
67) *mother* (Victoria, Bella).
68) *mother* (Maria, Tatiana).
69) *mother* (Olga, Zhanna).
70) *mother* (Olga, Paul).

71) *mother* (Jeanne, Irina).

72) *mother* (Irina, Igor).

73) *mother* (Irina, Kirill). 74) *mother* (Barbara, Novel).

75) genus $(x, y) \Leftarrow$ father (x, y).

76) *genus* $(x, y) \Leftarrow$ *mother* (x, y).

77) *son* $(x, y) \Leftarrow$ *genus* (y, x), *husband* (x).

78) *daughter* $(x, y) \Leftarrow$ *genus* (y, x), *wives* (x).

79) *brother* $(x, y) \Leftarrow$ *genus* (z, x), *genus* (z, y), *husband* $(x), x \neq y$.

80) *sister* $(x, y) \Leftarrow$ *genus* (z, x), *genus* (z, y), *wives* $(x), x \neq y$.

Options goals:

1. *father-in-law* (Felix, *x*), *nephew* (Paul, *y*).
2. *aunt* (Anna, *y*), *grandson* (*x* , Alice).
3. *cousin Brother* (Vadim, *y*), *granddaughter* (*x* , Sergey).
4. *uncle* (Efim, *y*), *mother-in-law* (Maria, *y*).
5. *daughter-in-law* (*x* , Maxim), *grandson* (*x* , Anna).
6. *niece* (Jeanne, *y*), *grandson* (Novel, *y*).
7. *mother-in-law* (Elena, *y*), *granddaughter* (Bella, *y*).
8. *mother-in-law* (Maria, *y*), *father-in-law* (*x* , Elena).
9. *grandson* (Eugene, *y*), *uncle* (Yuri, *y*).
10. *aunt* (Tatiana, *y*), *cousin Brother* (Dmitry, *y*).
11. *son-in-law* (Peter, *y*), *aunt* (Olga, *y*).
12. *mother-in-law* (*x* , Louise), *uncle* (Sergey, *y*).
13. *father-in-law* (*x* , Sergey), *cousin Brother* (Vadim, *y*).
14. *mother-in-law* (*x* , Sergey), *cousin sister* (*x* , Victoria).
15. *nephew* (Vadim, *y*), *granddaughter* (Irina, *y*).
16. *father-in-law* (Peter, *y*), *granddaughter* (Pauline, *y*).

As an example, we will trace the *Prolog program* “Family Relationships” for the first goal of option 16. For We will supplement this existing program with the following clause:

81) *father-in-law* (*x* , *y*) $\Leftarrow$

82) *father* (*x* , *z*),

83) *father* (*z* , u),

84) *mother* (*y* , *u*).

***Target* :** 85) *father-in-law*

(Peter, *y*). Program tracing:

1. **IN** : *target* ( ) - 85,
2. **IN** : *father-in-law* (Peter, _) - 81,
3. **IN** : *father* (Peter, _) - 82,
4. **P** : *father* (Peter, _) - 35,

…

15. **P** : *father* (Peter, _) - 46,
16. **U** : *father* (Peter, Efim) - 47,
17. **IN** : *father* (Efim, _) - 83,
18. **P** : *father* (Efim, _) - 35,

…

26. **P** : *father* (Efim, _) - 43,
27. **U** : *father* (Efim, Anatoly) – 44 **y** ,
28. **IN** : *mother* (_ , Anatoly) - 84,
29. **P** : *mother* (_ , Anatoly) - 54,

...

39. **P** : *mother* (_ , Anatoly) - 64,
40. **U** : *mother* (Tatiana, Anatoly) - 65 **at** ,
41. **U** : *target* ( ) – 85.

From corresponding facts And rules let's make up contradiction:

$$father\text{-}in\text{-}law(\text{Peter}, \overline{y}) \vee father(\text{Peter}, \overline{z}) \vee father(\overline{z}, u) \vee mother(y, u),$$

$$father(\text{Peter}, \text{Efim}),\ father(\text{Efim}, \text{Anatoly}),\ mother(\text{Tatiana}, \text{Anatoly}) \Rightarrow 0.$$

The target clause is neutralized by predicates numbers 47, 44 and 65 when the variables take on specific values: y = Tatyana, z = Efim, u = Anatoly.

2.2. Using predicate *spouses* (*wife* , *husband*), describe relationship *mother-in-law* , *brother-in-law* And *son-in-law* .

2.3. Write the following statement as a rule in Prolog: *a*) everyone who has a child is happy; *b*) every *X* who has a child who has a sister has two children.

Hint : In case *a*) introduce the one-argument relation *happy* ; in case *b*) introduce the new relation *have_two_children* .

2.4. Suppose a rectangle is represented by the term *rectangle* (p_1 , p_2 , p_3 , p_4), where p_1 , p_2 , p_3 , p_4 are ordered vertices of the rectangle. Determine the *regular relation* (*R*), which holds if *R is* a rectangle with vertical and horizontal sides.

Indication . Apply specification.

2.5. A tower of cubes can be described by a set of facts of the form *(* cube *1* , *cube* 2), which are true if *cube* 1 is placed on *cube* 2. Determine the predicate *above* (*cube* 1, *cube* 2), which is true if *cube* 1 is located higher in the tower than *cube* 2.

2.6. Write a logic program that defines the logical relation *fib* (*n* , *F*) to determine *n* -th Fibonacci number *F* .

Indication . Use recursion.

2.7. Write definition type Boolean formula".

2.8. Write a program that recognizes logical formulas in conjunctive normal form, that is, formulas that are conjunctions of disjunctions of literals, where a literal is an atomic formula or its negation.

2.9. Write a program that defines the relation *negation* (F_1 , F_2), which is satisfied if the logical formula F_2 is obtained from the logical formula F_1 by introducing all negation operations into conjunctions and disjunctions.

2.10. Write a program to reduce a logical formula to conjunctive normal form, that is, to a conjunction of literal disjunctions.

Conclusion

The practical significance and originality of the resolution method are determined by its two following properties.

1. The control over movement through the "and-or" tree, typically required for any proof, is absent in this case. This function is performed by a standardized set of clauses, and a simple "choice + unification" processing procedure can be defined.
2. Hypotheses, assumptions, and intermediate propositions play the same role here. Proofs are obtained by moving simultaneously in two directions: from hypotheses to conclusions and from conclusions to hypotheses, and proof is achieved when these two paths of observation meet.

A significant disadvantage of the resolution method is that by its very design it is intended only For proof theorems, A Not receive new theorems. TO to that same,

If the proposition is not a theorem, the resolution can lead to the construction of an infinite decision tree. Thus, the resolution method is interesting due to its simplicity and systematicity, but is applicable only to a limited number of cases (proof Not should have big depth, A number potential resolutions should not be big).

Exercise students For independent work

3. From methodological developments decide tasks With even numbers.
1. Get ready To practical occupation By topic "Logics predicates."

6 LOGICS PREDICATES

Educational objectives **:** The main objectives of the lesson are: the transition from propositional algebra to its extension – predicate logic, and the development of primary skills in using logical and quantifier operations on predicates to solve practical problems.

6.1 Educational questions classes:

1. Finding regions truth predicates.
2. Proof equivalence formulas logic predicates.
3. Application logic predicates For records mathematical definitions And proposals.

The language of predicate logic is an extension of the language of propositional logic. Therefore, both the alphabet and the set of operations are expanded. Predicate logic formulas receive a specific interpretation if a nonempty set *M is specified* and the truth values of each of the components of the formula are given. Of particular interest are universally valid formulas (identities, logical laws), whose truth value is equal to 1 in any interpretation. Therefore, students must master a sufficiently rich set of identities in predicate logic and be able to justify them using substantive set-theoretic considerations.

6.2 Questions For checks student readiness to class

1. What distinguishes predicate logic from propositional logic?
2. What represents by yourself statement form?
3. Give a definition of a one-place predicate, its domain of definition and truth set.
4. What predicates are called identically true (identically false)?
5. How can the concept of a one-place predicate be generalized to the concept of a multi-place predicate?
6. Define logical operations on predicates. How are the results of logical operations on predicates related to the truth sets of the original predicates?
7. Give definition quantifiers operations.
8. Formulate rule O introduction sign denials under sign quantifier.
9. Give definition formulas And subformulas logic predicates.
10. Give definition equivalent formulas logic predicates.

1. Finding regions truth predicates

The conjunction of unary *predicates* $P(x)$ and $Q(x)$ is a predicate $P(x)\&Q(x)$ that turns into a true statement if and only if each of the predicates $P(x)$ and $Q(x)$ turns into a true statement.

If J_P, J_Q, $J_{P\&Q}$ - areas truth predicates $P(x)$, $Q(x)$ And $P(x)\&Q(x)$ respectively, the $J_{P\&Q} = J_P \cap J_Q$.

Disjunction predicates $P(x)$ and $Q(x)$ it's called predicate $P(x) \vee Q(x)$, which turns into a true statement for those and only those values of the variable x from the domain of definition of the given predicates for which at least one of the predicates $P(x)$ or $Q(x)$ turns into a true statement.

Region truth $J_{P\vee Q}$ disjunctions $P(x)\vee Q(x)$ is set $J_P \cup J_Q$.

Denial predicate $P(x)$, given on set M, it's called predicate $\overline{P}(x)$, which turns into a true statement for those and only those values $x \in M$ for which the predicate $P(x)$ turns into a false statement.

If J_P - region truth predicate $P(x)$, J_P- addition sets J_P to sets M, That region truth predicate $\overline{P}(x)$ is set $J_{\overline{P}} = \overline{M} \setminus J_P$.

The implication of predicates $P(x)$ and $Q(x)$ defined on a set M is called the predicate $P(x) \to Q(x)$, which appeals V false statement For those And only x values $\in M$, for which the predicate $P(x)$ turns into a true statement, and the predicate $Q(x)$ - into false.

Freeing oneself from implications, easily show, What region truth predicate $P(x) \to Q(x)$ is set $J_{P\to Q} = J_{\overline{P}} \cup J_Q$.

The definitions of conjunction, disjunction, negation and implication can be generalized to the case of multi-place predicates.

6.3 Tasks

1.1. On set $M = \{1,2,3,...,20\}$ asked predicates:

$A(x) \equiv$ "x Not is divided on 5";

$B(x) \equiv$ "x - even number";

$C(x) \equiv$ "x - number simple ";

$D(x) \equiv$ "x multiple 3".

Find areas truth following predicates:

1.1.1. $A(x)\& B(x)$;

1.1.2. $B(x)\& C(x)$;

1.1.3. $B(x)\vee \overline{D}(x)$;

1.1.4. $A(x)\vee B(x)\vee D(x)$;

1.1.5. $C(x)\to A(x)$;

1.1.6. $D(x)\to \overline{C}(x)$;

1.1.7. $(A(x)\ \&\ C(x))\to \overline{D}(x)$;

1.1.8. $(A(x)\ \&\ D(x))\to \overline{C}(x)$.

Answers :

1.1.1. $J_{A\&B} = \{2,4,6,8,12,14,16,18\}$;

1.1.2. $J_{B\&D} = \{2\}$;

1.1.3. $J_{B\vee \overline{D}} = \{1,2,4,5,7,8,10,11,13,14,16,17,19,20\}$;

1.1.4. $J_{A\vee B\vee D} = J_A \cup J_B \cup J_D = \{1,2,3,4,6,7,8,9,11,12,13,14,16,17,18,19\} \cup$
$\cup \{2,4,6,8,10,12,14,16,18,20\} \cup \{3,6,9,12,15,18\} = M \setminus \{5\}$;

1.1.5. $J_{C\to A} = \overline{J}_C \cup J_A = \{1,4,6,8,9,10,12,14,15,16,18,20\} \cup$
$\cup \{1,2,3,4,6,7,8,9,11,12,13,14,16,17,18,19\} = M \setminus \{5\}$.

1.1.6. $J_{D\to \overline{C}} = \overline{J}_D \cup J_{\overline{C}} = \{1,2,4,5,7,8,10,11,13,14,16,17,19,20\} \cup$
$\cup \{1,4,6,8,9,10,12,14,15,18,20\} = M \setminus \{3\}$;

1.1.7. $J_{(F\&C)\to \overline{D}} = \overline{J}_{A\&C} \cup J_{\overline{D}} = \overline{J_A \cap J_C} \cup J_{\overline{D}} = \overline{J}_A \cup \overline{J}_C \cup \overline{J}_D =$
$= \overline{J_A \cap J_C \cap J_D} = M \setminus \{3\}$;

1.1.8. $J_{\overline{(A\&D)\to C}} = J_{A\&D} \cup J_{\overline{C}} = J_A \cup J_C \cup J_D$
$= \overline{J_A \cap J_C \cap J_D} = M \setminus \{ 3 \}$.

1.2. To depict on diagrams Euler-Venn areas truth For following predicates:

1.2.1. $\overline{P}(x) \,\&\, \overline{Q}(x)$;
1.2.2. $(P(x) \to Q(x)) \vee \overline{(R(x) \,\&\, Q(x))}$;
1.2.3. $P(x) \to (Q(x) \vee \overline{Q}(x))$;
1.2.4. $P(x) \,\&\, Q(x) \to \overline{R}(x)$.

1.3. Show on valid coordinate planes areas truth predicates:

1.3.1. $(x \geq 3) \to (y < 5)$;
1.3.2. $\overline{(x > 2)} \,\&\, (x < y)$;
1.3.3. $(x^2 + y^2 < 1) \to (xy > 0)$;
1.3.4. $\begin{cases} y > 2x - 4; \\ y < 1/3x + 2; \\ y > -\dfrac{1}{2}x + 3. \end{cases}$

2. Proof equivalence formulas logic predicates

Quantifier universality . Let $P(x)$ - predicate, definite on set M . Then under the expression $\forall xP(x)$ understand a statement that is true when the predicate $P(x)$ appeals V true statement For each element $x \in M$, And V false - otherwise.

Existential quantifier . The expression $\exists xP(x)$ is understood as a statement that is true V volume And only V volume case, When exists Although would one element $x \in M$ for which the predicate $P(x)$ turns into a true statement.

Variable x V predicate $P(x)$ they call *free* , A V expressions $\forall xP(x)$ And $\exists xP(x)$ - *bound* variable. Quantifiers $\forall$ And $\exists$ are called *dual to each other* . The negation sign can be introduced under the quantifier sign, replacing the quantifier its dual quantifier.

Predicate symbols from subject permanent And variables, A Also variables Statements are called *elementary formulas* . From elementary formulas, using the operations of conjunction, disjunction, negation, implication, and binding by universal and existential quantifiers, one can obtain more complex formulas of predicate logic.

6.4 Tasks

2.1. Which Which of the following expressions (words) are formulas of predicate logic? In each formula, identify the free and bound variables.

2.1.1. $\overline{\exists x \forall z (P(x, y) \to P(y, z))}$;
2.1.2 $\forall x(P(x) \to Q(x)) \to (\exists xP(x) \to xR(x, y))$;

2.1.3 $(p \to q) \& (\bar{r} \vee \bar{p})$;
2.1.4. $P(x) \& \forall x Q(x)$;
2.1.5. $(P(x) \to Q(x)) \vee \exists y(\forall y R(y))$;
2.1.6. $\exists x \forall z(P(x, y) \to P(y, z))$;
2.1.7. $p \to \forall x P(x, y)$;
2.1.8. $\exists x, yP(x, y)$.

Answer : The formulas are 2.1.1, 2.1.2, 2.1.3, 2.1.6 and 2.1.7. The remaining expressions are not formulas.

2.2. Calculate values formulas:

2.2.1. $\forall x (P (x) \& Q (x) \to R (x))$;
2.2.2. $\forall x \exists yP (x , and) \to \exists x \forall yP (x , and)$,

Where $P (x) \equiv$ « x is divided on 3", $Q (x) \equiv$ « x is divided on 4", $R (x) \equiv$ « x is divided on 2» and predicates defined on the set of natural numbers; $P (x , y) \equiv$ " the number x is less than the number y " and $P (x , y)$ is defined on the set N^2.

Answer : V case 2.2.1 given formula accepts meaning "true"; formula 2.2.2 accepts meaning "lie". In second case statement $\forall x \exists yP (x , y)$- truly, A statement $\exists x \forall yP (x , y)$ - false, hence, all formula accepts meaning "lie" By definition implications statements.

2.3. Prove next equivalence:

2.3.1. $\forall xA (X) \equiv \overline{\exists \overline{xA}} (x)$;
2.3.2. $\exists xA (x) \equiv \overline{\forall \overline{xA}} (x)$;
2.3.3. $c \& \forall xA(x) \equiv x(c \& A(x))$;
2.3.4. $c \vee cA(x) \equiv \forall x(c \vee A(x))$;
2.3.5. $\exists x(A(x) \vee B(x)) \equiv \exists xA(x) \vee \exists xB(x)$;
2.3.6. $\exists x(c \vee A(x)) \equiv c \vee \exists xA(x)$.

IN quality example we will prove it equivalence predicates V 2.3.3. Let's consider two cases.

A) Let statement *With* - false. Then For any x false will be statements $c \& \forall xA (x)$ and $\forall x (c \& A (x))$. Hence, V this case both original formulas identically are false.

b) Now let the statement c be true. Then, obviously, the values of the original formulas will be entirely depend from values predicate $A (x)$. If $A (x)$ - identically true predicate, That will identically true And predicate $c \& A (x)$, And, hence, will be identically true statements $\forall xA (x)$, $c \& \forall xA (x)$, $\forall x (c \& A (x))$, That There is identically true original formulas. If same predicate $A (x)$ Not identically true, That will Not identically true And predicate $c \& A (x)$, A statements $\forall xA (x)$, $c \& \forall xA (x)$, $\forall x (c \& A (x))$ will be false, That There is false values accept both original formulas, What V in the end proves their equivalence.

2.4. Find denials following formulas:

2.4.1. $\exists x(A(x) \& B(x) \& C(x))$;
2.4.2. $\forall x(A(x) \vee \exists yB(y))$;
2.4.3. $\forall x (A (x) \to \forall yB (and))$;
2.4.4. $\forall x(A(x) \to B(x)) \& \exists x(S(x) \& R(x))$;

2.4.5. $\exists x\,(R(x) \to R(x))$.
Let's decide, For example, 2.4.2 And 2.4.3.

2.4.2. $\overline{\forall x(A(x) \vee \exists y B(y))} \equiv \overline{\forall x A(x)}\ \&\ \overline{\exists y B(y)} \equiv \exists x \overline{A(x)}\ \&\ y\overline{B(y)}.$

2.4.3. $\overline{\forall x(A(x) \to \forall y B(y))} \equiv \exists x(\overline{A(x) \to \forall y B(y))} \equiv \exists x(A(x)\ \&\ \overline{\forall y B(y))} \equiv$
$\equiv \exists x(A(x)\ \&\ \exists y\overline{B(y))} \equiv \exists x A(x)\ \&\ \exists y\overline{B(y)}.$
We took advantage of rule introduction denials under quantifier sign .

2.5. Let $A(x)$ And $B(x)$ - two single predicate, certain on set M such that the statement $\exists x\,(A(x) \to (\overline{A(x)} \vee \overline{\overline{B(x)} \to A(x)}))$ is true.
Prove that the statement $\forall x A(x)$ is false.
Proof easily is being carried out method from the contrary.

3. Application logic predicates For records mathematical definitions and propositions

3.1. Write down on language logic predicates definitions:

3.1.1. A linearly ordered set (a set is called linearly ordered if for any elements of this set x and y either $x = y$, or $x < y$, or $x > y$).

3.1.2. Limited functions (function $f(x)$ it's called limited on the set M, If exists such non-negative number l, What For everyone $x \in M$ fair inequality $f(x) \le l$).

Answer : 3.1.1. $\forall (x \in M)\ \forall (and \in M)((x = and) \vee (x > and) \vee (x < and))$.

3.1.2. $\exists (l \in R)\ \forall (x \in M)(f(x) \le l)$.

3.2. Write down on language logic predicates next statements:

3.2.1. Some valid numbers are rational.
3.2.2. No one simple number Not is accurate square.
3.2.3. Some even numbers Not are divided on 8.
3.2.4. All sorts of things number, multiple 6, is divided on 3.
3.2.5. Numbers 5 And 12 Not have common divisors, excellent from +1 And -1.
3.2.6. Natural number, divisible on 6, is divided on 2 And on 3.
3.2.7. For anyone whole numbers x exists such whole number y, what $x = 2y$ or $x = 2y + 1$.
3.2.8. For any natural number, there is another natural number that is greater than the first.
3.2.9. Exists the smallest natural number.
3.2.10. Not exists such rational numbers x, that $x^2 - 2 = 0$.
3.2.11. For all sorts whole numbers x And z exists whole number y such, what $x + y = z$.
3.2.12. For any two rational numbers x And y exists rational number z is such, that $x < z$ And $z < y$.

Let $P(x)$ means « x - simple number", $Q(x)$ – « x - even number", $R(x)$ - « x - integer", $D(x, y)$ – " x divides y ". Formulate the following statements in words, written in the language of predicate logic. Indicate which of them are true and which are false:

3.3.1. $\forall x P(x) \to \underline{Q(x)}$;

3.3.2. $\forall x(\overline{P(x)} \to \forall y\, P(y) \to \overline{D(z, y)}))$;

3.3.3. $\forall x\,(Q(x) \to \forall y\,(D(x, y) \to Q(y)))$;

3.3.4. $\forall x \exists y\,(R(x) \& R(y) \to D(x, y))$;

3.3.5. $\forall y \forall x\,(R(x) \& R(y) \to D(x, y))$;

3.3.6. $\exists x \forall y\,(R(x) \& R(y) \to D(x, y))$.

Conclusion

By studying and researching a series of problems on predicate logic, we effectively transition to a new level of abstraction, similar to that achieved in school—from the arithmetic of real numbers to the algebra of numerical functions. What's new here, compared to propositional algebra and propositional calculus, is the introduction of quantifier operations. These operations are widely used to write definitions and compactly represent the proofs of various theorems.

6.5 Exercise students For independent work

Decide tasks No. 1.1.2, 1.1.6, 1.1.8, 1.2.2, 1.2.3, 1.3.2, 1.3.4, 2.2.2, 2.3.2, 2.3.4, 2.3.6, 2.4.1, 2.4.3, 2.4.5, 2.5, 3.1.2, 3.2.2, 3.2.4, 3.2.6, 3.2.8, 3.2.10, 3.2.12, 3.3.2, 3.3.4, 3.3.6.

Prepare for a practical lesson on the topic "Constructing inferences in predicate calculus."

7 CONSTRUCTION CONCLUSIONS IN CALCULATION PREDICATES

Learning objectives **:** To consolidate the basic concepts of calculus when solving problems predicates as a theory that forms the next level of abstraction in the construction of formal systems.

7.1 Educational questions classes:

1. Feasibility And general significance formulas logic predicates.
2. Pre-prepared, Skolemovskaya And clausal forms performances formulas in predicate calculus.

7.2 Questions For checks readiness students to class

1. Give definition alphabet predicate calculus .
2. Give inductive definition formulas And subformulas predicate calculus .
3. Prove, What expressions $\exists x (F^1 (x) \rightarrow \forall y G^2 (y , z))$ And $\forall x F^1 (x) \vee \forall x \exists y G^2 (x , y)$ are formulas, but the expression $\exists x F^1 (x) \& \exists y G^2 (x , y)$ is not a formula.
4. Explain, Why V formula free And related variables must be designated by different letters.
5. Why, If any quantifier located V areas actions another quantifier, then the variables associated with these quantifiers should be denoted by different letters?
6. Which formula it's called identically true V calculus predicates?
7. Formulate axioms, specific only For calculus predicates.
8. Formulate For calculus predicates main rules conclusion.
9. Formulate theorem deductions V calculus predicates.
10. How differ concepts feasible formulas V given areas from the concept of a formula that is simply executable?
11. Give definition generally significant formulas predicate logic .
12. Define concepts conclusion And provable formulas V predicate calculus.
13. Formulate necessary And sufficient conditions general significance predicate calculus formulas.
14. Define concepts pre-predicate, Skolem and clausal forms of representation of formulas of predicate calculus.

Preliminary intelligence

Predicate calculus, like any axiomatic system, contains symbols (an alphabet) from which formulas are composed. Then, among all the formulas, formulas called derivable formulas are identified. The identification of derivable formulas in predicate calculus is also as in propositional calculus, it is carried out by specifying a certain finite set of formulas, which are called axioms, and by specifying the rules of inference that allow one to obtain new derivable formulas from the formulas being derived.

Definition formulas predicate calculus

Each formula of the predicate calculus represents a finite sequence of symbols of this calculus (alphabet). The alphabet of the predicate calculus can be divided into six groups.

1. Small Latin letters With indices or without them: $a , b , c , \ldots, x , y , z , \ldots, a_1 , a_2 , \ldots, x_1 , \ldots$ These symbols are called *object variables* (object variables).
2. Uppercase Latin letters With indices down or without them: $A , B , \ldots, A_1 , A_2 , \ldots$. These symbols are called *statement variables* .

3. Uppercase Latin letters With indices up: F^P, G^P, …, S^P, …, And these same symbols With indexes below : $F^P_1, F^P_2, ..., G^P_1, G^P_2, ..., S^P_1, S^P_2, ...$ This symbol wear names *variables predicates* from r variables ($r = 1, 2, \ldots, n$).
4. Signs operations calculus statements: $\overline{\ }$, & , $\vee$, $\rightarrow$.
5. Symbols quantifiers $\forall, \exists$.
6. Round brackets: (,).

Each formula is *a word* in the alphabet containing The symbols specified above in points 1-6. Therefore, we will determine which words are called formulas.

1. Each variable statement There is a formula.
2. If F^P is a symbol of a variable predicate, a_1 , a_2 ,…, a_p - symbols of subject variables, not necessarily different, that is, the word $F^P(a_1, a_2, \ldots, a_p)$ *is a formula.* We will retain the name variable predicate for such formulas.

Formulas, certain V p.p.1 And 2, We we will call *elementary* .

3. Let formula F contains free variable X . Then words

$$\forall xF , \exists xF \qquad (7.1)$$

Also are formulas.

Let us recall that the symbols $\forall$ and $\exists$, as in predicate logic, are called quantifiers: the first is a universal quantifier, the second an existential quantifier. The variable x in formulas (1) is a bound variable. The remaining subject variables, which are free in formula F , remain free in both formulas (7.1). Variables that are bound in formula F remain bound in formulas (7.1).

4. Let F_1 and F_2 - formulas, and they do not contain such subject variables that are bound in one formula and free in another. Then the words

$$(F_1 \& F_2), (F_1 \vee F_2), (F_1 \rightarrow F_2), (\overline{F_1}) \qquad (7.2)$$

are formulas. In this case, the free variables in the formulas F_1 and F_2 remain free in all formulas (7.2), and the bound variables in formulas F_1 And F_2 remain connected in formulas (7.2).

The definition of the formula of predicate calculus has the same inductive character, which is the definition of formulas in propositional calculus.

To prove any statements about formulas one can apply the principle of complete mathematical induction to construct formulas. Namely, such The proof is as follows. The statement is proved for elementary formulas 1 and 2. Then it is proved that from the assumption of the truth of this statement for formula F follows its truth for formulas $\forall xF$ and $\exists xF$, and from its truth for formulas F_1 and F_2 its truth for formulas (2) follows. From this, we conclude that our statement is true for any formula.

From the definition of a formula, it is clear that all formulas of the propositional calculus are also formulas of the predicate calculus. Indeed, among the formulas of the predicate calculus are propositional variables, and we can, starting from them, form formulas using the same operations as in the calculus. statements. However, the restrictions specified in paragraph 4 of the definition of a formula do not apply to formulas constructed only from statement variables (i.e., not including predicate symbols), since they apply only to formulas containing subject variables.

As in propositional calculus, we will agree to introduce some simplifications in the notation of formulas: omit external brackets and some internal ones if the sequence of operations, based on their priority, is not in doubt; sometimes round brackets are replaced by square ones; in the notation F^p omit the superscript p indicating the number of variables if these variables are specified explicitly in the formula F .

Except Togo, from conditions 4 inductive definitions formulas follows, What:

A) V formula free And related variables should be designated different letters;

b) If any quantifier located V areas actions another quantifier, then the variables associated with these quantifiers are also denoted by different letters.

Violation conditions *A*) And *b*) it's called *collision* variables. For example, word $\forall x (F (x) \to \exists x G (x , y))$ formula Not is, So How For him violated condition *b*).

Axioms calculus predicates

Axioms calculus predicates are divided on two groups:

1) Axioms calculus statements, consisting out of 11 axioms and separated on four groups I_1 , I_2 , $II_1 – II_3$, $III_1 – III_3$, $IV_1 – IV_3$.
2) Two predicate axioms: V_1 .

$\forall x F (x) \to F (y)$.

V_2 . $F (y) \to \exists x F (x)$.

In axioms V_1 and V_2 $F (x)$ is any formula containing free occurrences of the variable x , none of which is in the scope of the quantifier over y ; the formula $F (y)$ is obtained from $F (x)$ by replacing all free occurrences of the variable x with y .

To clarify the importance of the requirement for *x to appear* in the formula F , let us consider as a formula $F (x)$ the formula $\exists y P (y , x)$, where this requirement is violated: the free occurrence x is in the domain of $\exists y$. Substituting this formula into axiom V_1 yields the formula $\forall x \exists y P (y , x) \to \exists y P (y , y)$. If we interpret it on the set N of natural numbers with the predicate $P (u , v) \equiv$ " u is greater than v ", we obtain the statement: "if for every natural number x there is a natural number y greater than x , then there is a natural number y greater than itself." The premise of this implication is true on the set N , but its conclusion is false, and therefore the statement itself is false.

Main inference rules

1. **Rule conclusions** (**modus ponens**): if G And $G \to H$ - output formulas, then H is also a derivable formula.

We see that this rule is formulated in the same way as in propositional calculus, only the scope of formulas to which this rule is applied is wider here.

2. Rule introduction quantifier universality:

$$\frac{F \to G (x)}{F \to \forall x G (x)},$$

Where $G (x)$ contains free occurrences variable x , A formula F their does not contain.

3. Rule introduction quantifier existence:

$$\frac{G(x) \to F}{\exists x G(x) \to F}$$

at those same requirements To formulas F And G , What And V rule 2.

Violations rules 1-3 can bring To false conclusions from true statements.

Let's consider V confirmation this is an example.

Example 1. Let $P (x)$ be a one-place predicate “ x is divisible by 6”, $Q (x)$ be a predicate " x is divisible by 3." The statement $P (x) \to Q (x)$ is obviously true for any x , but applying the rule of introducing the universal quantifier to it yields the statement $P (x) \to \forall x Q (x)$, which is not always true. If we apply the rule of introducing the existential quantifier to the formula $P (x) \to Q (x)$, we obtain the predicate $\exists x P (x) \to Q (x)$, from which by way already correct) applications rules 3 we will receive statement

$\exists x P (x) \to \forall x Q (x)$, false on set natural numbers.

Note that the substitution rule has finally disappeared from the presentation; thus, from the two possible interpretations of the system of axioms (which was discussed in the propositional calculus), the interpretation in which the *substitution rule* is absent has been chosen, and instead axioms are being considered schemes axioms. Actually this choice happened already

then, when axioms V_1 and V_2 were accompanied by a verbal description of the restrictions on the occurrence of variables. Thus, the axioms ceased to be expressions of the calculus, and along with this verbal text, they turned into *meta-descriptions* of the set of formulas that are axioms, that is, into *axiom schemas* .

Constructing a predicate calculus with a substitution rule is significantly more cumbersome due to the need to distinguish between free and bound occurrences of subject variables. Therefore, most modern books on logic use the approach with axiom schemes.

The concepts of inference and provable formula in predicate calculus are defined analogously to these concepts in propositional calculus. The assertion that a formula F is deducible in predicate calculus and is denoted in the same way as in propositional calculus: $\vdash F$.

Since all formulas derivable in propositional calculus are also derivable in predicate calculus, then, by making substitutions into derivable formulas of propositional calculus, we will obtain derivable formulas of predicate calculus.

Example 2. Replacing V output formula calculus statements $\vdash A \vee A$ formula $\overline{A}$ formula $F(x)$, we will receive output formula calculus predicates $\vdash F(x) \vee \overline{F(x)}$.

Example 3. Replacing in the derived formula $\vdash A \rightarrow A \vee B$ formula A with formula $F(x)$, formula B with formula $\forall y G(y)$, we get $\vdash F(x) \rightarrow F(x) \vee \forall y G(y)$.

By substituting into the derivable formulas of the propositional calculus, one can easily obtain many derivable formulas of the predicate calculus; however, in this way, every derivable formula calculus predicates withdraw it is forbidden, those more What We have abandoned the substitution rule in predicate calculus, but only imply this rule in axiom schemas.

In predicate calculus, the following derivative is often used: inference rule: if a formula F(x) containing a free subject variable x is derivable, then the formula ∀ x F(x) is also derivable in the predicate calculus.

So, if it has place $\vdash F(x)$, That output will and the formula $\vdash \forall x F(x)$, That There is:

$$\frac{F(x)}{\forall x F(x)} .$$

This rule it's called *derivative rule bindings quantifier universality* .
It applicable To any subject variable.

In predicate calculus, there is a theorem analogous to the deduction theorem in propositional calculus: *if a formula G is deducible from a formula F , then the formula $F \rightarrow G$ is also deducible in predicate calculus* .

This theorem allows obtain derivable formulas of the predicate calculus without performing all the operations of formal deduction on them, which significantly shortens the direct path to deriving provable formulas. Note that the proof of this theorem (like most other theorems of predicate calculus) is quite complex and requires special study.

1. Feasibility And general significance formulas logic predicates

Definition 1. Formula And predicate logic is called satisfiable. V area M, if there are values of variables from area M included in this formula, for which formula A turns into a true statement.

Definition 2. Formula A is called satisfiable if there is a domain on which this formula is satisfiable.

From definitions 1 and 2 it follows that if formula A is satisfiable, this does not mean that it is satisfiable in any domain.

Definition 3. Formula A is called identically true on a set M if it takes a true value for all values of the variables included in this formula and assigned to the domain M.

Definition 4. A formula A is called universally valid if it is identically true in any domain.

From definitions 1- 4 follows:

1º. If the formula A is of general significance, That she And feasible on all sorts areas.

2º. If formula A is identically true in domain M, then it is also satisfiable in this domain.

3º. If the formula A Not feasible, That she identically false on all sorts areas.

There are two classes of predicate logic formulas: the class of satisfiable and the class of unsatisfiable formulas. A universally valid formula is called *a logical law* .

Example 4. The formulas $\forall x \exists y P(x, y)$ and $\exists x \exists y (P(x) \& P(y))$ are feasible: the first is identically true in the region $M = N_0 \times N_0$ (N_0 is the set of non-negative integers) , if in as $P(x, y)$ take a predicate « $x < at$ »; second - if in quality $R(x, y)$ take the predicate "The number x is even" on the same set N_0.

Example 5. Formula $\forall x (P(x) \vee \overline{R(X)})$ identically true V any areas M, therefore, it is generally valid, that is, it is a logical law (the law of the excluded middle). At the same time, the formula $\forall x (P(x) \& \overline{P(x)})$ is identically false in any region M and therefore impossible to implement.

There are some criteria, allowing define, generally significant formula or not. These criteria are based on the relationship between the general validity and satisfiability of predicate calculus formulas. This relationship can be expressed by the following two theorems.

Theorem 1. For a formula A to be generally valid, it is necessary and sufficient that its negation be unsatisfiable.

Theorem 2. For a formula A to be satisfiable, it is necessary and sufficient that its negation is not universally valid.

7.3 Tasks

1.1. Feasible whether next formulas:

1.1.1. $\exists x P(x)$;

1.1.2. $\forall x P(x)$;

1.1.3. $\exists x \forall y P(x, y)$;

1.1.4. $\exists x \forall y (Q(x, x) \& \overline{Q(x, y)})$;

1.1.5. $\exists x \exists y (P(x) \& P(y))$.

Hint : To prove the satisfiability of a given formula, it is sufficient to find the domain of definition of the corresponding predicates and their values such that in this area the formula takes on true values.

1.2. Are whether identically true next formulas:

1.2.1. $\exists x P(x) \rightarrow \forall x P(x)$;

1.2.2. $\overline{\exists x P(x) \rightarrow \forall x P(x)}$;

1.2.3. $\exists x \forall y Q(x, x) \rightarrow \forall y \exists x Q(x, y)$;

1.2.4. $\forall x \exists y Q(x, and) \rightarrow \exists and \forall x Q(x, and)$.

Answer : Formula 1.2.1 represents the law of excluded middle, therefore, this formula identically true. Obviously, What formula 1.2.2 -

is identically false. Formula 1.2.3 is identically true, for example, on the set N^{2}, if we take the relation " as $Q(x, y)\ y \cdot x$ " Formula 1.2.4 is identically true on any set.

1.3. Prove, What next formulas Not are generally significant:

1.3.1. $(\forall x \exists and\ P(x, and) \rightarrow \exists and \forall x\ P(x, and)$;

1.3.2. $\exists x(P_1(x) \rightarrow P_2(x)) \rightarrow (\exists x P_1(x) \rightarrow \exists x P_2(x))$.

1.4. Which from below formulas are generally valid:

1.4.1. $\exists x(P_1(x) \& P_2(x)) \rightarrow (\exists x P_1(x) \& \exists x P_2(x))$;

1.4.2. $\forall x(U(x) \rightarrow V(x)) \rightarrow (\forall x U(x) \rightarrow \forall x V(x))$;

1.4.3. $\exists x P(x) \rightarrow \forall x P(x)$;

1.4.4. $\exists x P(x) \& (r \rightarrow Q(x)) \rightarrow (\forall x(P(x) \rightarrow \overline{Q(x)}) \rightarrow \bar{r})$.

Answer : Using equivalent transformations Can prove, What formulas 1.4.1 And 1.4.4 are identically true on any set and, therefore, are generally valid, which cannot be said about formulas 1.4.2 and 1.4.3.

2. Prepositional, Skolemian , and clausal forms of representation of predicate calculus formulas

In propositional algebra, disjunctive and conjunctive normal forms were introduced, which allow for the simplification of many proofs in propositional algebra. The problems that arise in predicate logic when dealing with quantifiers and quantifier domains can be quite complex. In particular, the presence of the same variable both bound and free in a formula can lead to certain difficulties. Therefore, in predicate calculus, by identity transformations, this formula is also reduced to a simpler or standard form—to *normal* form. *A formula in predicate logic is said to be in normal form if it contains only conjunction, disjunction, and quantifier operations, with the negation operation relegated to elementary formulas* . normal forms of predicate logic are large meaning have *prenex normal forms* . In these forms, quantifier operations are either completely absent or used after all propositional algebra operations. Let us give a rigorous definition of a prenex normal form.

Definition 5. **Preliminary form** formulas calculus predicates it's called formula $F \equiv K_1 x_1 K_2 x_2 \ldots K_n x_n R$, Where K_i - one from two quantifiers, x_i - variables, incoming V formula F, $(i = \overline{1\ n}), R$ - formula, Not containing quantifiers. Expression $K_1 x_1 K_2 x_2 \ldots K_n x_n$ it's called formula **prefix** F, a formula R - her **matrix** .

Theorem 3. For any formulas calculus predicates exists logically equivalent to it is the prenex normal form.

We can propose the following algorithm for reducing a predicate calculus formula to a preliminary normal form, based on the use of the laws of logic.

1. Exclude from this formulas All implications (If They are present V formula) By law: $P \rightarrow Q \equiv P \vee \overline{Q}$. Those the most given formula is transformed To normal form.

2. Reduce the scope of the negation operation so that the negation symbol is applied to no more than one predicate letter. To do this, use the rules for constructing negations of formulas containing quantifiers, as well as De Morgan's laws and the law of double negation:

$$\overline{\forall\, xP\,(x)} \equiv \overline{\exists\, xP}\,(x);$$
$$\overline{\exists xP(x)} \equiv \forall x\overline{P(x)};$$
$$\overline{P \,\&\, Q} \equiv \overline{P} \vee \overline{Q};$$
$$\overline{P \vee Q} \equiv \overline{P} \,\&\, \overline{Q};$$
$$\overline{\overline{P}} \equiv P.$$

3. Rename bound variables, if necessary, so that within the scope of each quantifier, each bound variable has a unique name, and the names of free and bound variables are distinct. This can be done because the name of a variable bound by a quantifier does not affect the value of the formula, for example:

$\forall xU(x, y) \,\&\, \exists xV(x, y) \vee U(x) \equiv \forall xU(x, y) \,\&\, \exists\, zV(z, y) \vee U(m).$

4. Remove quantifiers if their scope does not contain the variable bound by the quantifier, for example: $\forall\, x \exists\, z\, P\,(x, y, c) \equiv \forall\, x\, P\,(x, y, c)$.

5. Move quantifiers V start formulas, for example:

$\forall xP(x) \,\&\, \forall xQ(x) \equiv \forall x(P(x) \,\&\, Q(x)); \quad \exists xP(x) \vee \exists xQ(x) \equiv \exists x(P(x) \vee Q(x)).$

6. If P is a formula that does not contain a free variable x, then the following will take place: the following equivalences:

$P \,\&\, \forall xQ(x) \equiv \forall x(P \,\&\, Q(x)); \;\; P \,\&\, \exists xQ(x) \equiv \exists x(P \,\&\, Q(x)).$

The required algorithm has been constructed. Of course, in the process of reducing the formula to prenormal form, some steps of the algorithm discussed above can be omitted as unnecessary.

Example 5. Bring To preliminary normal form formula

$F \equiv \forall x\,(P(x) \to (\forall y\,(P(y) \to P(g(x, y))) \,\&\, \overline{\forall\, y(Q(x, y) \to P(y))})).$

Solution. 1. Let's get rid of it V formula F from implications:

$F \equiv \forall x\,(\overline{P(x)} \vee (\forall\, y\,(\overline{P(y)} \vee P(g(x, y))) \,\&\, \overline{\forall\, y(\overline{Q(x, y)} \vee P(y))})).$

2. Let's reduce it volume actions operations denials:

$F \equiv \forall x\,(\overline{P(x)} \vee (\forall\, y\,(\overline{P(y)} \vee P(g(x, y))) \,\&\, \exists\, y\,(Q(x, y) \,\&\, \overline{P(y)}))).$

3. Let's rename it related quantifier existence variable y on z:

$F \equiv \forall x\,(\overline{P(x)} \vee (\forall\, y\,(\overline{P(y)} \vee P(g(x, y))) \,\&\, \exists z\,(Q(x, z) \,\&\, \overline{P(z)}))).$

4. By moving the quantifiers to the beginning of the formula according to point 5 of the algorithm, we obtain the desired normal preform of the formula F:

$F \equiv \forall x \forall\, y \exists\, z\,(\overline{P(x)} \vee ((\overline{P(y)} \vee P(g(x, y))) \,\&\, (Q(x, z) \,\&\, \overline{P(z)}))).$

Definition 6. A formula of predicate logic is called **closed** if it does not contain free subject variables.

The same predicate calculus formula admits many equivalent prenex forms. The resulting result depends on the order of application of the rules, as well as on the arbitrary renaming of variables (step 3 of the algorithm). Nevertheless, prenex and normal forms retain all the expressive power of predicate calculus. In particular, there is a connection between the general validity of a predicate logic formula and its prenex normal form. This connection is expressed by the following two statements, which can be proved by contradiction.

1. If a closed formula of predicate logic in prenex normal form contains only existential quantifiers and is identically true on any domain consisting of one element, then it is universally valid.

2. If a closed formula of predicate logic in prenex normal form contains only universal quantifiers and is identically true on any domain consisting of elements, the number of which does not exceed the number of variables included in a given formula, then it is generally valid.

Working with formulas containing quantifiers is much more difficult than with quantifier-free ones. In 1928, T. A. Skolem (1887–1963) proposed a method of replacing quantifiers in formulas with certain functional symbols. Without loss of generality, only prenex forms can be considered, since by Theorem 3 any formula Maybe be transformed V equivalent to her preliminary form. More Moreover, it is sufficient to operate with closed formulas. The essence of Skolem's method boils down to the following reasoning.

Let, for example, $F \equiv \exists x \forall y \exists z \forall u \forall v \exists w P(x, y, z, u, v, w)$ - closed formula

predicate calculus. Let us denote the value of the variable x that exists according to the first quantifier by an object constant, for example, the letter a, while discarding the first existential quantifier in the formula F. As a result, we obtain the formula

$$F_1 \equiv \forall y \exists z \forall u \forall v \exists w P(a, y, z, u, v, w).$$

That fact, What By to everyone meaning variable y Maybe be found meaning variable z can be expressed by some $z = f(y)$, called function

Skolemovsky. Then formula F_1 is transformed V formula

$$F_2 \equiv \forall y \forall u \forall v \exists w P(a, y, f(y), u, v, w).$$

The fact that for any values of the variables y, u and v there is a value for the variable z, can be expressed by another Skolem function $w = g(y, u, v)$, which in the previous formula is substituted for the variable w:

$$F_3 \equiv \forall y \forall u \forall v P(a, y, f(y), u, v, g(y, u, v)).$$

Since, by the condition, all variables included in the formula F must be bound by quantifiers, existential quantifiers are excluded, and the order of universal quantifiers does not play a role in calculating the value of the formula, then universal quantifiers are also discarded, *assuming* that all variables are bound by them. As a result, we obtain Skolemovskaya form performances this formulas $F : S_F \equiv P(a, y, f(y), u, v, g(y, u, v))$.

So, For formulas example 5 her Skolemov form has view

$$S_F \equiv \overline{P(x)} \vee ((\overline{P(y)} \vee P(g(x, y))) \,\&\, (Q(x, h(x)) \,\&\, \overline{P(h(x))})),$$

Where $z = h(x)$ - Skolemovskaya function.

So, We compared formula F formula S_F more simple buildings With guarantee, What formulas F And S_F or both feasible, or both are impossible to implement. This connection between formulas F and S_F weaker, how logical equivalence, But she Very useful at proof that some conclusion is a logical consequence of given hypotheses.

Definition 7. **The clausal** form of a formula of predicate calculus is called its Skolem form. form, which one matrix presented V view conjunctive normal form, that is, as a conjunction of elementary disjunctions. Any Skolem form admits an equivalent clausal form.

Using distributive law disjunctions relatively conjunctions, let's bring Skolemovskaya form S_F example 5 To conjunctive normal form. IN result we will receive clausal form S functions S_F

$$S = (\overline{P(x)} \vee \overline{P(y)} \vee P(g(x, y))) \,\&\, (\overline{P(x)} \vee Q(x, h(x))) \,\&\, (\overline{P(x)} \vee \overline{P(h(x))}),$$

which, replacing symbol conjunctions comma, are writing down V view So called set *of clauses*:

$$S = \left\{ \overline{P(x)} \vee \overline{P(y)} \vee P(g(x, y)),\ \overline{P(x)} \vee Q(x, h(x)),\ \overline{P(x)} \vee \overline{P(h(x))} \right\}.$$

Substantive meaning formulas, output in predicate calculus, shows that these formulas represent by yourself identically true statements, That There is

logical tautologies. Conversely, every identically true formula is derivable in the predicate calculus. It follows from this that no meaningful statement, in particular a mathematical one, can be deduced in the predicate calculus. However, if we apply the axioms of the predicate calculus If we add any non-derivable formulas as new axioms (while maintaining the same rules of inference), we will get another calculus in which, in addition to identically true formulas, other formulas are also derivable.

7.4 Tasks

Bring To preliminary normal form next formulas predicate logic :

2.1.1. $F \equiv \overline{\exists x \forall y \exists z \forall u\, P(x, y, z, u)}$;

2.1.2. $F \equiv \overline{\forall x\, R(x)} \vee \exists x\, Q(x, y)$;

2.1.3. $F \equiv \overline{p \to \exists x\, R(x)}$;

2.1.4. $F \equiv \forall x\,(A(x) \to \exists y\, C(y))$;

2.1.5. $F \equiv \exists x \forall y\, P(x, y) \& \exists x \forall y\, Q(x, y)$;

2.1.6. $F \equiv \exists x \forall y\, P(x, y) \vee \exists x \forall y\, Q(x, y)$.

Answer: $\forall x . y \forall z \exists u\, P(x, y, z, u)$; 2.1.2. $. x(\overline{R(x} \vee Q(x, y))$;

2.1.3. $\forall x(p \& \overline{R}(x))$; 2.1.4. $\forall x \exists y(\overline{A(x)} \vee C(y))$;

2.1.5. $\exists x \exists z \forall y(P(x, y) \& Q(z, y))$; 2.1.6. $\exists x \forall y \forall z(P(x, y) \vee Q(x, z))$.

2.2. Prove hatchability following formulas V predicate calculus

2.2.1. $F(x) \overline{\vee} F(x)$;

2.2.2. $F(x) \to F(x) \vee \forall y\, G(y)$.

Instructions. 2.2.1. Replacing V output formula calculus statements $\vdash A \vee \overline{A}$ formula A on $F(x)$, we will receive output formula calculus predicates $\vdash F(x) \vee \overline{F(x)}$.

2.2.2. Replacing in the output formula $\vdash A \to A \vee B$ formula A formula $F(x)$, formula *IN* formula $\forall y G(y)$, we will receive $\vdash F(x) \to F(x) \vee \forall y\, G(y)$.

In axioms and formulas We have nowhere discussed propositional calculus and predicate calculus used operation equivalence. However in many cases meet formulas species $(x \to y) \& (y \to x)$, which We For brevity we will designate symbol equivalences, That There is $(x \to y) \& (y \to x) \equiv x \leftrightarrow y$. Except Togo, let's introduce concept sequences:

sequents are expressions of $A_1, A_2, ..., A_n \vdash B$ (read: from the form formulas $A_1, A_2, ..., A_n$ follows formula B or formula B - provable). Here $n > 0$, A $A_1, A_2, ..., A_n, B$ - arbitrary formulas calculus predicates (or (propositional calculus). A sequent is said to be derivable in predicate calculus if there is a derivation from a finite sequence of sequents.

2.3. Let the variable x not enter freely into the formula A. Prove the derivability of the following sequents:

2.3.1. $\vdash (\forall xA \leftrightarrow A)$;

2.3.2. $\vdash (\exists A \leftrightarrow A)$;

2.3.3. $\vdash (\forall x \forall and\, B(x, and) \leftrightarrow \forall and \forall x(x, and))$;

2.3.4. $\vdash (\exists x \exists and\, B(x, and) \leftrightarrow \exists and \exists x\, B(x, and))$;

2.3.5. $\vdash (\forall x \forall y\, B(x,y) \to \forall x\, B(x,x))$;

2.3.6. $\vdash (\exists x B(x,x) \to \exists x \exists y B(x,y))$.

2.4. Prove, What output sequences:

2.4.1. $(A \leftrightarrow B) \vdash (\overline{A} \leftrightarrow \overline{B})$;

2.4.2. $(A \leftrightarrow B) \vdash ((A \& C) \leftrightarrow (B \& C))$;

2.4.3. $(A \leftrightarrow B) \vdash ((A \vee C) \leftrightarrow (B \vee C))$;

2.4.4. $(A \leftrightarrow B) \vdash ((A \to C) \leftrightarrow (B \to C))$.

2.5. Prove hatchability following formulas V predicate calculus :

2.5.1. $F(x) \vee \overline{F}(x)$;

2.5.2. $F(x) \to F(x) \vee \forall y G(y)$.

Instructions . 2.5.1. Replace V output formula calculus statements $\vdash A \vee \overline{A}$ formula A formula $F(x)$. 2.5.2. Replace V output formula $\vdash A \to A \vee B$ formula A formula $F(x)$, formula IN formula $\forall y G(y)$.

2.6. Build conclusions, corresponding next reasoning (legends):

2.6.1. No drug dealer is an addict. Some addicts have been prosecuted. Therefore, some people who have been prosecuted are not drug dealers.

2.6.2. All rational numbers are real numbers. Some rational numbers are integers. Therefore, some real numbers are integers.

Hint : These arguments should be formalized (construct *clauses*). Next, using the axioms and inference rules of predicate calculus, construct a derivation of the conclusion from these premises, writing out all the steps of the proof sequentially.

Conclusion

Substitution into the output many derivable formulas of the predicate calculus can be easily obtained from formulas of the propositional calculus; however, in this way any derivable formula calculus predicates withdraw it is forbidden, those more What We We have abandoned the substitution rule in predicate calculus, but only imply this rule in axiom schemas.

7.5 Exercise students For independent work

1. Decide from methodological developments tasks No. 1.1.2, 1.1.4, 1.2.2, 1.2.4, 1.3.2, 1.4.2, 1.4.4, 2.1.2, 2.1.4, 2.1.6, 2.2.2, 2.3.2, 2.3.4, 2.3.6, 2.4.2, 2.4.4, 2.5.2, 2.6.2.
2. Prepare for a practical lesson on the topic "Models of theories of first-order logic".

8 MODELS FIRST-ORDER LOGIC

Learning objectives **:** To expand and deepen the fundamental concepts of constructing first-order theories. To acquire basic skills in studying the properties of objects in specific mathematical theories from linear and general algebra, and the arithmetic of natural numbers using specific axioms.

8.1 Educational questions classes:

1. Theories With equality And partial ordering
2. Theory groups, rings and fields
3. Theory natural numbers
4. Theory vector spaces

8.2 Questions For checks readiness students to class

1. Define concepts alphabet language theories first order, words V in this alphabet, the equality of words, the set of expressions of the theory.
2. Give definition signatures language theories first order.
3. Formulate inductive definition terma And formulas theories first order.
4. Bring examples meaningful mathematical theories first order.
5. Formulate the rule of inference and the rule of binding by the universal quantifier of a first-order theory.
6. What it's called proof V theories first order? And the theorem?
7. What understand under interpretation language theories first order?
8. Prove that if Γ is the set of axioms of a theory **T** of signature σ, M is an algebraic signature system σ, in which all formulas from the set are true Γ, then in abundance M all theorems of the theory **T are true** .

Indication . Apply induction By length conclusion from sets Γ .

9. Prove that if all axioms of a theory **T** are true in some algebraic system $\boldsymbol{M}$, then there exists a model of the theory **T** .

Indication . Let V system $\boldsymbol{M} =< M ; \sigma >$ true All axioms theories **T** . Let's put it this way $x \sim y \Leftrightarrow \boldsymbol{M} \vdash (x = y)$, Where symbol ~ designated binary attitude. Necessary prove, What ~ is an equivalence relation in the set M. Then the quotient set $M / \sim$ and will be the required model.

Preliminary intelligence

In propositional algebra and propositional calculus, the use of truth tables provided an effective way to decide whether a given formula is a tautology (identically true) or not. In predicate logic, there is no effective way to For each formula, it is necessary to decide whether it is generally valid or not (except in special cases). Therefore, in mathematical theories that use the concept of a predicate and the related concepts of quantifier operations, the axiomatic method becomes essential. This method is based on simple relationships between symbols and expressions of precise formal languages and utilizes fairly simple arithmetic methods.

An axiomatic theory constructed on the basis of a given system of axioms is understood to be the set of all theorems proven on the basis of this system of axioms.

Axiomatic theories are divided on formal And informal theories.

Informal axiomatic theories are filled with set-theoretic content, the concept of derivability in them is rather vague and relies largely on common sense.

A formal axiomatic theory is said to be definite if the following conditions are met :

Asked language theories.

Definitely concept formulas V this theories.

Highlighted some set formulas, called axioms. The rules are defined conclusion in this theory.

Here, the language of a theory is understood as the set of all words of the theory over a given alphabet, together with this alphabet.

The substantive meaning of formulas derived in predicate calculus shows that these formulas are identically true statements, that is, logical tautologies. Conversely, every identically true formula is derivable in predicate calculus. It follows that no essentially meaningful statement, particularly a mathematical one, can be derived in predicate calculus. However, if we apply the axioms of predicate calculus add any non-derivable formulas as new axioms (keeping those (the same rules of inference), then we get another calculus in which, in addition to identically true formulas, other formulas are also derivable.

Among the numerous mathematical theories, so-called first-order theories stand out. They differ from higher-order theories in that they do not allow predicates that have other predicates as possible values of their arguments. Furthermore, they do not allow quantifier operations on predicates. Predicate calculus is an example of a first-order theory. In second-order calculi, for example, predicate quantifiers are possible, that is, expressions of the form $\forall P(P(x))$. Applications of such calculi are much less common.

First-order theories are, in principle, sufficient to describe most mathematical theories. Most higher-order theories can be reduced to first-order theories.

First-order theories are also related to applied calculus, characterized by the fact that What V them To logical axioms are added own axioms. IN proper axioms, as a rule, involve specific (individual) predicate letters (for example, $=$, $<$), functional letters ($+$, $\cdot$, $-$) and subject constants (for example, natural numbers, the empty set in set theory, etc.).

The value of any formal theory is ultimately determined by its ability to describe certain objects and the relationships between them. Therefore, one of the first questions for any theory is: what objects is this theory suitable for describing? Of course, if we pose this question as the general problem of the correspondence of scientific knowledge about the world to the world itself, then it cannot be discussed solely by means of mathematics (since mathematics, unlike the natural sciences, does not deal with directly with the world, but with a formal description of its various fragments) and becomes a fundamental problem of the philosophy and methodology of science. This problem goes far beyond the scope of our discipline. We are talking about a simpler case, when the set of objects for which a formal theory is constructed is itself a sufficiently strictly described subject of research. More precisely, the problem of the adequacy of a formal theory and the objects it describes is considered as a mathematical problem of the correspondence between a substantively constructed theory and a set of statements about this theory, constructed as the formal meaning of calculi. Therefore, the assignment of values (meaning) to mathematical expressions (symbols, formulas, etc.), that is, the interpretation of the theory.

Let us introduce the basic concepts of first-order formal axiomatic theories. These concepts include the concepts of alphabet, language theory, language signatures, terms and formulas.

Definition 1. Alphabet A language theories first order it's called a non-empty finite set of symbols called letters.

Definition 2. A word in the alphabet A is any finite sequence of letters. alphabet A. An empty sequence of letters is called an empty word.

Empty word We let's agree designate symbol Λ.

Definition 3. Two words a_1 and $_2 \dots a_n$ And $b_1 b_2 \dots b_k$ V alphabet A are called equal, If k = n And $\forall$ i ($a_i = b_i$), $i = \overline{1, n}$. IN this case they write a_1 and $_2 \dots a_n = b_1 b_2 \dots b_k$. Number n is called the length of the word.

Definition 4. Let **T** be some theory. Denote by A(**T**) the alphabet of this theory. The set A^*(**T**) of words of the alphabet A(**T**) is called the set expressions of theory **T.**

Definition 5. Language Theory **T** it's called pair <A(**T**), A^*(**T**)>.

First-order languages serve first-order theories. Into the alphabet of every Theory T The first order essentially includes the same symbols as in the construction of the predicate calculus.

Definition 6. Set predicate letters (or predicate variables) F^p, $_1$ $F_2^P \dots$, S_1^P, S_2^p, … together With variables statements and values subject variables (constant) it's called signature language this theories And is its specific part.

Different first-order theories may differ from each other in the composition of the letters in alphabet and signature.

Further description theories first order **T** requires inductive definitions of terms and formulas. Terms and formulas are two classes of words in the set A(**T**).

Definition 7 (inductive definition term).

1. Subject variables And constants are terms.
2. *If* $r_1, r_2, \dots, r_n$ *- baths And A - symbol n-local operations , That* $A_i^n(r_1, r_2, \dots, r_n)$ *- term .*
3. No other terms other than the terms listed in paragraphs 1 and 2, V There is no theory **T.**

According to the natural interpretation, a term is the name of some object. Besides variables and object constants, terms are also chains formed from variables and object constants by means of operation symbols, since in the implied interpretation of a term it is understood as the value of some function.

Definition 8 (inductive definition formulas).

1. *If A – predicate or function , a* $r_1, r2, \dots, r_n$ *- term , wedding A* (*1 , r2 , …,* r_n) *- Formula . IN in particular , If A variable predicate , then by* $A_i^n(r_1, r_2, \dots, r_n)$ *is called an elementary formula .*

2. If A And IN - formulas, then (A & B), (A $\vee$ IN), (A $\rightarrow$ B), ($\bar{A}$) - formulas.

3. If A is a formula, and y is a subject variable, free in A or not contained in formula A, then the expressions $\forall$ yA, $\exists$ yA – formulas. In this case, A is called the domain of the quantifier.

4. None others formulas, except listed V p.p. 1-3, No.

Axioms of first-order theory **T** are divided into two classes: logical axioms and special ones (special axioms are also called proper axioms of theory **T**).

Logical And special axioms

What are would no were formulas A, IN And WITH theories T, next formulas are schemes of logical axioms of theory T:

$$A \to (In \to A), \tag{8.1}$$

$$(A \to (B \to C)) \to ((A \to IN) \to (A \to C)), \tag{8.2}$$

$$(\bar{\bar{IN}} \to \bar{\bar{A}}) \to ((\bar{\bar{IN}} \to A) \to B), \tag{8.3}$$

$$\forall \text{ xi } A(xi) \to A(t), \tag{8.4}$$

Where A(xi) There is formula theories T, A t - term this theories, free V formula A(xi). Let us note, what t Maybe and coincide with hi, AND Then We we are coming To axiom

$$\forall \text{ xi } A(xi) \to A(xi).$$

$$\forall x_i (A \to B) \to (A \to \forall x_i B), \tag{8.5}$$

if x_i does not enter freely into formula A.

Special axioms cannot be formulated in the general case, since they vary from theory to theory.

A first-order theory that does not contain its own axioms is obviously a purely logical theory. It is called first-order predicate calculus. Note that predicate calculus, like propositional calculus, may not necessarily be constructed on the axiom systems we formulated in studying these issues. These calculi may be constructed on other axiom systems with a significantly smaller number of them. A first-order theory is a restriction of predicate calculus to a given domain. The axiomatic system of any particular theory is obtained by adding axioms specific to this theory to an adequate complete axiomatic system of predicate calculus. This does not mean that a theorem of predicate calculus remains a theorem in the theory under consideration, since the language of this theory is, generally speaking, more limited than the language of predicate calculus. Very often, it has only a finite number of different predicate and functional constants.

There may be various reasons for the creation and development of a formal or informal theory. In most cases, a theory is developed to reflect some real objects and their properties. The goal of the theory then is to derive other true facts (formulas, expressions) of this theory from a set of true expressions (formulas) accepted as axioms of the given theory.

Rules of inference and the concept of proof in first-order theory, deduction theorem

IN number rules conclusion theories first order enters two rules:

1. **Rule conclusions** (modus ponens):

$$\frac{\vdash A, \vdash A \to B}{\vdash B}. \tag{8.6}$$

2. Rule bindings quantifier universality (or generalization rule):

$$\frac{\vdash A}{\vdash \forall x A}. \tag{8.7}$$

In axiomatic theory concept true the statement makes sense only in connection with the possible applications of this theory.

Definition 9. Proof in theory **T** is called a finite sequence $S_1, S_2, \ldots, S_k$ statements of theory **T** , each of which is either an axiom, or is derived from the previous statements of this sequence according to logical rules of inference.

Definition 10. A theorem or provable statement is a statement that is the last statement of some proof.

It is clear that any axiom is a theorem, and its proof consists of one step.

Many proofs are based on the deduction theorem, which we are already familiar with, we looked at it in propositional calculus and in predicate calculus, but In first-order theories, the deduction theorem takes place in a somewhat weakened form. Before formulating the deduction theorem for first-order theories, let us clarify The concept of inference from a set of formulas in propositional calculus can be generalized to the case of first-order theories with two caveats.

Let us consider the set of formulas $\{ \Gamma , A \}$ and the derivation of $F_1 , F_2 , \ldots, F_m$ from this set. We will assume that the formula F_k, where $1 \le k \le m$, depends on formula A in this derivation in two cases:

1) the formula F_k coincides with the formula A and it is included in the derivation as a formula contained in the set $\{ \Gamma , A \}$;
2) formula F_k obtained by the rules of inference and binding by the universal quantifier formulated at the beginning of this section, from the formulas preceding it in the derivation, of which at least one depends on the formula A.

Theorem deductions. Let Γ, $A \vdash F$ and let there be such a conclusion F from the set of formulas $\{ \Gamma , A\}$, in which, under any application of the rule of binding by a universal quantifier to formulas that depend in this derivation on A, no free variable included in A is bound by a quantifier. Then $\Gamma \vdash A \to F$.

The conditions of the deduction theorem are too cumbersome, so in practice weaker consequences are often useful.

Investigation 1. If $\Gamma , A \vdash F$ And exists conclusion, built without applications rules (5) To free variable formulas A, That $\Gamma \vdash A \to F$.

Investigation 2. If formula A closed And $\Gamma , A \vdash F$, then $\Gamma \vdash A \to F$.

Let's consider example. Make sure, What $\vdash \forall x \forall y A \to \forall y \forall x A$. Really, formula $\forall x \forall y A$, How hypothesis, provable. By scheme axioms (4) provable is formula $\forall x \forall y A \to \forall y A$. Then By rule *modus ponens* we will receive $\vdash \forall y A$ And, again applying diagram axioms (4), we will receive provability formulas $\forall y A \to A$. Again applying rule *modus ponens*, we get $\vdash A$, A By rule (5.7) at first we get $\vdash \forall x A$, then $\vdash \forall x \forall y A$. IN result everyone these reasoning We we have $\forall x \forall y A \vdash \forall y \forall x A$, at this V built conclusion We no at one application rules (7) Not connects free variable of the formula $\forall x \forall y A$, therefore, based on Corollary 1, $\vdash \forall x \forall y A \to \forall y \forall x A$, What And was required.

Concepts interpretations And models theories. Isomorphism interpretations

Under ***interpretation*** *theories* **T** understand all sorts system, including V myself:

1) Not empty set M, called *region interpretations*.
2) Any correspondence f that to each element of the language of the theory **T**

compares the only one element sets M: $< A(\mathbf{T}), A^*(\mathbf{T})> \rightarrowtail M$.

IN in particular, each predicate letter $A^n \in <A(\mathbf{T}), A^*(\mathbf{T})>$ f puts V correspondence some n-local attitude in set M; each functional letter $f^n \in <A(T), A^*(T)>$ is put V correspondence some n-local operation V M, That There is function $M^n \to M$; each subject variable X - some element from sets *M.* Any formula without free variables represents by yourself a statement that true or false, A all sorts formula with free variables expresses some attitude on areas interpretations. This attitude Maybe be true For some values variables from areas interpretations And false For others.

Let some interpretation of a theory **T** with domain *M be given* , and *G* be the set of all countable sequences of elements from *M.*

WITH help sequences $S = (b_1, b_2, \ldots, b_n, \ldots) \in G$ let's define inductively function S^*, displaying all terms in the set M, as follows:

if term t – subject constant, the S* (t) = purpose;

If term t - subject constant, That S*(t) matches With interpretation this constant in the set M;

if f_j^n - functional letter, interpreted as operation g in M, And t1, t2, …, tn - thermal baths , that $With_j^*(f_n^n(t_1, t_2, \ldots t)) =$ g(S*(t1), S*(t2),…, S*(tn)).

In simple terms, for any sequence S = (b1, b2,…,bn, …) and for anyone term t S*(t) is an element of the set M that is obtained by substituting, for each i, the element bi into the places of all occurrences of the variable xi in the term t and then performing all interpretation operations corresponding to the functional letters of the term t.

For example, let the term t = f2(x3, f1(x1, a1)); M = Z - interpretation area; f1 will be interpreted as an operation of addition of integers, f2 - as a multiplication operation, a1 - How whole number 15. Then for everyone sequences S = (b1, b2,…,bn, …) integers S*(t) is the integer b3·(b1+15).

Definition 11 (inductive definition completed formulas).

1. If A is an elementary formula Aj (t1, t2, …, tn) (here the superscript omitted) and Bj is the relation corresponding to formula A in the interpretation of the theory, then formula A is considered to be satisfied on the sequence S if and only if Bj(S*(t1),…, S*(tn)) belongs to relation Bj.

Formula $\overline{A}$ is satisfied on a sequence S if and only if the formula And not performed on S.

3. The formula A→B is satisfied on S if and only if the formula A is not satisfied on S or if the formula B is satisfied on S (cf. the definition of the operation of predicate implication).

4. Formula $\forall x_i A$ completed on S Then And only Then, When formula A completed on any sequence from G that differs from S by no more than its i-th component.

Not follows confuse concepts feasible And completed formulas.

Definition 12. A formula A is called true in a given interpretation if and only if it holds for every sequence in G. This formula is called false in the same interpretation if it does not hold for any sequence in G.

Definition 13. Interpretation I is called a model for a set of formulas Γ if any formula in the set Γ is true in a given interpretation I. An interpretation I is called a model of a theory T if it is a model of the set of all theorems of the theory T, that is, if every formula in T that is provable in T is true in interpretation I.

The concept of a model of a theory T can be viewed as an interpretation of the language of that theory. Specifically, given a theory T, we ascribe a new meaning to the initial concepts of that theory. If a certain set of objects and relations between them, chosen as the initial concepts of an axiomatic theory, is V quality her interpretations, satisfies to everyone axioms theories, That she is called a model of a given axiomatic theory.

Definition 14. An interpretation I1 of a given first-order theory T is isomorphic to an interpretation I2 of the same theory if there exists a bijective mapping g (called an isomorphism) of the domain M1 of the interpretation I1 onto the domain M2 of the interpretation I2 such that:

1. If A1j And A2j – interpretations of the predicate letter Aj respectively in the interpretations I1 and I2, then whatever b1, b2,…,bn ∈ M1 A1j (b1, b2,…,bn) is satisfied if and only if A2j (g(b1), g(b2),…, g(bn)) is satisfied.

2. If f1j and f2j are the interpretations of the functional letter fj in I1 and I2, respectively, then ∀ b1 ∀ b2… ∀ bn ∈ M1 f1j(b1, b2,…,bn) = f2j (g(b1), g(b2),…, g(bn)).

If a1j And a2j - interpretations subject constant respectively V I1 And I2, That $\alpha_{2j} = g(\alpha_{1j})$.

It is clear that if the interpretations I1 and I2 are isomorphic (they write I2 ≅ I1), then the areas of these interpretations have the same powers.

Theories With equality And partial ordering

1. ***Theories with equality*** . Many theories that can be axiomatized as first-order theories use the concept of equality and the special axioms that define it:

1. $\forall x\,(x = X)$.

1. If X, y, z - various subject variables And $F(z)$ - formula, That

$\forall x \forall y\,(x = y \rightarrow F(x) = F(y))$.

Any theory that contains the special axioms 1 and 2 is called ***a theory*** **with** ***equality*** . From axioms 1 and 2, the basic properties of equality are derived: *reflexivity* , *symmetry* and *transitivity* :

1) $\vdash t = t$ for any term t;

2) $\vdash (x = and) \rightarrow (and = x)$;

3) $\vdash (x = and) \rightarrow ((and = z) \rightarrow (x = z))$.

These properties of equality are of great importance for applied calculus. From the properties of equality it follows that the relation of equality of objects in any theory with equality is an equivalence relation, which allows us to partition the set of all objects on which this relation is defined into pairwise disjoint classes. Axiom 2 of a theory with equality expresses the identity (indistinguishability) of equal elements. The meaning of this concept is that if x and y are identical, then they denote the same object. Moreover, each of the classes into which the set of all objects of a theory with equality under the relation "=" is partitioned consists of only one object. If we consider first-order predicate calculus from the point of view of naive set theory, then equality, which is defined by axioms 1 and 2, represents precisely a set-theoretical identity.

2. ***Theory partial ordering*** . Let theory **T** contains one predicate letter A_1^2 And Not contains functional letters And subject constants. Instead of formulas $A_1^2(x_1, x_2)$ And $A_1^2(X, X)$ usually they write $x_1 < x_2$ And $x_1 \not< x_2$. Except Togo, let theory $\mathbf{T}_1$ contains two special axioms:

1. $\forall x_1\,(x_1 \not< x_1)$ - irreflexivity;

2. $\forall x_1 \forall x_2 \forall x_3\,((x_1 < x_2) \,\&\, (x_2 < x_3) \rightarrow (x_1 < x_3))$ - transitivity.

Any model this theories it's called *partially ordered structure* .

Tasks

1.1. To prove the basic properties of equality - reflexivity, symmetry and transitivity.

1.2. Show, What $\forall x \forall y\,(x < y)\,\&\,(y < x)$.

This property of partial order shows that in any interpretation of theory **T** no element can simultaneously precede and follow any other element.

1.3. If To axioms 1 And 2 theories partial ordering add more one axiom 3: $\forall x \forall y (x < y \vee y < x)$, That order With signature « < » it's called *full*, A The set on which it is defined is called a well-ordered set or *a chain* . In any chain of any two distinct elements, one necessarily precedes the other. An example is the standard relation " < " on the sets of natural, integer, rational, and real numbers. On the set **R** of real numbers, one can define And other relationship strict linear order. For example, $\forall x \forall y (x < y)$, if $(xy \neq 0 \& x < y) \vee (x \neq 0 \& y = 0)$. Prove for a given such way relationship order execution axioms 1 - 3.

1.4. Let us define on the set $\mathbf{N}^2 = \mathbf{N} \times \mathbf{N}$ is a set of ordered pairs of natural numbers (x, y), for which the difference $x - y$ is a positive even number. Prove that the corresponding binary relation is a partial order relation.

1.5. Define a complete linear ordering relation on the set $\{a, b, c\}$. In how many ways can this be done? Give an example of a linear ordering relation on a given set such that no element in the set precedes the other elements.

1.6. Let us define a binary relation " ≤ » in some interpretation of **I** theory **P** as a relation with the following three properties:

1) $\forall x (x \leq x)$ - reflexivity;

2) $\forall x \forall y (x \leq y \& y \leq x \rightarrow x = y)$ - antisymmetry;

3) $\forall x \forall y \forall z (x \leq y \& y \leq z \rightarrow x \leq z)$ - transitivity.

Such attitude it's called attitude non-strict order. Let's take V quality sets **I** set Z everyone whole numbers, And let $x \leq y$ means, What number x is divided an integer $y \neq 0$ Prove that the divisibility relation is a partial order relation.

1.7. Let us denote through **E** theory With equality. Are whether next offers theorems theories **E** :

1.7.1. $\forall x \exists y \forall z (\bar{z} = x \vee \bar{y} = z)$;

1.7.2. $\forall x \exists y \forall z \exists v (v = z \& z = x \& \bar{x} = y \& v = y))$.

2. Theory groups, rings and fields

Let us remind you definitions groups, rings And fields.

Monoid $G = (G, *, e)$ it's called **group** relatively binary operations $*$, If For each element $g \in G$ exists symmetric to him element. Group it's called **commutative** (or Abelian), If group operation, except Togo, commutative.

Here through e designated neutral element relatively operations $*$.

More in detail, With points vision theories first order, definition groups Can formulate So. Let theory **T** contains one predicate letter A_1^2, one functional letter f_1^2 And one subject constant a_1. Taking advantage of accepted V algebra designations, we will write:

- $t = s$ instead of $A_1^2(t, s)$,
- $t * s$ instead of $f_1^2(t, s)$,
- *yes* instead of a_1.

Special axioms Theory **T** Here are next formulas:

A) $\forall x_1 \forall x_2 \forall x_3 (x_1 * (x_2 * x_3)) = ((x_1 * x_2) * x_3)$ - associativity;

b) $\forall x_1 (e * x_1 = x_1)$ - property neutral element;

V) $\forall\, x_1 \exists\, x_2 (x_1 * x_2 = e)$ - existence opposite element;

G) $\forall x_1 (x_1 = x_1)$ - reflexivity of equality;

d) $\forall x_1 \forall x_2 ((x_1 = x_2) \rightarrow (x_2 = x_1))$ - symmetry of equality;

e) $\forall x_1 \forall x_2 \forall x_3 (((x_1 = x_2) \& (x_2 = x_3)) \rightarrow (x_1 = x_3))$ - transitivity equality;

and) $\forall x_1 \forall x_2 \forall x_3 ((x_2 = x_3) \rightarrow (x_1 * x_2 = x_1 * x_3) \& (x_2 * x_1 = x_3 * x_1))$ - substitutivity of equality.

Any model this theories it's called *group* . If V group true formula

$\forall x_1 \forall x_2 (x_1 * x_2 = x_2 * x_1)$,

That group it's called *Abelian* or *commutative* .

A ring is a non-empty set *K* in which two binary algebraic operations are defined: addition and multiplication, and by addition This must be Abelian group - additive group rings K, A multiplication should be connected with addition left And right laws distributivity: For any elements $a, b, c \in K$ $a(b+c) = ab + ac, (b+c)a = ba + ca$.

There are no restrictions on multiplication in the general case, that is, the ring *K* by multiplication is only a groupoid (a set with a multiplicative binary algebraic operation). If multiplication in a ring is associative, then the ring is called associative. In this case, we can talk about the multiplicative semigroup of the ring K. If multiplication in a ring K is associative and commutative, then this ring is called associative-commutative. If (K, ·, e) is a monoid, then the ring (K, ·, 0, e) is called a ring with unity or a unitary ring. A commutative ring K is called **an integral domain** if it has $0_k \neq e_k$ and $\forall a \in K \; \forall b \in K \; (a \cdot b = 0 \Rightarrow a = 0 \vee b = 0)$. Elements a and b rings K are called zero divisors if $(a \neq 0 \; \& \; b \neq 0) \Rightarrow (a \cdot b = 0 \vee b \cdot a = 0)$.

A field P is an associative-commutative ring with unity. (Not coinciding with the additive neutral element), in which each element $a \neq 0$ is invertible (that is, for each element $a \neq 0$ there is an inverse a^{-1}).

It follows from the definition of a field that, with respect to multiplication, nonzero elements of the field form a commutative group, which is called the multiplicative group of the field P. Thus, a field is a hybrid of two Abelian groups - additive and multiplicative, related by the law of distributivity (due to the commutativity of multiplication, the left and right distributivity laws coincide). The product ab^{-1} is usually written as a fraction (or ratio, quotient) a/b. Therefore, the fraction a/b, which has meaning only when $b \neq 0$ is the unique solution to the equation bx = a, and operations on fractions obey the usual rules familiar from high school mathematics and following from the definition of a field. Thus, fields are a class of division rings.

Tasks

2.1. Does this form set relatively the specified operation group. If so, will it be Abelian? If not, indicate which group axioms are not satisfied.

2.1.1. Set all mutually unambiguous mappings sets *M* on itself, considered together with the operation superposition of mappings.

2.1.2. Set non-negative whole numbers relatively addition.

2.1.3. Set *Z* everyone whole numbers relatively addition operations .

2.1.4. Set In_2 everyone vectors planes relatively operations addition.

2.1.5. Set everyone is safe even numbers regarding subtraction.

2.1.6. Set everyone positive valid numbers, If operation is defined as follows: $a * b = a^b$.

2.1.7. The set of all vectors of *an n*- dimensional arithmetic space with respect to the operation of coordinate-wise addition of vectors (the coordinates of the vectors are real numbers).

2.1.8. The set of all real matrices of the same size with respect to the operation addition.

2.1.9. *n -th* roots of unity (real and imaginary) with respect to multiplication.

2.1.10. Set everyone positive whole numbers relatively multiplication.

2.1.11. Set everyone odd numbers relatively addition.

2.1.12. Set everyone rational numbers relatively multiplication.

2.1.13. The set of all square matrices of order *n* with real entries under multiplication.

2.1.14. The set of real polynomials of degree *n* with respect to addition of polynomials.

2.1.15. The set of all rotations with a common center on a plane relative to their sequential execution (composition).

2.1.16. Set all square matrices of order *n* with integer elements and determinant equal to one, with respect to matrix multiplication.

2.2. Determine which of the following sets are rings (but not fields) and which are fields with respect to the specified operations. If the operations are not specified, addition and multiplication are assumed.

2.2.1. Set whole numbers.

2.2.2. Set even numbers.

2.2.3. The set of integers that are multiples of a given number *m* (consider, in particular, the case m =0).

2.2.4. Set rational numbers.

2.2.5. Set valid numbers.

2.2.6. Set complex numbers.

2.2.7. Set numbers species $a+b\sqrt{2}$ With whole a and b.

2.2.8. Set numbers species $a+b\sqrt{3}$ With rational a And b.

2.2.9. The set of complex numbers of the form $a+bi$ with rational a and b. 2.2.10.Set square matrices order n With whole elements relatively additions And multiplication matrices.

2.2.11. Set square matrices order n With valid elements with respect to addition and multiplication of matrices.

2.2.12. Set polynomials from one variable With whole coefficients relative to the usual operations of addition and multiplication of polynomials.

2.2.13. The set of all matrices of the form $\begin{pmatrix} a & b \\ 2b & a \end{pmatrix}$ with rational or valid elements a And b relatively additions And matrix multiplication .

Answer : 2.2.1 – 2.2.3. Rings. In 2.2.3, for m = 0, we obtain the zero ring, consisting of a single number 0, which is the identity of the ring and its own inverse. The zero ring is not a field, since a field must contain more than one element. 2.2.4 – 2.2.6. Fields. 2.2.7. Ring. 2.2.8. Field. 2.2.9. Field. 2.2.10. Ring. 2.2.11. Ring. 2.2.12. Ring.
2.2.13. Matrices With rational elements a And b form field , A With valid a and b – ring, but Not field.

2.3. Prove that the set of numbers of the form $a+b\sqrt[3]{5}+c\sqrt[3]{25}$ with rational coefficients a, b And c form field; find V this field number, the opposite number $x=2+3\sqrt[3]{5}-\sqrt[3]{25}$.

3. Theory natural numbers

An axiomatic construction of the natural series of numbers, proposed by the Italian mathematician D. Peano.

Initial concepts: a non-empty set N, called a set natural numbers; the succession relation " $'$ " and the highlighted element 1. Special axioms:

1. $\forall x \in N\,(x' \neq 1)$;
2. $\forall x \forall u\,(X = at \rightarrow x' = at')$;
3. $\forall x \forall u\,(x' = at') \rightarrow x = at)$;
4. Let $M \subset N$. Then $((1 \in M)\ \&\ \forall x\,(x \in M \rightarrow x' \in M)) \rightarrow (M = N)$.

Axiom 4 is called the axiom of mathematical induction. It is based on the principle of mathematical induction, which underlies the method of proving theorems called the method of complete mathematical induction. In order to To prove that all natural numbers have some property $P\,(x)$, *first prove,* What them possesses number 1, A then prove his For arbitrary numbers x' at
"inductive assumption" that the number x *already has* the property $P\,(x)$. Due to Axiom 4: The set of numbers with property $P\,(x)$ must contain the set of all natural numbers.

Axiomatic construction systems natural numbers

Let us assume that the natural numbers are already constructed in such a way that Peano axioms 1-4 are satisfied. Each pair of natural numbers x, y can be uniquely assigned a natural number, denoted by $x + y$, such that the following axioms are satisfied:

5. $\forall x \in N\,(x' = x + 1)$;
6. $\forall x \in N \forall\, and \in N\,((x + and)' = x + and')$.

Number $x + y$ it's called *sum* natural numbers x And y, A corresponding display $N^2 \rightarrow N$ - *operation additions* .

Each pair of natural numbers *x, y can be uniquely assigned a natural number, denoted by* $x \square y$ *or by* xy , so *that* the φολλοωινγ *two* axioms *are* satisfied:

7. $\forall x \in N\,(x \cdot 1 = x)$;
8. $\forall x \in N \forall y \in N\,(x \cdot y' = x \cdot y + x)$.

Number $x \cdot y$ it's called *work* numbers x And y, A corresponding display $N^2 \rightarrow N$ - *operation multiplications* .

Numerical system $\boldsymbol{N} = <N, 1, +, \cdot>$, operations additions And multiplication V which satisfy axioms 1-8, it's called *system natural numbers* .

Based on special axioms of the system of natural numbers, logical axioms of first-order theories, using the method of complete mathematical induction and the rules of inference, one can prove most of the properties of the operations of addition and multiplication of natural numbers (commutativity and associativity of these operations, the distributive law of multiplication with respect to addition, etc.), introduce a linear order relation on the set N, etc. One can also prove the following important theorem: *every non-empty set of natural numbers contains the smallest number, that is, such a number, which less everyone the rest numbers this sets* . If To axioms 1- 8
add more two axioms zero 9. $x + 0 = x$, 10. $x \cdot 0 = 0$, We we will receive system ***non-negative integers*** .

Tasks

3.1. Prove method complete mathematical induction, What For any the following equalities hold for a natural number:

3.1.1. $\frac{1}{1\cdot 8}+\frac{1}{8\cdot 15}+\cdots+\frac{1}{(7n+6)(7n+1)}\delta\frac{n}{7n+1}$;

3.1.2. $\frac{1}{1\cdot 3}+\frac{2^2}{3\cdot 5}+\cdots+\frac{n^2}{(2n-1)(2n+1)}=\frac{n(n+1)}{2(2n+1)}$;

3.1.3. $(7^n+3n-1)\vdots 9$;

3.1.4. $1\cdot 2+2\cdot 3+\cdots+n(n+1)=\frac{1}{3}n(n+1)(n+2)$;

3.1.5. $(6^{2n}+3^{2n+2}+3^n)\vdots 11$;

3.1.6. $(3^{3n+2}+5\cdot 2^{3n+1})\vdots 19$.

IN as an example we will prove it equality 3.1.1. Let us denote amount, standing V left parts provable equality through S_n, That There is $S_n=\frac{1}{1\cdot 8}+\frac{1}{8\cdot 15}+\cdots+\frac{1}{(7n-6)(7n+1)}$.

Obviously, What $S_{n+1}=S_n+\frac{1}{(7n+1)(7n+8)}$. It is necessary to prove, What $S_n=\frac{n}{7n+1}$.

1. We're checking original equality For $n=1$. From last equality at $n=1$ we get $S_1=\frac{1}{8}$, hence, For $n=1$ original the equality is true.
2. Let's assume, What given equality true For some natural numbers $k=n$, That There is $S_k=\frac{k}{7k+1}$, And we will prove it at this assumption truth this equality at $k=n+1$. For this let's consider the sum S_{k+1}:

$$S_{k+1}=S_k+\frac{1}{(7k+1)(7k+8)}=\frac{k}{7k+1}+\frac{1}{(7k+1)(7k+8)}=\frac{7k^2+8k+1}{(7k+1)(7k+8)}=\frac{k+1}{7k+8}.$$

Expression S_{k+1} matches With those, which it turns out from S_n at $n=k+1$. Replacing S_k through $\frac{k}{7k+1}$, We those the most, used assumption, What original equality true at $n=k$.

Conclusion: on basis principle complete mathematical induction This equality is true for any natural number.

3.2. Let *s denote* the symbol of a one-place operation; through "+" and " · " - symbols of two-place additive and multiplicative operations, respectively; through 1 - the neutral element with respect to the operation " · ". Then $\sigma_N=\langle s, +, \cdot, 0\rangle$ is the signature of the system of natural numbers. Define these symbols so that The models of the theory of natural numbers became sets:

3.2.1. $N\cup\{a\}=\{1, 2, 3, \ldots; a\}$ $(a\notin N)$;

3.2.2. $N \cup \{a, b\} = \{1, 2, 3, \ldots; a, b\}$ $(a, b \notin N)$;

3.2.3. $N \cup \{ a_1, a_2, \ldots\} = \{1, 2, 3, \ldots; a_1, a_2, \ldots\}$ ($a_i \notin N$ for everyone i and at $i \neq j a_i \neq a_j$).

4. Theory vector spaces

For every natural number $n > 0$ let's define the theory E_n - *elementary theory of vector space of dimension* n. This theory is built on the language of vectors. Special axioms E_n are divided into two groups. The first group is simply the axioms of the field of real numbers with the usual operations of addition and multiplication, relating to the variables x, y, z, ..., formulas and terms of real number theory. The second group of special axioms are the well-known axioms of linear space:

In the axioms below, the initial letters of the Latin alphabet denote vectors, and the symbol Θ denotes the zero vector. The last two axioms express the fact that the dimension of the implied space is equal to n.

Natural model of the theory E_n is n-dimensional linear vector space over the field of real numbers.

1) $a + \Theta = a;$

2) $a + b = b + a;$

3) $(a + b) + c = a(b + c);$

4) $\exists b(a + b = 0);$

5) $x \cdot (a + b) = x \cdot a + x \cdot b;$

$(x + y) \cdot a = x \cdot a + y \cdot a;$

6) $x \cdot (and \cdot to) = (x \cdot and) \cdot to;$

7) $a \cdot 1 = a;$

8) $\exists a_1 \exists a_2 ... \exists a_n \ \forall x_1 \forall x_2 ... \forall x_n \ (x_1 \cdot to_1 + \cdots + x_n \cdot a = \Theta \rightarrow x_1 = \Theta \& x_2 = \Theta \& \cdots \& x_n = \Theta);$

9) $\forall a_1 ... \forall a_{n+1} \exists x_1 ... \exists x_{n+1} ((x_1 \neq 0 \vee ... \vee x_{n+1} \neq 0) \& x_1 \cdot from_1 + ... + x_{n+1} \cdot a_{n+1} = \Theta).$

Tasks

4.1. Is each of the following sets of vectors a linear subspace of the corresponding vector space:

4.1.1. All vectors n-dimensional vector space whose coordinates are integers?

4.1.2. All vectors planes, every from which lies on one from axles coordinates?

4.1.3. All vectors planes, beginning And ends which are lying down on this direct?

4.1.4. All vectors of a plane whose ends lie in the first quadrant of the coordinate system?

4.1.5. All vectors of an arithmetic vector space over the field of real numbers, the sum of all coordinates of which is equal to 0?

4.1.6. Prove that everything square matrices of order n with real elements form a vector space over the field of real numbers if we take the addition of matrices and multiplication of a matrix by a number as operations.

4.1.7. Prove that all symmetric matrices form a linear subspace of the space of all square matrices of order n.

Conclusion

Methods of mathematical logic for describing models find the most wide application in the formalization of various subject areas. Roughly speaking, when constructing a subject area model, everything begins with the introduction of suitable notations and relations, followed by an investigation of their properties. Introduction of mathematical terminology logic, like this in this way, enters V arsenal funds, necessary For

abstract modeling preceding practical programming of tasks in a specific subject area.

Any specific mathematical theory **T** can be translated into the language of a suitable formal system *S* in such a way that every meaningful (false or true) sentence of the theory **T** expressed by some formula of the system *S.* However, K. Gödel showed in the 1930s that such a method of formalization does not allow one to construct all the positive content of mathematical theories. Therefore, there is certain boundaries possibilities of the axiomatic method in the form it acquired within the framework of Hilbert's formalism (it was D. Hilbert who first proposed such a formalism at the end of the 19th century).

8.3 Exercise students For independent work

1. Decide from methodological developments tasks With even numbers.
2. Get ready To control work By topics 1- 8.

1. Main concepts algorithmization And programming

1.1. Stages solutions tasks on computer

Problems solved with the help of computers can be classified according to various criteria: by the type of information and information technologies, by the method of finding a solution (simple and enumeration), by the nature of the goals (problems of optimization, control, training, information retrieval), by functional purpose, as well as by the level of achievement of the goal and the level of their solution, including the level of automation of the stages of their solution.

A problem becomes solvable if a rule, a method for obtaining a result, is found. In computer science, such a rule is called an algorithm.

Solving a problem on a computer consists of several stages , the main ones being the following:

1. **Production tasks:**
   - collection information O task;
   - formulation conditions tasks;
   - definition final goals solutions tasks;
   - definition forms issuance results;
   - description data (their types, ranges quantities, structures And T. p.).
2. **Formalization (analysis And study tasks, models, performance it in the form of equations, relationships, constraints, etc.):**
   - analysis existing analogues tasks;
   - analysis technical And software funds;
   - development mathematical models;
   - development structures data.

Concept modeling

At decision tasks usually are exploring Not real object, A his model **.**

A model is an artificially created object that possesses all the essential characteristics of a real object, phenomenon or process.

Modeling is a method of cognition that consists of creating and exploring models.

Goals modeling:

1) understand essence studied object;
2) learn manage object And determine the best control methods ;
3) to predict straight or indirect consequences;
4) decide applied tasks.

A mathematical model is a system of mathematical relationships (data) – formulas, equations, inequalities, etc. and the relationships between them, describing the behavior of an object with a certain degree of accuracy and reflecting the essential properties of the modeled process (object or phenomenon).

Order compilation mathematical models:

- select a real object on which the mathematical model will be based, and from the multitude of its properties, patterns, internal connections, individual characteristics of the phenomenon and parameters, we select those that are essential for the problem being solved, and discard those that are unimportant;
- define, What count original data And results;
- select a mathematical object with the same number of similar parameters, reflecting the essence of a real object; write down mathematical relationships that link the results with the original data.

3. **Choice method solutions.**
4. **Development algorithm:**
   - choice method design algorithm;
   - choice forms records algorithm (block diagrams, pseudocode And etc.);
   - choice tests And testing method ;
   - design algorithm.
5. **Programming:**
   - choice language programming;
   - clarification ways organizations data;
   - recording algorithm on selected language programming.
6. **Testing, debugging And correction discovered errors:**
   - syntactic debugging;
   - debugging semantics And logical structures;
   - test calculations And analysis results testing;
   - improvement programs.

Debugging programs

Debugging a program is the process of finding and eliminating errors in a program based on the results of running it on a computer.

In modern software systems, debugging is accomplished using specialized software tools called debuggers. These tools allow **one to examine the internal behavior of a program** .

Debugger program usually provides next possibilities:

- **step-by-step execution of a program** with a stop after each command (operator);
- **viewing the current value of any variable or finding the value of any expression** , including using standard functions; if necessary, you can set a new value for the variable;
- **setting up "checkpoints" in the program** , i.e. points at which the program temporarily stops its execution so that intermediate results can be assessed, etc.

At debugging programs **important remember following:**

- At the beginning of the debugging process, you should use **simple test data** ;

- the difficulties that arise should be clearly **separated and resolved strictly one by one** ;
- **There is no need to consider the machine as the cause of errors** , since modern machines and translators are extremely reliable.

Test And testing programs

Testing is a test, a check of the correct operation of a program as a whole or its components.

Debugging identifies and corrects syntax errors and obvious coding errors, while testing verifies the functionality of a program that is free of obvious errors. Testing establishes the presence of errors, while debugging identifies the cause of the program's malfunction.

No matter how carefully a program is debugged, the decisive stage in determining its suitability for work is the program control based on the results of its execution on a test system.

A program can be considered conditionally correct if its execution for the selected system of test input data in all cases yields correct results. But, as the famous programming theorist E. Dijkstra argued that testing can only reveal the presence of errors, not their absence. It's not uncommon for new input data to cause a program that was previously considered fully debugged to fail or produce incorrect results.

To implement the testing method, reference results must be prepared or known in advance. Calculating reference results is essential. to, A Not after receipt machine results. IN otherwise In this case, there is a risk of involuntary adjustment of the calculated values to the desired ones, previously obtained by the machine.

7. **Analysis of the results of solving the problem and refinement of the mathematical model** by repeating stages 2–6 (if necessary).
8. **Program maintenance:** these are works related to servicing programs during their operation:
   - revision programs For solutions specific tasks;
   - compilation documentation To solved task, To mathematical models, to the algorithm, to the program, to the test set, to the use of the program.

1.2. Basics algorithmization

An algorithm is a method (technique) for solving a problem, written according to certain rules, in the form of a finite sequence of unambiguous instructions, the execution of which allows, using a finite number of steps, to obtain a solution to the problem, uniquely determined by the initial data from a certain set of values.

Properties algorithm

1. **Discreteness** (discontinuity, separateness). The algorithm should represent the problem-solving process as the sequential execution of a finite number of simple (or previously defined) complete steps.
2. **Clarity** For performer — those. executor algorithm must know, how to perform it.
3. **Certainty** (precision, determinism). Each rule The algorithm must be clear and unambiguous, containing actions on known data. Each action must be understandable to the performer (each algorithm assumes a specific performer).

Note : Often, the property of determinism of an algorithm is understood as the simultaneous fulfillment of the properties of accuracy and clarity.

4. **Efficiency** (or finiteness). An algorithm must solve a problem, achieving a specific result in a finite number of steps.
5. **Correctness.** The ability of an algorithm to produce exactly the desired result. Incorrectness can be explained by incompleteness of our understanding of the object's properties or an omission in the solution. To prove the correctness of an algorithm, the problem is often divided into blocks, and correctness is proven for each block, although such verification is not complete.
6. **Massivity** . The algorithm for solving a problem is developed in a general form, meaning it must be applicable to a certain class of problems that differ only in their initial data. The initial data may be selected from a specific domain, called the algorithm's domain of applicability.
7. **Universality** . The algorithm should be designed so that anyone can use it to solve a similar problem. (For example, the rules for addition and multiplication of numbers are valid for any number, not just specific ones.)
8. **Efficiency.** Selecting an algorithm that will execute in minimal time and with minimal resource expenditure.

Thus, the performer acts ***formally*** , i.e. is distracted from the content delivered tasks, A only strictly performs some rules, instructions and at the same time get the desired result.

Criteria quality algorithm

1. **Connectedness.** Determined quantity intermediate results. The higher the number of intermediate results, the lower the coupling.
2. **The size of an algorithm** is the number of operations or steps that must be performed, as well as the complexity of these steps.
3. **Logical complexity.** It is being determined quantity branches And cycles.

Order execution algorithm

1. Actions V algorithm are being carried out V in order their records.
2. It is forbidden change in some places none two actions algorithm.
3. It is forbidden Not having finished one actions cross over To to the next one.

Methods descriptions algorithms

1. ***Verbal-formulaic.*** Description of the algorithm using words and formulas in natural language.

The verbal method is not widely used for the following reasons:
- such descriptions strictly Not formalizable;
- suffer verbosity records;
- allow for ambiguity in the interpretation of individual prescriptions. "He met her in a field of flowers."

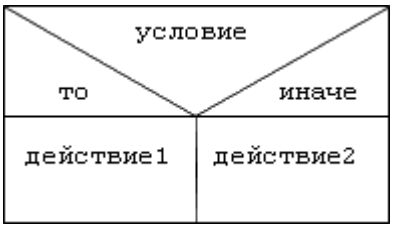
Example. Create a salary calculation algorithm according to the following rule: if the employee's length of service is less than 5 years, then the salary is 130 rubles, work experience from 5 to 15 years - 180 rubles, with experience over 15 years, the salary increases every year by 10 rubles.

Verbal-formulaic description algorithm solutions tasks:

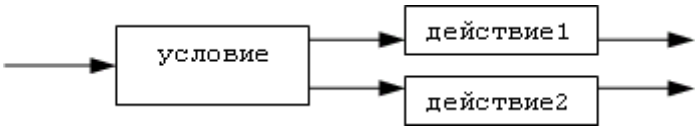
1. Enter *ST,* go to To p. 2.
2. If *ST<5, That ZP:.=l30,* go to To p. 4, otherwise - go to To p. 3.
3. If *ST* <15, That *ZP* :=180, go to To p.4, otherwise *ZP* :=180+(*ST* -15)10, go to To p. 4.
4. Withdraw (print) meaning *ZP,* go to To p. 5.
5. Calculations stop.

2. ***Tabular.*** Algorithm seems to be V form tables And calculated formulas (physics, chemistry, etc.).

3. ***Structogram***



4. ***Syntactic diagram*** **(** ***formulas) Backus- Naur*** **)**



5. ***Pseudocodes.*** Semi-formalized descriptions of algorithms in a conventional algorithmic language, incorporating both programming language elements and natural language phrases, commonly accepted mathematical notations, etc.

Pseudocode occupies a middle ground between natural and formal languages. On the one hand, it is close to ordinary natural language, so algorithms can be written and read in it like plain text. WITH another sides, V pseudocode are used some formal

constructions and mathematical symbols, which brings the recording of the algorithm closer to the generally accepted mathematical notation.

Pseudocode does not adopt strict syntactic rules for writing commands, which are inherent in formal languages, which makes it easier to write an algorithm in its design stage and allows for a wider range of commands. However, pseudocode, just like formal languages, contains function words whose meaning is defined once and for all.

There is no single or formal definition of pseudocode, so various pseudocodes are possible, differing in the set of function words and basic (base) constructions.

Example pseudocode is school algorithmic language.

An algorithmic language is a means for writing algorithms in an analytical form, intermediate between writing an algorithm in natural (human) language and writing it in computer language (programming language).

Recording algorithm solutions tasks example 1 on algorithmic language:

```
algae  salary (goal. ST , thing  ZP )
        arg ST
        cut ZP
beginning
        if ST<5
                then ZP:=150
                otherwise
                        if ST<=15
                                then ZP:=180
                                otherwise ZP:=180+(ST- 15)*10
                        everything
        eve
  horse ryt
        hin
        g
```

6. ***Software*** . Description algorithm With help languages programming.

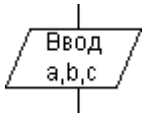
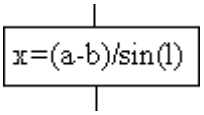
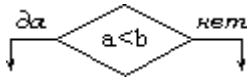
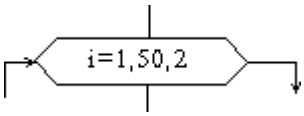
7. ***Graphical.*** The algorithm is depicted as a sequence. interconnected functional blocks, each of which corresponds to the execution of one or more actions. Such a graphical representation it's called scheme algorithm or **flowchart** .

The flowchart of the algorithm is a system of interconnected geometric figures.

Rules constructions block diagram

1. In the flowchart, each type of action (input of initial data, A geometric figure, represented as **a block symbol ,** corresponds to the calculation of expression values, condition checking, control of action repetition, termination of processing, etc. For clarity, operations of different types are depicted in the diagram using different geometric figures.
2. Block symbols are connected **by transition lines** , which determine the order in which actions are performed. The order of actions is indicated by arrows connecting the blocks.
3. IN scheme blocks they are trying place above down, V in order their execution.
4. All turns connecting lines are being carried out under angle 90 degrees.

IN table given most often used symbols.

Name	Designation and an example of filling out	Performed function (explanation)
Start/end (input/output)		Start or end programs, entrance or exit into a subroutine
Blocks input/output		Input/output data
		Conclusion data on printing device
Block calculations		Arithmetic block defines computational action or sequence of actions
Logical block		The logic block checks the truth or falsehood of a condition and selects the direction of execution algorithm V dependencies from conditions. The block must specify the question, condition, or comparison it defines.
Predetermined process		Calculations By standard or user subroutine
Block modifications		Execution actions, changing points of the algorithm, the beginning of the cycle. Inside block is being recorded parameter cycle, for which its initial value, boundary condition and step of changing the parameter value for each repetition are indicated.
Interpage connector		Instruction connections between in parts diagrams located on different pages

General rules constructions schemes algorithm of the problem

1. Identify original data, results, appoint them names.
2. Choose method (order) solutions tasks.
3. Break method solutions tasks on stages (With taking into account opportunities COMPUTER).
4. To depict every stage V view corresponding block schemes algorithm and indicate the order of their execution with arrows.
5. IN received scheme at any option calculations:

   A) provide for issuance results or messages about their absence;

   b) provide opportunity after execution any operations, So or otherwise, go to the *Stop block (to* the circuit output).

These rules constitute **the "Basic Principles of Algorithmization."** We will assume that knowledge and application of these "principles" are essential when developing an algorithm for any problem.

Types algorithms

- structured;
- unstructured (i.e. with violation of structure - with unconditional jump operators);
- auxiliary (used V composition others algorithms).

Types algorithms

- linear algorithm;
- branching algorithm ;
- cyclical algorithm;
- algorithm With subroutines;
- mixed (those. containing And cycles, And branching, And functions).
- recursive algorithm appeals To myself to myself, Bye Not a certain condition will be met.

Basic algorithmic designs

Structured programming – development methodology software software , which is based on the representation of the program in the form of a hierarchical block structures . Suggested V 70s years XX centuries E. Dijkstra , developed and supplemented by N. Wirth . Based on the structure theorem.

According to **the structure theorem** (theorem Boehm – Jacopini, 1966) logical structure any algorithm Maybe be presented combination three basic structures: *sequence, branching, cycle.*

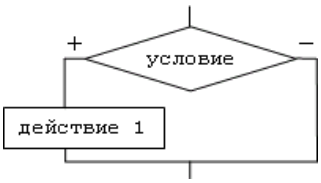
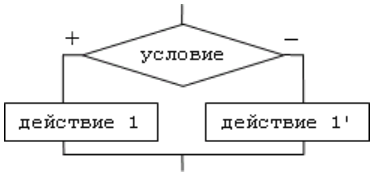
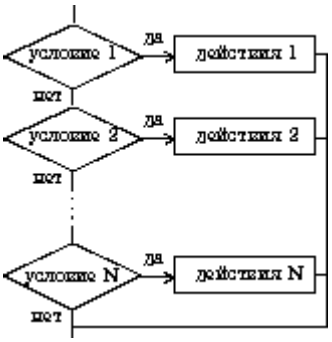
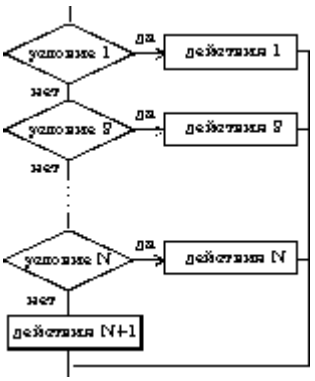
A characteristic feature of basic structures is the presence of one input and one output.

1. **The basic structure <u>of a sequence</u> (linear algorithm)** is formed from a sequence of actions following one after another:

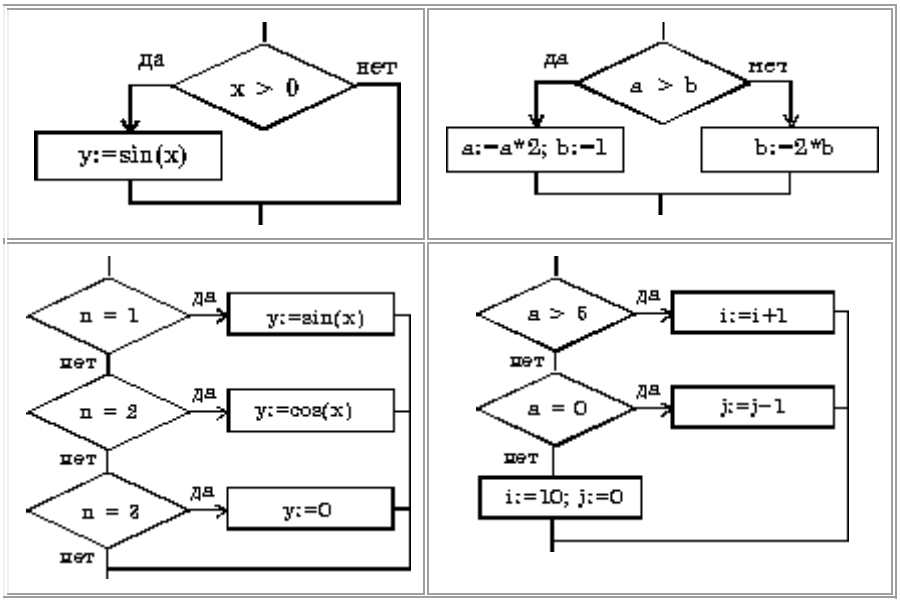


2. **<u>branching</u> structure (branching algorithm)** ensures that, depending on the result of checking a condition, one of the alternative paths of the algorithm's operation is selected. A condition is a question with two possible answers. Answer: **yes** or **no.** Each path leads to **a common exit** , so the algorithm will continue regardless of which path is chosen. Branching notation comes in two forms: complete and incomplete.

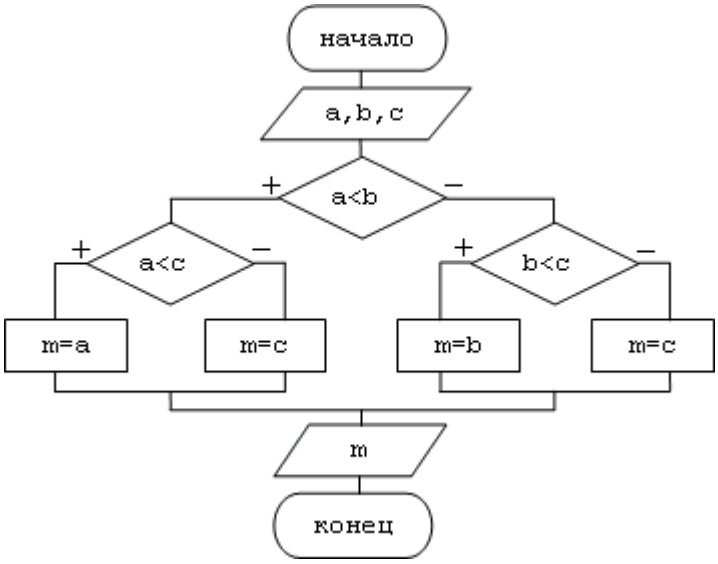
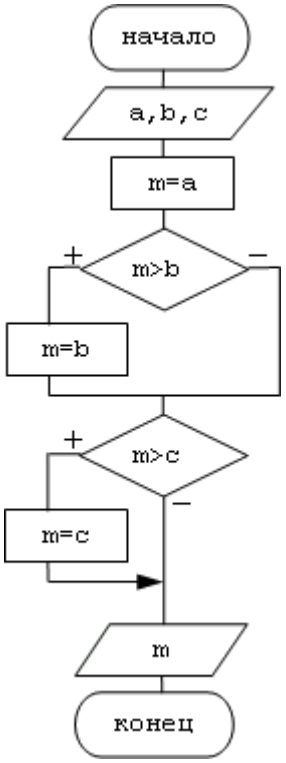
Structure **branching** exists V four main options:

Incomplete form records		**Full form records**	
1. **if-then**		2. **if-then - else**	
3. **choice**		4. **choice - otherwise**	

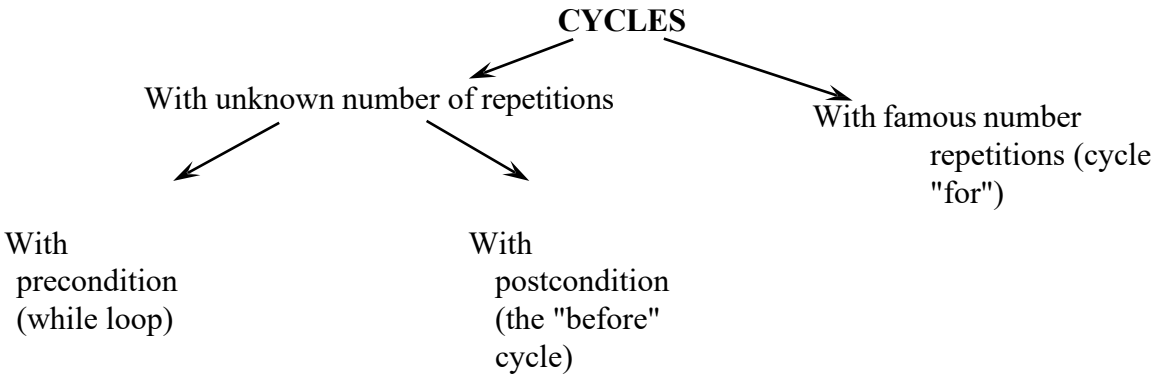
Examples teams *If*

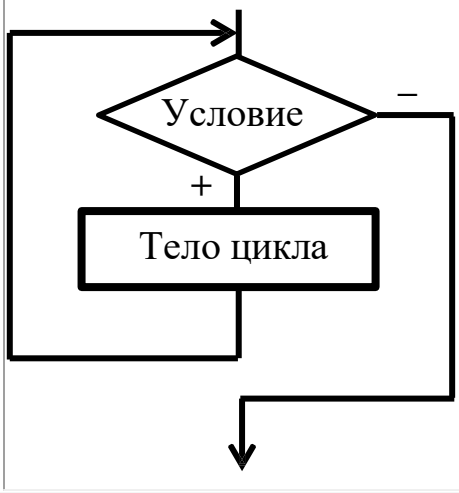


Example: find the smallest from three numbers.

1 option solutions:	2 option solutions:
	

3. **Basic structure <u>of the cycle</u>** ensures **the repeated execution** of some aggregates actions, which it's called **body cycle,** over new data.



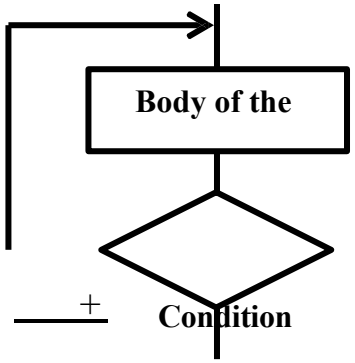


Цикл типа «пока» (Цикл с предусловием)

Выполнение цикла «пока» начинается с проверки условия, поэтому такую разновидность циклов называют цикламис **предусловием**. Переход к выполнению действия осуществляется только в том случае, если условие выполняется, иначе происходит выход из цикла. Тело цикла выполняется до тех пор, пока выполняется условие. Условие цикла необходимо подобрать так, чтобы действия, выполняемые в цикле, привели к нарушению его истинности, иначе

will happen looping.

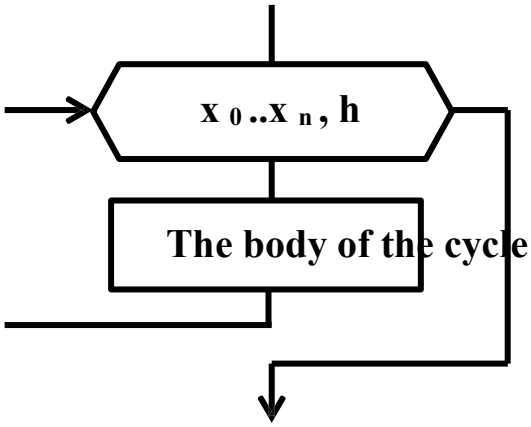
Looping - infinite repetition performed actions.



Cycle type "to" (Cycle With postcondition)

Execution Loop cycle it begins With execution actions. Such in this way, body cycle will implemented Although would once . After that, the condition is checked . Therefore, cycle "to" they call cycle **With postcondition** .

If the condition is met, then a return to execution of actions, otherwise an exit is performed – cycle. To prevent looping, it is necessary provide for actions, leading To falsities conditions.



Cycle type "For"

Cycle With parameter, or cycle with counter, or arithmetic cycle - This cycle With in advance famous number repetitions. Prescribes fulfill body cycle For all values of some variable (parameter cycle) within a given range.

x 0 - initial meaning parameter; **h** - step; **x n** – the last value of the parameter.

For creation cycles With parameter necessary use rules:

1. Parameter cycle, his initial And final values And step should be of the same type.
2. Forbidden change V body cycle initial, current and the final value for the parameter.
3. Forbidden enter V cycle, bypassing block modifications.
4. After exit from cycle meaning variable parameter uncertainly And cannot be used in further calculations.
5. From cycle Can go out, Not having finished his, Then variable the parameter retains its last value.

1.3. Theoretical basics programming

Programming is a branch of computer science that studies the methods and techniques of writing programs for computers. In addition, programming – this is the preparation of a problem for solving it on a computer.

A program *is* a sequence of actions that a computer must perform in a strictly specified order to achieve a specific result.

Programming languages are a set of tools and rules for representing an algorithm in a form acceptable to a computer.

Programming systems are a set of tools for entering, editing, translating and executing programs on a computer.

A translator is a device or a set of programs that translates a program written in a symbolic language into a set of machine commands, or transmits/converts data or another program.

For example , a translator accepts operators of one language and produces corresponding operators of another language.

A compiler is a translator that translates a program written in an algorithmic language into a set of machine commands without executing it on a computer.

The compiler evaluates the source text according to the syntactic construction of the language and translates it into machine language.

For example , a compiler takes a program written in C and converts it into an assembly language program.

An interpreter is a translator that ensures the translation of each construction of an algorithmic language into machine commands and the simultaneous execution of this construction in a computer.

When executing program statements, the interpreter must first scan each statement to read its contents and then perform the requested operation.

1.4. Rules records V C++ arithmetic expressions

Arithmetic expressions

An expression consists of operators and operands. Operands can be, in turn, expressions or one of its special cases – numbers (constants) or variables; **operators** denote the actions performed on them (+ addition, - subtraction, * multiplication, / division (for integer operands – the integer part of the division), % remainder of the division (only for integers), …).

All main operations C++ language Can break on next groups:

- arithmetic operations;
- logical operations;
- operations relationship;
- operations With bits information;
- operations with lines;
- operations assignment;
- operationsizeof;
- conditional operation (?:).

Examples expressions:

```
(a + 0.12)/6
x && || !z
(t * sin(x)-1.05e4)/((2 * k + 2) * (2 * k + 3))
```

The expression may include operands of different types, which is strictly speaking, Not follows admit. If operands have same type, That the result of the operation will have the same type. If the operands are of different types, before the calculations are being carried out transformations types By certain rules,

providing for the conversion of shorter types into longer ones to preserve significance and precision.

The order of expression evaluation is determined by the rank (priority) of the operations included in it (Table 3). The rank of operations adopted in C++ is closest to the mathematical one, as is the adopted order of their evaluation. Thus, multiplication and division (multiplicative operations) are higher than addition and subtraction (additive operations). Unary operations + and - are higher than binary ones, i.e., the sign of the operand is evaluated first. Assignment-type operations are lower than others, which allows them to be executed only after the value of the expression is fully evaluated. Relational operations are lower than arithmetic operations, which allows the use of natural notation of logical expressions, for example, `x>0 && y>0` . Here V first of all are calculated relational values, which are then the operands of the conjunction.

Table 1 - Order calculations expressions

Group	Type actions	Operations or elements
1	Calculations V in parentheses	( )
2	Calculations values functions	Functions
3	Unary operations	++, --, ~ , !, unary - And +, &, * (deaddressing), …
4	Operations type of multiplication	*,/, %
5	Operations type additions	+,–
6	Operations relationship	<,<=,>, >=, ==, !=

Arithmetic expressions are constructed from **operands, arithmetic operations, and parentheses.**

Parentheses are used to enclose the portion of an expression whose values must be evaluated first. An expression can contain any number of parentheses, but the number of opening parentheses must be equal to the number of closing parentheses. Subexpressions enclosed in parentheses must either be non-intersecting or nested within each other.

Arithmetic expressions are being recorded By next rules:

- Recording is being conducted V line.
- It is forbidden let down sign multiplication between factors.
- For designations variables are used letters Latin alphabet.
- Operations are performed according to *priorities* : first, functions are calculated, then multiplication and division, then addition and subtraction.
- When multiple operators of the same precedence are written in a single expression, unary operators, the conditional operator, and assignment operators are evaluated *from right to left* , while other operators are evaluated *from left to right* . For example, **a = b = c** means **a = (b = c),** and **a + b + c** means **(a + b) + c** .
- For changes order actions are used round brackets.
- When using standard functions, the argument must be in parentheses.
- IN language **C++** provided basic mathematical operations (cm. adj. 2). Examples of writing expressions:

Mathematical expression	Recording on C++
1. $10x + 3\sqrt{\cos x}$	1. `10*x + 3*sqrtf(cos(x))`
2. $\frac{\sin 2x - 1}{x^2 - 1}$	2. `(sin(2*x)-1)/(pow(x, 2)-1)`
3. $\lvert x \rvert + 2\,ctgx$	3. `abs(x) + 2*cos(x)/sin(x)`

Laboratory Job No. 1
Recording arithmetic expressions

PURPOSE OF THE WORK: to consolidate knowledge on theoretical foundations algorithmization and programming, acquisition of skills in using arithmetic operations, functions and rules records arithmetic expressions according to syntax of the C++ language.

EXECUTION WORKS: write down next expressions By rules language C++ .

Exercise I

1. $\sin 5x + 2\,\mathrm{tg}\,x^2$	5. $\dfrac{2xy + 4x^2}{5y}$;
2. $2x + 3y^2 - \sin 3x$	6. $4\sqrt{\cos 3\ b 2\sigma\iota\nu b}$;
3. $\sqrt{\dfrac{\sin y^2}{2\cos 2y}} + 2\lvert tgy - 25 \rvert$	7. $-2\cos x^2 + 2y^7$;
4. $\sqrt{1 - \dfrac{2}{x^2}} + \lvert x^5 - 3x \rvert$	8. $\dfrac{\lvert 2a + tg\,3\,d \rvert}{\sqrt{1 + b^8} + 11}$

Exercise II

Execute according to option:

1.	$a = \dfrac{2\cos\left(x - \dfrac{\pi}{6}\right)}{0.5 + \mathrm{son}^2\ y}$,	$b = 1 + \dfrac{with^2}{3 + \dfrac{With^2}{5}}$;
2.	$s = \left\lvert x^{\frac{y}{x}} - \sqrt[3]{\dfrac{y}{x}} \right\rvert + \ln x$,	$t = (y - x)\dfrac{y - z(y - x)}{1 + (y - x)^2}$.
3.	$s = 1 + x + \dfrac{x^2}{2!} + \dfrac{x^3}{3!} + \dfrac{x^4}{4!}$,	$t = x\left(\sin x^3 + \cos^2 y\right)$.
4.	$and = and^{-bt}$ without $\left(\sqrt{\lvert at + b \rvert}\right) -$ $bt + to$,	$s = b\sin\left(at^2 \cos 2t\right) - 1$.

5.	$w=\sqrt{x^2+b}-\dfrac{b^2\sin^3(x+a)}{x}$,	$y=\cos^2 x^3-\dfrac{x}{\sqrt{a^2+b^2}}$.
6.	$s=x^3\cdot tg^2(x+b)^2+\dfrac{\alpha}{\sqrt{x+b}}$,	$d=\dfrac{bx^2-a}{e^{ax}-1}$.
7.	$z=\dfrac{x(x+1)-2(+)}{b}\sin x a$,	$s=\sqrt{\dfrac{xb}{a}}-\cos^2(x+b)^3$.
8.	$y=\sin^3(x^2+a)^2-\sqrt{\dfrac{x}{b}}$,	$z=\dfrac{x^2}{a}+\cos(x+b)^3$.
9.	$f=\sqrt[3]{m\cdot tg\,t+\lvert c\cdot\sin t\rvert}$,	$z=m\cos(bt\cdot\sin t)+c$.
10.	$y=b\,tg^2x-\dfrac{a}{\sin\left(\frac{x}{a}\right)}$,	$d=a\cdot e^{-\sqrt{\frac{a}{}}}\cdot\cos\left(\dfrac{bx}{a}\right)$.
11.	$f=\ln(a+x^2)+\sin^2\left(\dfrac{x}{b}\right)$,	$z=e^{-cx}\cdot\dfrac{x+\sqrt{x+a}}{x-\sqrt{\lvert x-b\rvert}}$.
12.	$y=\dfrac{a^{2x}+b^{-x}\cdot\cos(a+b)x}{x+1}$,	$k=e^{\frac{ax}{\sqrt{2}}}\cdot\cos\sqrt{\dfrac{bx}{2}}$.
13.	$z\,\alpha\xi\sqrt{\cdot\sin 2x+e^{-2x}\cdot(x)+b}$,	$w=e^3\cdot\lvert\sin bx\rvert-\dfrac{x^3}{a}$.
14.	$u=\dfrac{a^2x+e^{-x}\cdot\cos bx}{bx-e^{-x}\cdot\sin bx+1}$,	$f=e^{2x}\cdot\ln(a+x)-b^{3x}\cdot\ln(b)-x)$.
15.	$z=\dfrac{\sin x}{\sqrt{1+m^2\sin^2 x}}-cm\cdot\ln(mx)$,	$s=e^{-ax}\cdot\sqrt{x+1}+e^{-bx}\cdot\sqrt{x+1.5}$.
16.	$a=e^{-mx}\cdot\dfrac{\sqrt{m+x^2}}{\lvert m^2-x^2\rvert}$,	$b=\ln\left(c^2+\sqrt{c+y}\right)+e^{-cy}$.
17.	$z=x^3\cdot\sin^2(x+b)+\dfrac{x}{(x+b^2)}$,	$s=(m^2+y^2)\cdot e^{my}+\sqrt{3m+\ln(m+4y)}$.

18.	$l = \dfrac{a\cos^3 bx}{\sin a + bx\cos b - e^a},$	$w = \dfrac{\sin\left(x^3 - a\right) + \lvert b\rvert x}{\cos(ax + b)}.$
19.	$y = \cos^2\left(x + a^2\right)^2 + \sqrt{\dfrac{x}{b^2}},$	$z = \dfrac{a^3}{3\,x^2} + and^{(x+b)^3}.$
20.	$s = \left\lvert x^{\frac{x}{y}} - 5\sqrt{\dfrac{x}{y}}\right\rvert,$	$\underline{t = mm}(x - y)\dfrac{\sin(x) + y) - tg(y - x)\cdot}{1 + (x - y)^2}$
21.	$s = \dfrac{e^{my}}{2 + 5x} + \sqrt{\dfrac{\ln(m-3)}{\lvert 2a\rvert}},$	$v = \sqrt{\lvert bt - a\rvert} - e^{\sqrt{bt}}\cdot\sin(bt + a).$
22.	$y = 1 + \dfrac{b\sin\left(at^2\cos 3t\right)}{12x},$	$f = \sqrt[3]{m\cdot tg\,t + \lvert c\sin 3t\rvert}$
23.	$and = to\cdot tg^2(x + a) - \dfrac{b}{\text{without}^2\left(\dfrac{}{x}\right)}(to),$	$d = b\cdot e^{-\frac{b}{\sqrt{}}}\cdot\cos\left(\dfrac{ax}{b} + 1.4\right).$
24.	$z = \sqrt{\dfrac{x^2}{b}} - \ln\left(^2 + x^2\right),$	$f = e^{-cx}\cdot\dfrac{x + \sqrt{\lvert x - b\rvert}}{x - \sqrt{x + to}}.$
25.	$s = \dfrac{\left(m^2 + and^2\right)}{5\,a} + \sqrt{\lvert m\rvert} + (tg\,m - 2and)$	$f = \dfrac{e^{2x}}{6\cdot\ln(a + x)} - b^{3x}\cdot\sin(b - x).$

Control questions

1. What What is an algorithm?
2. List properties algorithms.
3. Methods records algorithm.
4. List the basic algorithmic constructs . What is a flow chart?
5. What is it? block diagram complete (incomplete) forms teams branching?
6. What is it? block diagram cycle With precondition?
7. What is it? flowchart of the cycle With postcondition?
8. What is it? flowchart of the cycle With parameter?
9. What such program?
10. What such translator?
11. What such compiler?
12. What such interpreter?

13. What such operator and the operand?
14. IN what kind cases at records mathematical expressions on language Does C++ use parentheses?
15. How are used standard mathematical functions in C++?

2. Programming algorithms linear structures

2.1. General characteristic language programming C++

C++ - compiled statically typed language general-purpose programming.

It supports programming paradigms such as procedural programming, object-oriented programming, and generic programming. It provides modularity, separate compilation, exception handling, data abstraction, declaration of object types (classes), and virtual functions. The standard library includes, among other things, commonly used containers and algorithms. C++ combines features of both high-level and low-level languages. Compared to its predecessor, C, it places greater emphasis on supporting object-oriented and generic programming.

C++ is widely used for software development, being one of the most popular programming languages. Its applications include the creation of operating systems, various Applications, device drivers, embedded systems, high-performance servers, and entertainment applications (games). There are many implementations of the C++ language, both free and commercial, and for various platforms. For example , on the x86 platform , these include GCC, Visual C++, Intel C++ Compiler, Embarcadero (Borland) C++ Builder , and others . C++ has had a huge influence on other programming languages, most notably in Java and C#.

The syntax of C++ is inherited from the C language. One of the design principles was to maintain compatibility with C. However, C++ is not strictly a superset of C; the set of programs that can be translated equally well by both C and C++ compilers is quite large, but does not include all possible C programs [1].

2.2. Main concepts language

The basis of any programming language is formed by three components: alphabet, syntax and semantics.

An alphabet is a fixed set of basic symbols (letters of the alphabet) for a given programming language, which constitutes any text in that language. No other symbols are allowed in the text. All symbols in the text are encoded using the standard ASCII code table. Each ASCII character has a non-negative ordinal number, meaning the set of all symbols is ordered.

Alphabet WITH ++ includes:

1. Uppercase And lowercase Latin letters And sign underscores.
2. Arab numbers from 0 to 9.
3. Special signs:

“ {} , | [ ] ( ) + - / % * . \ ` : ? < = > ! & # ~ ; ^

4. Whitespace characters: space, tab characters, newline characters.

Syntax is a system of rules defining the permissible constructions of alphabetic characters. These constructions are used to represent individual components of an algorithm and the algorithm as a whole, written in a given programming language. Thus, for each sequence of alphabetic characters, syntax allows one to determine whether it is text in the given programming language.

C++ syntax defines how to form correct text and write programs in this language. A syntactically correct program compiles without errors.

From symbols alphabet are being formed *lexemes* language:

- **identifiers;**
- **key (reserved) words;**
- **signs operations;**
- **constants;**
- **separators (brackets, dot, comma, whitespace symbols).**

A lexeme , or elementary construction, is the smallest unit of language that has independent meaning. Lexeme boundaries are defined by other lexemes, such as separators or operation signs.

One of the most important syntactic concepts of any programming language is the concept **of an identifier** .

Identifier - *Name software object.*

The identifier may contain Latin letters, numbers, and underscores. Uppercase and lowercase letters are distinguished, for example, `alpha` , `Alpha` and `ALPHA` – three different names. The first character of the identifier may be a letter or an underscore, but not a number. Spaces are not allowed within the names.

To improve program readability, objects should be given meaningful names. There is a naming convention called Hungarian notation (since it was proposed by a Hungarian Microsoft employee), according to which each word that makes up an identifier begins with a capital letter and is preceded by a prefix corresponding to the value type, for example, `lpfnSetFirstDialog` . Another tradition is to separate the words that make up a name with underscores: `max_length` , `number_of_galosh` [2].

The standard doesn't limit the length of an identifier, but some compilers and linkers impose restrictions. An identifier is created when declaring a variable, function, type, etc., and can then be used in subsequent program statements. When choosing an identifier, keep the following in mind:

– the identifier must not coincide with the keywords and names of the standard language objects used;

- It is not recommended to start identifiers with an underscore character, since they may coincide with the names of system functions or variables, and, in addition, this reduces the portability of the program;
- Identifiers used to define external variables are subject to linker restrictions.

Examples of correct identifiers: `value25, My_File, index, Operation` . Examples of syntactically incorrect identifiers:

`1_Side` (Name Not should start off from the number),
`Volume 2` (no spaces allowed),
`New-Name` (Not allowed hyphen),
`break` (service word),
`exp` (predetermined Name exponential functions).

Note, what are the names, constants, service words it is forbidden write together with each other, they must be separated from each other by a space, a line break, or a comment.

Reserved (keywords) are identifiers that have special meaning to the compiler. They can only be used in the sense in which they are defined. For example **,** `const` – a keyword that indicates that a value cannot be modified, `int` – a keyword used to denote an integer data type.

Operator signs. An operator sign is one or more symbols that define the action on the operands. Spaces are not allowed within the operator sign. Operators are divided into unary, binary, and ternary. the number of operands involved. The same sign can be interpreted differently depending on the context. All operation signs except [ ] , ( ) and ? : are separate lexemes.

The semantics of a language is the semantic meaning of words. In programming, this is the initial semantic meaning of operators, basic language constructs, etc. [1].

Example . First code: `i=0; while(i<5){i++;}`
Second code : `i=0; do{i++;}while(i<5);`

Logically, these two code fragments perform the same function, and their results are identical. However, semantically, they are two different loops.

Like this in this way,

- **syntax** is a formal set of rules that determines the method of constructing any language constructs;
- **Semantics** is a set of rules that determine the meaning of syntactic constructions.

Comments are text that is ignored during compilation, which can be useful for programmers to clarify certain sections of code. The compiler treats comments as white space. Comments can also be used to temporarily disable parts of the code during debugging.

Comments V program on C++ or are beginning With two symbols "forward slash" (// `single-line comment`) and ends with an escape character on new line, or consists of between symbols - brackets

/* `multi-line comment` */. You can use any characters valid on your computer within a comment, not just characters from the C++ alphabet, since the compiler ignores comments. Nested comments (brackets) are not allowed by the standard, although they are allowed in some compilers.

2.3. Data And methods their organizations

Data - This information, presented V view suitable For automatic processing means, V particulars COMPUTER, at possible participation person.

IN programs processed data appear V quality values certain program objects. Data that is fixed in the text and does not change during program execution is called **constants** ; other data is the values of objects, called **variables** , since the values of these objects can change during program execution.

The data a program works with is stored in RAM. The compiler needs to know exactly how much space it takes up, how it's encoded, and what operations can be performed on it. All of this is specified during the description. data With help type.

Type data definitely defines:

1. Format performances data V memory computer.
2. The set of valid values accepted by a variable or constant belonging to a selected type.
3. The set of valid actions applicable to this type (operations and functions).

For example, whole And material numbers, occupy same volume memory (`longint` And `float`), But have different ranges possible values.

Table 2 - Classification types data C++

Basic	**Determined programmer**	
Simple (scalar)	**Simple**	**Compounds**
Logical (bool) Whole (shortint, int, longint) Real (float, double, long double) Character (char, wchar_t) Empty type (void)	Enumerations Pointers Links	Arrays Structures Unions Classes

Constants V language C++

So, a constant represents a value that cannot be changed. Constants have a type, and the type is determined by the constant's notation. The syntax of the language distinguishes five types constants: whole, valid (real), symbolic, enumerable, null pointer.

For example:

```
constint k=1;                                    // integer named constant
```

```
constfloat Pi = 3.1415926, Eps = 0.1e-5;   // valid constants
constchar a = 'a';                         // symbolic constant enum
DAY  {SUNDAY, MONDAY, TUESDAY, WEDNESDAY,
          THURSDAY, FRIDAY, SATURDAY};     // enumeration
char *pc = nullptr;                        // null pointer
```

Order announcements And initialization variables

A variable is a named memory location that stores data of a specific type. type. Each one variable should be announced V text programs before first her use. For beginners recommended declare variables in beginning bodies programs before first feasible operator. At variable declaration indicated her type And Name:

```
type_name     variable_name;
```

When describing Can appropriate variable initial meaning, This it's called **initialization** .

For example:

```
int d = 5;  // Integer variable d is initialized to 5
float b;   // A real variable is declared b
char s = '+'; /* Variable s announced How symbolic And initialized to '+' */
```

Can simultaneously announce some variables, Then V list names they separate Friend from friend comma, For example:

```
int a, b, c;           // Signed integers
char ch, sh;           // Single-byte characters
long l, m, k;          // Long integers [-2147483648 – 2147483647]
float x, y, z;   // Real
longdouble u,v,w; // Long double- precision floats
```

So, before using a variable , four characteristics need to be defined :

- Name;
- type;
- meaning (Not Necessarily);
- region visibility.

Name variable (identifier) means V text programs a quantity that changes its meaning. For each variable in the computer's memory some stands out region memory, capable keep meaning. Name allows gain access To areas memory, storing meaning variable.

type determines the amount of memory allocated and how the value is stored.

Variable value . If a variable is not initialized to an initial value, its value will be undefined before its first use . Regardless from Togo was given or No initial meaning a variable (when it is described or when it is entered from the keyboard, for example) its value can change V in the course execution programs.

Region visibility. Variables V language Si are classified By scopes and are divided into two categories: local and global.

A local variable is a variable declared within a block (in C, a block is a sequence of statements enclosed in curly braces). Memory for this variable is allocated when its declaration statement is executed. Upon completion of the block, this variable is destroyed, and the memory it occupied is freed. In other words, the lifetime of such a variable is limited to the block in which it is declared.

A global variable is a variable declared outside of any block. Therefore, its lifetime is the lifetime of the program itself.

Another difference between a local variable and a global variable is their initialization procedure. Any global variable is automatically initialized to zero when declared, but a local variable is not. Therefore, if the initial value of a local variable is not yet known when declared, it is best to force it to zero.

To summarize All From the above, we can conclude that before than to describe any data, necessary execute next actions:

- choose Name variable (describe identifier);
- determine the variable type based on the required range of number representation ;
- define region visibility variable;
- reset or initialize variable initial meaning.

2.4. Standard simple types data

In any algorithmic language, every constant, variable, expression result, or function must have a specific type. All C++ types can be divided into basic and composite. C++ defines six basic data types for representing integer, real, symbolic, and logical values. Based on these types, the programmer can define composite types.

Basic (standard) data types are often called *arithmetic* because they can be used in arithmetic operations. The following keywords are defined to describe the basic types:

- **`int`** (whole);
- **`char`** (symbol);
- **`wchar_t`** (extended) symbolic);
- **`bool`** (logical);
- **`float`** (real);
- **`double`** (real With double accuracy).

The first four types are called integers , the last two are *floating-point types.* The code that the compiler generates for The code for handling integer values differs from the code for floating-point values.

There are four type specifiers that specify the internal representation and value range of standard types:

- **short** (short);
- **long** ;
- **signed** (iconic);
- **unsigned** .

By default, all integer types are considered signed, meaning the signed specifier can be omitted.

The basic types in the C++ language type system are the simple types presented in the following table:

Table 3 - Standard simple types data

Simple types	Type	Name	Size, bytes	Range
Logical	bool	Boolean (logical)	1	false, true (0, 1)
Symbolic	signed char	symbol	1	-128 … 127
	unsigned char	unsigned symbolic	1	kit acceptable symbols With codes 0 … 255
Whole	short int	short whole (short)	2	-32 768 ... 32 767 (-2^{15} .. 2^{15} - 1)
	unsigned short int	unsigned short integer (word)	2	0 ... 65 535 (0 .. 2^{16} - 1)
	long int	long whole (long)	4	-2 147 483 648 … 2 147 483 647
	unsigned long int	unsigned long whole	4	0 … 4294967295
Real	float	real	4	3.4e-38 … 3.4e+38
	double	double accuracy	8	1.7e-308 … 1.7e+308
	long double	extended	10	3.4e-4932 … 3.4e+4923

For real types, the table shows the absolute values of the minimum and maximum values.

The ANSI standard does not specify ranges of values for the basic types; it only defines the relationships between their sizes, for example:

```
sizeof (float) <=sizeof(double) <=sizeof(long double)
sizeof (char)<=sizeof(short) <=sizeof(int)<=sizeof(long)
```

A derived type (programmer-defined) is a type that must be declared by the user before use. These include arrays, enumerations, functions, structures, references, unions, and classes.

Primary in the type hierarchy are the standard scalar types, including which include integer, real, symbolic, Boolean types and their modifications, taking into account the architectural features of the hardware.

1. The Boolean (logical) type is denoted by the standard identifier **bool** . In C++, there are two Boolean constants, which are also denoted by standard identifiers: **true** and **false** . The internal representation of the value false-0 (zero). Any other value is interpreted as true. When converted to an integer type, true has the value 1. Comparison operations are allowed on Boolean values, moreover, counts, What false < True. Except Togo, there are four,

standard logical operations, denoted by the symbols **&&** (conjunction), **||** (disjunction), and **!** (not – logical negation). These operations have the following precedence: subsequence **! - && - ||** . Values Bulevsky type occupy one byte of memory.

2. The character type has the standard identifier **char** . The values of variables of this type are characters from the ASCII set. This set consists of 256 different characters, ordered in a specific way, and contains uppercase and lowercase letters, digits, and various other symbols, including special control characters. If a character value has a graphic image, it is represented by the corresponding character enclosed in apostrophes, for example: '*', '1', ';'. The **char type** , like other integer types, can be signed or unsigned. Values of the char type are also used to store integers that do not exceed the boundaries of the specified ranges (see Table 3).

The wide character type (wchar_t) is designed to handle character sets that cannot be encoded in a single byte, such as Unicode. The size of this type is implementation-dependent; it typically corresponds to the short type . String constants of type wchar_t are written with the prefix L, for example, L”Gates”.

3. Integer types – this group of types represents sets of integers in various ranges. C++ has five integer types, differing in their range of values and the amount of RAM they occupy. Integer types are designated by the identifiers **int , shortint , longint , unsignedshortint , and unsignedlongint** ; their characteristics are listed in Table 3. The size of the int type The size of the int type is not defined by the standard, but depends on the computer, compiler, and operating system. For a 16-bit processor, 2 bytes are allocated for values of this type; for a 32-bit processor, 4 bytes. Therefore, when writing programs that will be portable across different platforms, one should not make assumptions about the size of the int type . To obtain it, one must use the sizeof operation , which returns the size of the type in bytes.

A number is of integer type if and only if its notation contains neither a decimal point nor an order sign e.

Constants found in a program are assigned a type according to their type. If for some reason this type is not suitable for the programmer, he can explicitly indicate the required type using the suffixes L, l (long) and U, u (unsigned). For example, the constant 32L will be of type long and occupy 4 bytes. You can use suffixes L and U at the same time, for example, 0x22UL or 05Lu.

Four arithmetic operations are allowed on integer values, each with its usual meaning and denoted by symbols: + – addition, - – subtraction, * – multiplication, / – the integer part of division. In addition, in C++ Can use more unary operations (++, --, unary minus And plus,
...), division-type operations: % – finding the remainder of a division, assignment operations (=, *=, /=, %=, +=, -=). All of the above operations with integer operands produce an integer result. Also, with integer data It is possible to perform bitwise, logical and comparison operations.

4. Real types - this group of types represents sets of real values in various ranges. C++ supports three real types, which are denoted by the standard identifiers **`float` , `double` , `longdouble`** (see table. 3). Real numbers can be represented in two forms: in fixed-point form (whole and fractional parts) separated from each other by a period, e.g. -19.324, 0.1281) and in floating point form (<mantissa>e<order>, e.g. 16.2e21, 2e-48).

Floating-point constants are of type double by default. You can explicitly specify the constant's type using the suffixes F, f (`float`) and L, l (`long`). For example, the constant 2E+6L will be of type `longdouble` , and the constant 1.82f will be of type `float` .

Four arithmetic operations are allowed on real numbers: addition, subtraction, multiplication and division; assignment and comparison operations.

For real data types, as well as for integer data types, it's possible for the result of an arithmetic operation to be outside the acceptable range. In this case, the computer will return a result converted to the type described for the given result, which bears no resemblance to the true value.

5. `void` type . In addition to those listed, `the void` type is one of the language's fundamental types , but the value set of this type is empty. It is used to define functions that do not return values, to specify an empty list of function arguments, as a base type for pointers, and in type casting .

2.5. Structure programs

A C++ program consists of *functions* , *declarations* , and *preprocessor directives* . One of the functions must be named *main* . Execution The program begins with the first statement of this function.

Format simplest definitions functions:

```
return_type  function_name([parameters]) // function header
{
// Function body block :
object definitions ;
// executable operators;
// return expression;              // if the function returns a value.
}
```

Typically, a function is used to calculate a value, so its type (the return type) is specified before the function name. If the function should not return a value, the **`void type is specified`** . The function body is block And, hence, consists of curly Brackets. Any function other than `main` can be called from within another function. Examples of functions are library functions, such as **`sin`** (x), **`fabs`** (a).

Example structures of the simplest program	**Text programs, consisting from one functions And containing components that may be almost all absent**

1. Preprocessor directives	`#include<iostream> /* inclusion of a directive (header file) For organizations input-output V C++ */`
2. Descriptions	`using namespace std; // description spaces names std`

3. Definition of the main function	`int main() // title main functions`
	`{`
	`// Block bodies functions`
	`// Here located program code`
	`}`

A block (compound statement) is an arbitrary sequence of definitions and statements enclosed in curly braces. A block is used to enlargements structures programs. Dot With comma V at the end block Not is put.

C++ program text has a structure. The general principles for writing a syntactically correct program are as follows:

1. **Each statement ends with a ";" character.** A common beginner's mistake is the ";" character ending function headers or loop statements. In the first case, the syntax error is recognized How absence bodies functions, in second case body cycle is an empty statement, which is not a syntax error, and the program is executed.
2. Before using functions, variables, data types, you should declare them or include files with their declarations (make (known to the compiler). In C++, a variable (object) can be declared not only at the beginning of a program, but also anywhere in the text before it is first accessed. The scope of such an object is only the block immediately enclosing it. Typically, working variables are declared this way.
3. In order for a function call to be executed, the function must be defined, i.e. the actions it performs must be described.
4. When defining a function, it is useful to follow these guidelines:

– announce And initialize entrance And weekend variables;
– intermediate variables declare And initialize at the beginning blocks or before the statements in which they are used.

5. In any sequence of actions, you need to strive for a triad: initialization (input), processing, returning a value (output).

Example structures programs, containing 2 functions: main And mysum.

1	Comments area	`// my first program`
2	Zone directives preprocessor	`#include <iostream.h> #include <conio.h>`
3	Custom Prototype Declaration Zone functions And global variables and data structures	`int a = 10;` `int mysum( int x);`

4	DOT ENTRANCE IN THE PROGRAM
	Zone main() functions
		`int main()`
		`{`
		`  int . = 15,`
		`  c; c =`
		`  mysum(b);`
		`  cout<<c;`
		`_getch();`
		`}`

5	Zone definitions user-defined functions	`int mysum( int x)`
		`{`
		`  int z = x + a;`
		`  return z;`
		`}`

When analyzing the given program, the following major parts of a typical source module can be identified:

- *connection* headings files libraries;
- *declarations* of user-defined symbolic constants, auxiliary functions, classes and data types, global variables;
- *title main functions;*
- *definition main functions* - block, containing:
  - *announcement* local variables And constants And their initialization (assigning initial values);
  - *input* original data (dialogue With user);
  - *processing* – calling functions , evaluating expressions, executing operators;
  - *conclusion* result;
  - *return code completion main functions;*
- *definitions custom auxiliary functions* And methods classes; for each function, after the header, there is a block containing:
- *announcement* local variables And their initialization;
- *processing* formal parameters With using local variables;
- *return* result.

This structuring of the source module text, which is located in *a single* file with *the cpp extension* , is not the only possible approach. However, following this pattern makes programs easier to understand and debug.

The rules for formatting program text, aimed at facilitating understanding of the meaning and increasing clarity, are as follows:

- divide logical parts programs empty lines;
- divide operands And operations spaces;
- For each curly brackets take away separate line;
- V each line should be, How rule, Not more one operator;
- limit length lines 60-70 symbols;
- indents left reflect nesting operators And blocks;
- long operators to arrange V several lines;

- carry out algorithmization in such a way that the definition of one function takes up, as a rule, no more than one screen of text;
- strive to use standard templates of program fragments, including the standard structure of the block and the program as a whole.

Certainly, these rules need to use together With *rules naming* elements programs And *commenting* text programs:

- **comments V text necessary;**
- **The names of program objects are chosen thoughtfully.** Each name emphasizes the purpose and logic of the object, for example, the names of the library functions `sin` , `abs` , `printf` and others speak for themselves. The names of objects entered by the programmer emphasize their abstract meaning, for example, `Count` , `Square` , `Point.x` , `Point.y` and so on.

2.6. Operators

An operator is a basic element of a programming language, representing a complete phrase and defining a certain logically complete stage of data processing.

Operators can be conditionally divided into two categories: *executable ones* - with their help the algorithm of the problem being solved is implemented, and *descriptive ones* , necessary for defining user types and declaring program objects, for example, variables.

Among executable operators Can highlight:

- simple operators (operator-expression, operators transmissions management);
- composite operators (composite operator, block)
- structured operators (conditional, choice, cycle). Let's consider

simple operators

Operator "expression"

Any expression ending with a semicolon is considered a statement, the execution of which consists of evaluating the expression. A special case of an expression is *the empty operator* (;). It is used when the syntax requires an operator, but the meaning does not.

Examples:

```
x ++;           // expression - operator; is being carried out operation
increment c = a * 1.5;  // expression with assignment operation
a *= b + c;     // is being carried out multiplication With assignment: a
= a * (b + c) fun(i, k);      // the function is called
```

assignment operator in the C++ language; it is replaced by an expression that contains **the assignment operation** .

General view:

```
identifier = expression;
```

Example:

```
x = y + 4; // expression with assignment operation
```

In progress V two stages:
a) the value of the expression on the right side of the assignment sign is calculated ;
b) calculated meaning is assigned variable, standing V left parts from the assignment sign, in which case the previous value of this variable is lost.

It is recommended to be strict about data types, not to mix types in expressions, and to ensure that the type of the left operand of an assignment matches the type of the expression on the right side.

When performing an assignment, if the types of the left and right sides do not match, is happening implicit transformation. C++ Always is trying This to do, and in simple terms, we can consider that the conversion occurs without data loss from a smaller type to a larger one, for example from `int` to `float` or from `char` to `int` , and with data loss from a larger type `to` smaller, for example, from `float` to `int` . This is easy to understand if you remember that the data type is the amount of memory it occupies.

Obvious transformation types is being carried out at assignment type:

```
name = (type) expression;
```

For example:

```
int         a = 5, b = 2;
float       c;
c = a/b;
```

It would be tempting to think that the value of c would be 2.5, since it is a real number, but the order of operations is such that division comes before assignment, and it is performed with integer operands, and its result is 2, meaning that the type conversion will only be performed during assignment.

For obvious transformations types is used famous reception:

```
c = ( float ) a/( float ) b; // c = 2.5
```

The assignment operation is right-associative, i.e. calculations are performed from right to left, so it is possible to write chains of assignments, for example:

```
x = y =z = 1; // Each variable will be equal to 1.
```

In *complex assignment operations* (+=, *=, /=, etc.), the L-value from the left-hand side is also used when evaluating the expression on the right-hand side. For example, in addition with assignment, the first operand is added to the second operand, and the result is written to the first operand. That is, the expression `a += b` is a more compact notation for the expression `a = a + b.`

Another type of operator that can be classified as an expression operator is This *operator appeals To functions* (call functions). His The operands are the function parameters.

format :

```
Name functions ([list of actual parameters] – may be absent);
```

For example:

```
scanf("%d%f", &my_int, &my_float);                    // input data
printf ("Whole = %d, Material = %f\n", my_int, my_float) //
conclusion
```

Spaces in a line of text are significant, that is, if you enter the integer 5 and fractional 9.9, then the output line will look like:

```
Integer = 5, Real = 9.900000
```

Operators transmissions management let's consider Later (p. 3.6, p. 4.1)

2.7. Organization input/output data

To input a given value means to assign an arbitrary value to a variable during program execution. **To output a given value** means to print the variable's value on the screen during program execution.

C++ has no built-in I/O capabilities—it's handled through functions, types, and objects contained in standard libraries. Two methods are used: functions inherited from the C language and C++ objects.

Organization input/output V style WITH

The simplest method of input and output (exchange) data is formatted, with defined rules for data placement in the input and output streams. Implementing this type of exchange requires the stdio.h library (standard input/output library), which is included in the program with the #include <stdio.h> directive.

To enter a value from the keyboard (with echoing on the screen), the scanf function is used, the syntax of which is:

```
scanf("format string", input_list);
```

Here the "input list" is the names of the variables whose values will be entered from the keyboard when executing the scanf function. The variable names are preceded by the & symbol, which is a flag for an address operation and means that the entered value is sent to the address specified variable name. When entering data, separate it with spaces or by pressing Enter.

To display the value of this data on the screen, the printf function is used, the syntax of which is:

```
printf("format string", output_list);
```

Here, the "output list" is a list of variable names and expressions (including constants) whose values will appear on the screen when the `printf function is executed` .

A format (control) string is a string of characters enclosed in double quotation marks, containing control characters and text. When inputting data, the scanf function reads the text from the input stream character by character, recognizes the tokens, and converts them to machine representation according to the format flag associated with the variable expecting the data. When outputting, the `printf function` takes the machine representation of the variable's value and, according to the format flag, converts it to text representation and displays it on the screen.

Number managers symbols equals number objects V list input- output.
Control symbol has sign %, And one from following values:

```
%d – input/output of an integer decimal number        (int);
%u – input/output of unsigned integer                 ;
%f – input/output of floating point numbers           (float and double);
%e – input/output of a number in exponential form     (double and float);
%c – input/output of character                        (char);
%l – input/output of a long value                     ;
```

and others.

At input And conclusion necessary strict correspondence type entered given the format control character.

Example formatted input And conclusion.

```
# include <stdio.h>
void main(void)
{
```

```
        int        my_int;
        float my_float;
        printf("\nEnter whole And fractional
        number\n"); scanf ("%d", &my_int);
        scanf("%f", &my_float);
        printf("%d %f", my_int, my_float);
}
```

When the program starts, it will display a prompt for data entry, then wait for data entry each time it executes scanf. The user must enter the required amount of data, separating each entry with spaces or by pressing the Enter key. Once the input is complete, the data will be immediately displayed on the screen in the most primitive manner. So, if you enter integer 5 and fractional 9.9, then the output line will look like this:

5 9.900000

Since the scanf function is waiting for input when entering data, it is recommended to precede each input with a prompt that displays a data entry prompt, detailing to the user what and how to enter the data correctly. This simple technique will significantly improve the interface of any program.

When outputting data, it is recommended to use some techniques to improve readability.

1. Managers symbols flow, For example:

```
\n for line feed in output;
\t to perform tabulation.
```

2. Arbitrary text in the format string for prompting for input and for explanations during output, for example, the output function can be written like this:

```
printf(" Integer = %d, Real = %f\n", my_int, my_float);
```

Spaces in a line of text are significant. Now, if you enter the integer 5 and fractional 9.9, then the output line will look like:

```
Integer = 5, Real = 9.900000
```

3. Modifiers formats. They are used For design conclusion. By default (without modifiers), data is displayed in a minimum-width field with a precision of 6 decimal places, aligned to the right edge of the field. This display can be controlled.

a) Field width is a string of numbers that defines the smallest size of the output field (positioning). Any number not included in the field is ignored.

b) Output precision, two digits that determine the total size of the output field and the number of decimal places. Used for real numbers.

IN examples let's designate familiar ␣ spaces, which will be V line output.

`printf ("Integer = %4d, Real = %5.2f\n", my_int, my_float);` If you enter the values 10 and 2.3, the output line will look like this:

```
Integer = __10, Real = _2.30
```

If enter values 19951 And 12.9999, That line conclusion will have the form:

```
Integer = 19951, Real = 13.00
```

It can be concluded that the integer, despite the output field being limited to 4 characters, will be output unchanged, and the real number will be rounded to hundredths.

c) The minus sign is used to left-justify the number within the output field.

`printf ("Whole = %-4d, Material = %-5.2f\n", my_int, my_float);` If you enter the values 2 and 2.36666, the output line will look like this: `Integer = 2___, Real = 2.37_`

If enter values -1999 And 12.9999, That line conclusion will have view:

```
Integer = -1999, Real = 13.00
```

Example use formatted input- output.

```
# include <stdio.h>
# define STR "Program"            // To illustrate the output of lines.
void main ( void )
{
// Output the integer 336
   printf("%d\n", 336);    // 336
   printf("%2d\n", 336);  // 336, 2d format is ignored
   printf("%8d\n", 336); // _____336 // output field width = 8
   printf ("%-8d\n", 336); // 336_____ // pressed leftwidth fields8
   printf("\n");           // Skip a line when outputting
// Output the real number 12.345
     printf("%f\n",         12.345); // 12.345000
     printf ("%e\n",12.345); // 1.234500e+01 printf
     ("%10.1f\n", 12.345);  // _____12.3 printf
     ("%-12.1f\n", 12.345);// 12.3 printf("\n");
// Output a character string by format s
     printf("%s\n", STR);             // Program
     printf ("%12s\n", STR);// ___Program
     printf("%12.5s\n", STR);         // _______Prog
     printf ("%-12.5s\n", STR);       // Program
     printf("\n");
}
```

Organization input/output V style C++

To manage input/output of data in C++ style, the header is used file `<iostream.h>` . IN him defined standard the stream objects `cin` for keyboard input and `cout` for screen output, as well as the operations of putting into a stream and reading from a stream.

```
cin»input_list; //input values this With keyboard
cout"output_list; //output the value of this data to
the screen
```

Example programs, using functions input/output V style WITH:

```
# include <stdio.h>
int main()
{
   int i;
   printf(" Enter whole number \n");
   scanf("%d",&i);
   printf(" You introduced number %d, thank
   you !",i); return 0;
}
```

A Here How looks like same program With using libraries classes C++:

```
# include <iostream.h>
int main()
{
   int i;
   cout«" Enter whole number \n";
   cin»i;
   cout«"You entered the number "«i«", thank you!";
   return 0;
}
```

Formatted input/output V C++

The ability to control input/output in C++ is provided by formatting member functions, flags, and manipulators. Flags, functions, and manipulators all perform the same task: they specify a specific format for input/output information in streams. Input/output to/from the screen in C++ is performed With help `cin` operators And `cout` respectively, This means that formatting manipulators are used in conjunction with these input/output operators. The difference between functions, flags, and formatting manipulators lies in how they are used. Now let's look at how formatting objects are used.

```
//Basic formatting member functions: cout.fill('/*symbol*/');
                                // sets symbol aggregate
//where symbol is a placeholder symbol, the symbol is passed in single quotes

cout.width(/*width_field*/); // sets the field width
//where width_field is the number of positions (one position holds one
character)

cout.precision(/*number*/); //sets the number of digits after the decimal
// points, where number is the number of digits after the decimal point
```

Functions are accessed using the **dot operator** , and the argument is passed in parentheses. The argument `to the fill() function` It can be passed as a character enclosed in single quotes or as a number (character code). Functions alone are not sufficient for formatting input/output streams, so C++ provides another formatting method: flags.

Formatting flags allow you to enable or disable one of the input/output options. To set the I/O flag, you must call the function `setf()` , if it is necessary to disable the output flag, then use function `unsetf()` . Next shown designs installations And removing output flags.

```
// setting the output flag
cout.setf( ios::/*name_flag*/ );
// where name_flag is the name of the flag
```

Access to the output operator functions is performed through the dot operator. Method setf() takes one argument – the flag name. Output flags are declared in the class ios, so before accessing the flag, you need to write the class name - ios, after which, using the scope resolution operation, call the required flag.

```
// clearing the output flag
cout.unsetf( ios::/*name_flag*/ );
// where name_flag is the name of the flag
```

If you need to set/clear several flags during input/output, you can use the bitwise logical OR operation. |. In this case, the C++ language construct will be as follows:

```
// setting multiple flags
cout.setf( ios::/*name_flag1*/| ios::/*name_flag2*/| ios::/*name_flag_n*/);
// removing multiple flags
cout.unsetf( ios::/*name_flag1*/| ios::/*name_flag2*/| ios::/*name_flag_n*/);
```

Table 4 describes some of the basic formatting flags and shows examples of their use.

Table 4 - Some flags formatting V C++

Flag	Purpose	Example	Result

`scientific`	Conclusion numbers With floating point V exponential form	`cout.setf(ios::scientific);`
		`double value = 1024.165;`
		`cout << value << endl;`
			`1.024165e+003`

`fixed`	Output floating-point numbers in fixed form (default)	`double value = 1024.165;` `cout << value << endl;`	`1024.165`
`right`	Right-aligned (default). You must first set the field width (width) fields should be deliberately greater than the length of the output line).	`cout.width(40);` `cout << "cppstudio.com" << endl;`	`__cppstudio.com`

`left`	Left alignment . First, you need to set the field width (the field width should be significantly larger). how, length output lines).	`cout.setf(ios::left);`
		`cout.width(40);`
		`cout << «cppstudio.com» << endl;`
			`cppstudio.com___`

Another way to format is using handles. A handle is a special type of object that controls input/output streams, for formatting the data transferred to the streams Information. Manipulators partially supplement the functionality for formatting input/output. However, most manipulators perform exactly the same functions as formatting flags. There are cases where it's easier to use formatting flags or functions, while other times it's more convenient to use formatting manipulators. This is why C++ provides several input/output formatting tools. Table 5 shows the main C++ formatting manipulators.

Table 5 – Some manipulators formatting V C++

Manipulator	Purpose	Example	Result
`endl`	Transition on new line at conclusion	`cout << "website:" << endl << "cppstudio.com";`	`website: cppstudio.com`
`scientific`	Printing floating - point numbers in scientific notation	`double value = 1024.165; cout << scientific << value << endl;`	`1.024165e+003`
`fixed`	Output floating point numbers in fixed form (default) .	`double value = 1024.165; cout << fixed << value << endl;`	`1024.165`
`setw(int number)`	Install width fields, Where number - number of positions, characters (default alignment is right-aligned). Manipulator with parameter.	`cout << setw(40) << cppstudio.com << endl;`	`__cppstudio.com`
`right`	Right-aligned (default). First, you need to set the field width (the field width must be greater than the length of the output line).	`cout << setw(40) << right << cppstudio.com << endl;`	`__cppstudio.com`
`left`	Left alignment. First, you need to set the field width (the field width must be greater than the length of the output line).	`cout << setw(40) << left << cppstudio.com << endl;`	`cppstudio.com __`
`setprecision(int count)`	Specifies the number of decimal places, where count - quantity signs after the decimal point	`cout << fixed << setprecision(3) << (13.5 / 2) << endl;`	`6,750`
`setfill(int symbol)`	Set the filler character. If the field width is greater than the output value, the empty spaces in the field will be filled. symbol symbol - placeholder symbol	`cout << setfill('0') << setw(4) << 15 << ends << endl;`	`0015`

When using manipulators, you must include the <iomanip> header file.

Laboratory Job No. 2
Programming algorithms linear structures

TARGET WORKS: consolidation knowledge O types data, transformations types, acquiring skills in composing and debugging linear programs in the C++ programming language

EXECUTION WORKS: in in accordance With option make up And implement programs.

Exercise I

Write a program to calculate using two formulas. Prepare test examples for the second formula using a calculator (the calculation results for both formulas must match). A list of standard C++ mathematical functions is provided in Appendix 2. Express functions missing from the language using existing ones.

1.	$z_1 = 2\sin^2(3\pi - 2\alpha)\cos^2(5\pi + 2\alpha)$;	$z_2 = \frac{1}{4} - \frac{1}{4}\sin\left(5p - 8\alpha\right)$.
2.	$z_1 = \text{Cart}\,\alpha + \text{my}\,\alpha + \text{Cart}\,3\alpha + \text{my}\,3\alpha$;	$z_2 = 2\sqrt{2}\cos\alpha \cdot \sin\left(\frac{\pi}{4} + 2\alpha\right)$.
3.	$z_1 = \frac{\sin 2\alpha + \sin 5\alpha - \sin 3\alpha}{\cos\alpha + 1 - 2\sin^2 2\alpha}$;	$z_2 = 2\sin\alpha$.
4.	$z_1 = \frac{\sin\alpha + \sin 5\alpha - \sin 3\alpha}{\cos\alpha - \cos 3\alpha + \cos 5\alpha}$;	$z_2 = tg\,3\alpha$.
5.	$z_1 = 1 - \frac{1}{4}\text{without}^2 2\alpha + \cos 2\alpha$;	$z_2 = \cos^2\alpha + \cos^4\alpha$.
6.	$z_1 = \cos\alpha + \cos 2\alpha + \cos 6\alpha + \cos 7\alpha$;	$z_2 = 4\cos\frac{\alpha}{2} \cdot \cos\frac{5}{2}\alpha \cdot \cos 4\alpha$.
7.	$z_1 = \cos^2\left(\frac{3}{8}p - \frac{\alpha}{4}\right) - \cos^2\left(\frac{11}{8}p + \frac{\alpha}{4}\right)$;	$z_2 = \frac{\sqrt{2}}{2}\sin\frac{\alpha}{2}$.
8.	$z_1 = \cos^4 x + \sin^2 y + \frac{1}{4}\sin^2 2x - 1$;	$z_2 = \sin(y + x) \cdot \sin(y - x)$.
9.	$z_1 = (\cos\alpha - \cos\beta)^2 - (\sin\alpha - \sin\beta)^2$;	$z_2 = -4\sin^2\frac{\alpha - \beta}{2} \cdot \cos(\alpha + \beta)$.

10. $z_1 = \dfrac{\sin\left(\dfrac{\pi}{2} + 3\alpha\right)}{1 - \sin(3\alpha - \pi)}$;	$z_2 = ctg\left(\dfrac{5}{4}\pi + \dfrac{3}{2}\alpha\right)$.
11. $z_1 = \dfrac{1 - 2\sin^2\alpha}{1 + \text{Chinese } 2\alpha}$;	$z_2 = \dfrac{1 - tg\alpha}{1 + tg\,\alpha}$.
12. $z_1 = \dfrac{\sin 4\alpha}{1 + \cos 4\alpha} \cdot \dfrac{\cos 2\alpha}{1 + \cos 2\alpha}$;	$z_2 = ctg\left(\dfrac{3}{2}\pi - \alpha\right)$.
13. $z_1 = \dfrac{\sin\alpha + \cos(2\beta - \alpha)}{\cos\alpha - \sin(2\beta - \alpha)}$;	$z_2 = \dfrac{1 + \sin 2\beta}{\cos 2\beta}$.
14. $z_1 = \dfrac{\cos\alpha + \sin\alpha}{\cos\alpha - \sin\alpha}$;	$z_2 = tg\,2\alpha + \sec 2\alpha.\left(\sec x = \dfrac{1}{\cos x}\right)$.
15. $z_1 = \dfrac{\sqrt{2b + 2\sqrt{b^2 - 4}}}{\sqrt{b^2 - 4} + b + 2}$;	$z_2 = \dfrac{1}{\sqrt{b + 2}}$.
16. $z_1 = \dfrac{x^2 + 2x - 3 + (x + 1)\sqrt{x^2 - 9}}{x^2 - 2x - 3 + (x - 1)\sqrt{x^2 - 9}}$;	$z_2 = \sqrt{\dfrac{x + 3}{x - 3}}$.
17. $z_1 = \dfrac{\sqrt{(3m + 2)^2 - 24m}}{3\sqrt{m} - \dfrac{2}{\sqrt{m}}}$;	$z_2 = \sqrt{m}.(m > 0)$
18. $z_1 = -2 - \left(\dfrac{a + 2}{\sqrt{2}\,a} - \dfrac{a}{\sqrt{2}\,a + 2} + \dfrac{2}{a\,2\sqrt{a}}\right) \cdot \dfrac{a - 2}{a + 2}$;	$z_2 = \dfrac{1}{\sqrt{a}\,2\sqrt{}}.(a > 0)$
19. $z_1 = \left(\dfrac{1 + a + a^2}{2a + a^2} + 2 - \dfrac{1 - a + a^2}{2a - a^2}\right)^{-1}$	$(5 - 2a^2\zeta 2 = \dfrac{4 - a^2}{2}$.
20. $z_1 = \dfrac{(m - 1)\sqrt{m} - (n - 1)\sqrt{n}}{\sqrt{m^3 n + nm + m^2} - m}$;	$z_2 = \dfrac{\sqrt{m} - \sqrt{n}}{m}$.
21. $of_1 = (1 + tg\,\alpha)(1 + ctg\,\alpha) - 2$;	$z_2 = \dfrac{1}{\sin\alpha \text{ something } \alpha}$.

22. $z_1 = 1 + ctg^2\alpha + \frac{1}{\cos^2\alpha}$;	$z_2 = \frac{1}{\sin^2\alpha \cos^2\alpha}$.
23. $z_1 = 1 - \frac{1}{1 + ctg^2\alpha}$;	$z_2 = \frac{1}{1 + tg^2\alpha}$.
24. $z_1 = \frac{1 - 2\cos^2\alpha}{\text{Chinese}\,\alpha \cos\alpha} + ctg\,\alpha$;	$z_2 = tg\,\alpha$.
25. $z_1 = \text{without}^4\alpha - 1 + 2\,\text{without}^2\alpha \cos^2\alpha$;	$z_2 = -\cos^4\alpha$.

Exercise II

1. Compose program calculations volume cone By given diameter and forming.
2. Given a side of an equilateral triangle, write a program to find the area of this triangle and the radius of the circumscribed circle.
3. Known radii two concentric circles. Compose a program to find the area of the ring formed by these circles.
4. Create a program to find the sum of n terms of an arithmetic progression for which the first term, the difference, and the number *n are known* .
5. Calculate the volume of a prism whose lateral faces are squares and whose base is an equilateral triangle inscribed in a circle of radius r .
6. Calculate square rectangle, inscribed V circle radius r , if the ratio of its sides is equal to R .
7. Given two sides triangle And corner between them. Define third side and area.
8. Calculate percent material, departed V waste, If from Cuba With on edge A was chiseled ball radius r ($r < A$).
9. Calculate the area of a ring whose width is a and the ratio of the radii of the circles is b .
10. Calculate perimeter And square rectangular triangle, described around a circle of radius r if the hypotenuse of the triangle is equal *to c* .
11. Calculate the diameter of a pipe whose capacity allows it to replace two pipes with diameters d_1 and d_2 .
12. Calculate heights triangle with sides a , b And *With* .
13. Calculate the area of a regular n -gon circumscribed about a circle of radius r .
14. Height cone equal h , A radius grounds r . Calculate volume a ball inscribed in a cone.

15. Calculate the mass of a lead pipe whose length is b m (the density of lead is 11.4 g/cm ^{3}), the wall thickness is a mm, and the internal diameter of the pipe is d mm.
16. Given an equilateral triangle, calculate the side, height, and area of this triangle if the radius of the inscribed circle is r .
17. A steel shaft, b mm long and *d mm in diameter, is turned on a lathe. The diameter decreases by s mm* during turning . Calculate the decrease in mass (the density of steel is 7.4 g/cm3) .
18. Calculate volume prisms, lateral edges which - squares. The base of the prism is an equilateral triangle, defined by the length of its side.
19. IN cone With radius grounds r And height h inscribed cylinder, the radius of the base of which is equal to a . Calculate the volume of the cylinder.
20. Given two sides of a triangle d and c and the angle between them ⟨ . Find the radii circles: inscribed in a triangle and circumscribed around it.
21. Given the length of a cube's edge , find the cube's volume and its lateral surface area .
22. Triangle asked three parties. Calculate his heights.
23. Determine the perimeter of a regular n -gon circumscribed around a circle of radius R.
24. V_1 liters of water at temperature T_1 and V_2 liters of water at temperature T_2 are mixed . Find the volume and temperature of the resulting mixture.
25. Known length circles. Find square circle, limited this circle.
26. Given hypotenuse And leg rectangular triangle. Find second leg and the radius of the inscribed circle.
27. Find square rings, interior radius whom equal R_1 , A external - R_2 ($R_2 > R_1$).
28. Determine the time after which two bodies moving towards each other with uniform acceleration will meet if their initial velocities, acceleration, and initial distance between them are known.
29. Calculate distance between two dots With coordinates (x_1 , y_1) And (x_2 , y_2).
30. Given legs rectangular triangle. Find his hypotenuse And square.

Exercise III

1. Radon baths used for treatment contain 1.8 x 10 ^{6} radon atoms in 1.0 dm $^{3\ of\ water}$. How many water molecules are there per radon atom in a therapeutic bath?
2. How many mercury atoms are contained in 1.0 m3 of air ^{in} a room contaminated with mercury at a temperature of 20 ° C, if the saturated vapor pressure of mercury at this temperature is 133 mPa?

3. What is the length of the edge of a cube containing 1.0 x 10^{6} molecules of an ideal gas under standard conditions?
4. At what temperature do helium molecules have the same average speed as oxygen molecules at 23 ° C?
5. Find the average kinetic energy of translational motion hydrogen molecules, if at a pressure of 0.5 atm. their concentration is equal to $1.5 \cdot 10^{9}$ m^{-3}.
6. The gas is heated in an open vessel at normal atmospheric pressure from 27 ° C to 327 ° C. What increase does the number of molecules per unit volume of gas receive?
7. A vessel with a volume of 1 dm^{3} contains a certain gas at a temperature of 17 ° C. Find the increment gas pressure, if due to gas leakage from 10^{21} molecules will come out of it.
8. A vessel with a volume of 3.0 dm^{3} contains helium with a mass of 4.0 mg, nitrogen with a mass of 70 mg and 5.0 x 10^{21} hydrogen molecules. What is the pressure of the mixture if its temperature 27 ° C?
9. Determine the average kinetic energy of rotational motion of hydrogen molecules contained in 1.0 mol at 18 ° C.
10. At what temperature do oxygen molecules have the same average speed as hydrogen molecules at 25 ° C?
11. How many molecules contains at normal conditions V 1m^{3} air?
12. A vessel with a volume of 2.0 dm3 ^{contains} gas under a pressure of 0.50 MPa. What is the average kinetic energy of the translational motion of gas molecules?
13. Oxygen weighing 12 g is at a temperature of 700 ° C, with 40% of the molecules dissociated on atoms. What equal average kinetic energy of thermal motion of particles? The vibrational degrees of freedom of oxygen molecules are not get excited.
14. The volume of a room is 50 m3 · The air temperature in winter is 0 ° C, and in summer – 40 ° C. What is the difference in the mass of air filling the room in winter and summer? Take μ = 29 kg/mol.
15. The density of air under normal conditions is 1.3 g/l. What is the density of air at a temperature of 100 ° C and a pressure of 4.0 $10^{5?}$ N/m^{2} ?
16. The air temperature in a 10 m3 cylinder ^{at} a pressure of 700 mmHg was 15 ° C. After heating the air to 20 ° C, some of the air was displaced. Find the mass of the displaced air.
17. How many particles (atoms and molecules) are in nitrogen with a mass of 1.0 g if the degree of dissociation of nitrogen is 7.0%.
18. What pressure on the walls of a vessel does oxygen produce if the mean square velocity of its molecules is 400 m/s and the concentration of molecules is $2.7 \cdot 10^{19}$ m^{-3} ?

19. Find the average kinetic energy of the translational motion of helium molecules if at a pressure of 0.5 atm their concentration is $1.5 \cdot 10^{9}$ m ^{3}.
20. Determine the temperature of the gas if the average kinetic energy of the translational motion of its molecules is $1.6 \cdot 10^{-19}$ J.
21. What is the pressure of a gas if each cubic centimeter contains $1.0 \cdot 10^{6}$ molecules and the gas temperature is 87 ° C?
22. The gas is heated in an open vessel at normal atmospheric pressure from 22 ° C to 320 ° C. What increase does the number of molecules per unit volume of gas receive?
23. A vessel with a volume of 15.0 dm3 contains a gas under a pressure of 0.50 MPa. What is the average kinetic energy of the translational motion of the molecules? gas?
24. At what temperature do helium molecules have the same average speed as hydrogen molecules at 27 ° C?
25. The gas occupies a volume of 2 liters at a pressure of $5 \cdot 10^{5}$ N/m ^{2}. Determine the total energy of the translational motion of the gas molecules.

Tasks For independent work

1. Given two valid numbers. Find average arithmetic And the geometric mean of these numbers.
2. Given four whole numbers. Find average arithmetic these numbers and the geometric mean of these numbers.
3. A triangle is defined by the lengths of its sides. Using Heron's formula, find the area of this triangle.
4. It's coming k -ya second days. Define How many full hours H, full minutes M passed To to this moment.
5. Given two moment time some days: H_1, H_2 - watch, M_1, M_2 - minutes. Determine the interval between these moments in hours. H and minutes M.
6. By three coordinates peaks some triangle find his height.
7. Find the volume and lateral surface area of a cone of radius r, height h and forming l.
8. Calculate lateral surface pyramids (pyramid correct, truncated), if p and $p1$ are known - the semiperimeters of the lower and upper bases and k is the apothem.
9. Find the volume and total surface area of a hollow sphere if r_1 and r_2 are known - the radii of the outer and inner spherical surfaces.
10. Calculate the surface area and volume of a truncated cone of height h, radii r_1 and r_2, and generator l.
11. A cone with an angle a at the apex in the axial section is inscribed in a sphere of radius r cone. Determine the volume and total surface area of the cone.

12. Calculate the volume of a cylinder inscribed in a regular hexagonal prism, each edge of which is equal *to a* .
13. Find square isosceles trapezoids With grounds *A* And *IN* And angle α with a larger base *A* .
14. A triangle is defined by the magnitudes of its angles and the radius of the circumscribed circle. Find the sides of the triangle.
15. Find the volume of a regular triangular pyramid if the sides of its base are equal to *a* and the plane angle at its apex is equal to α .
16. Calculate the radius of a sphere inscribed in a regular quadrangular pyramid if the sides of the base of the pyramid are equal to *a* and the dihedral angle at the base is equal to α .
17. Calculate the area of a circle and an isosceles trapezoid described around it if the perimeter of the trapezoid is equal to *p* and the angle at the lower base is equal to *a* .
18. Calculate the volume of a prism whose lateral faces are squares. The base of the prism is an equilateral triangle, and the radius of the inscribed circle is known to be *r* .

Control questions

1. What such program on C++?
2. Which program is linear?
3. What is it? structure programs on C++?
4. What such identifier?
5. By what rules are being created identifiers V C++?
6. What such type of quantity?
7. Which types quantities are used V language C++?
8. Which types quantities relate To simple types?
9. IN what range can change variables type **int** and **float** ?
10. How are described variables V C++? (Bring example.)
11. What such operator brackets? What their appointment?
12. How is being carried out team assignment?
13. Which instructions intended For conclusion on screen messages?
14. Which instructions intended For input data?
15. What such heading file? Bring example heading file.
16. What is it like? appointment headings files?
17. How save program? (Orally)
18. How to carry out compilation text programs? (Orally)
19. How to carry out launch programs? (Orally)
20. How count ready program With disk? (Orally)

Example execution laboratory work

Task I. Write program For calculation By two formulas. Absent In the language library, express functions through existing ones.

$$z_1 = tg\left(x\right) - ctg\frac{7\pi}{6},$$

$$z_2 = news\frac{7\pi}{8} - \sqrt{3}\cdot news\frac{\pi}{8}\cdot tg\left(x\right)$$

Решение

1. Mathematical model

The C++ math.h library does not have the function $ctg\left(x\right)$, so we will express it through the existing function $tg\left(x\right)$, using the well-known relation:

$$ctg\left(x\right) = \frac{1}{tg\left(x\right)}.$$

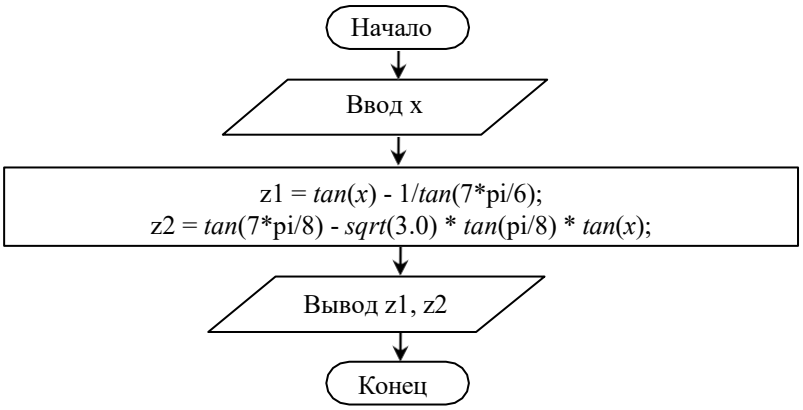
Recording expressions on language C++ will take the form:

z1 = *tan* (*x*)- 1/ *tan* (7*pi/6); z2 = *tan* (7*pi/8)- *sqrt* (3.0)* *tan* (pi/8)* *tan* (*x*).

Arguments: *x - material type.*

Results : z_1, z_2 *– material type.*

2. Algorithm (block - scheme)



3. Program

```
                              //connection headings files
#include <iostream>     //for organizing input/output in C++
#include <conio.h> //to organize console input/output #include
<math.h>                //For work With mathematical functions

usingnamespace std; // description of the namespace

std int main()          // title main functions
{     //body functions
      constdouble pi = 3.14159265359;     // description
      constants double z1, z2, x;   // variable descriptions
      cout<< " Enter X = " ;              // conclusion messages on screen
      cin>>x;                             // input variable X
      //calculate values variables z1, z2
      z1 = tan(x)-1/tan(7*pi/6); //assignment variable z1 values expressions
```

```
      z2 = tan(7*pi/8)- sqrt(3.0)*tan(pi/8)*tan(x);
      cout<< " z1 = " <<z1<< " z2 = " <<z2; // conclusion data on screen
      _getch();   //function expectations pressing keys from conio.h
      return 0;   //main function returns integer value 0
}
```

4. Result work programs

```
Enter X =  4
z1 =  -0.57
z2 =  - 1.24
```

Exercise II. Write program For calculations lengths circles And area of a circle of a given radius.

Solution

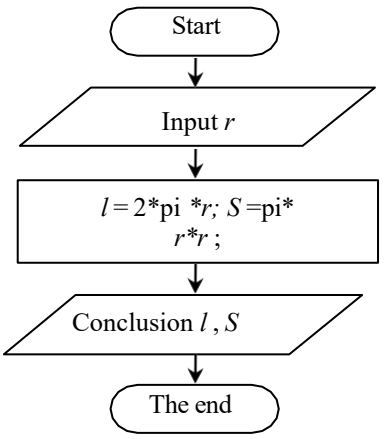
1. Mathematical model $l = 2 \cdot \pi \cdot r \quad S = \pi \cdot r^2$.

,

Arguments: radius circles r , material type.

Results: length circles l And square circle S , material type.

2. Block - scheme	3. Program

	`//program findings length circles and squares`
	`// circle given radius #include`
	`<iostream> #include <conio.h>`
	
	`usingnamespace std; int main() //`
	
	`main function header`
	`{ // description variables , initial installations`
	`setlocale(LC_ALL, "Rus" );`
	`double pi = 3.14;`
	`double r, l, S;`
	`cout<< "Enter the radius: r`
	`= " ; cin>>r; // variable`
	`input`
	`//calculation By`
	`formulas l =`
	`2*pi*r;`
	`S = pi*r*r;`
	`//conclusion data`
	`cout<<endl<< "Circle length: l = " <<l;`
	`cout<<endl<< "Circle area: S = " <<S;`
	`_getch();`
	`//home function returns whole meaning 0 return 0;`
	`}`

4. Result work programs:

```
Enter the radius: r = 5
Length circles: l = 31.4
Area of a circle: S = 78.5
```

3. Operators branching

3.1. Simple And composite conditions

In C++, a condition is an expression of logical type (**bool**) that can take one of two values: "true" (**true**) or "false" (**false**). The internal form of representing the value **false** - 0 (zero). Any other value is interpreted as **true** . When converted to an integer, true has the value 1.

Six operators are used to compare the values of expressions, numeric variables, and the value of a variable and a constant:

> (more),	!= (Not equals),
< (less),	>= (more or equals),
== (equals),	<= (less or equals).

Conditions that are constructed using a single comparison operator are called **simple conditions, for example, x + y> 0** .

General view:

```
expression  comparison operator  expression
```

From simple conditions, which are expressions of logical type, it is possible to construct **complex conditions** by applying logical operators to them as operands: ! (not), && (and), || (or).

Logical operations are performed from left to right. If the value of the first operand is sufficient to determine the result of the operation, the second operand is not evaluated. Parentheses are used to change the order of operations.

So, **compound conditions** are simple conditions connected using logical operators: `(x > 5) || (a + b < 0).`

Operation &&

The logical operator **&&** (and) is used when you want two conditions to be met simultaneously. In C++, this is written like this:

```
(condition1) && (condition2)
```

In practice, this logical operation is often used to create a condition for a variable to belong to a specified range. For example, let's create a condition that will be true if and only if the variable x belongs to the range from 10 to 20: $10 < x < 20$. That is, when $x > 10$ ***and*** $x < 20$. In C++, this is written as follows: `(x> 10) && (x< 20)` .

Operation ||

The logical operation || (or) is used when one wants to formulate a condition that will be true if at least one condition is true. of two. In C++, this operation is used like this:

```
(condition1) || (condition2)
```

Operation !

The logical operator **!** (not – logical negation) is used when you need to test a condition (expression) for falsity. This operation makes a true condition false and a false condition true. The operation is written as:

```
!(condition)
```

Table 6 shows the results of applying logical operators to operands of a Boolean type. For clarity, 0 and `true are used instead of the false value.` – 1.

Table 6 - Logical operations

a	b	a && b	a || b	!a
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

When writing complex conditions, it is necessary to take into account the priority of operations and, in If necessary, use parentheses.

3.2. Composite operator

TO composite operators relate actually composite operators and blocks.

A compound statement is a sequence of statements enclosed in statement brackets. Operators, incoming V him, should be carried out V in the order in which they are written.

Format operator:

```
{
    operator1;
    operator2;
       ...
    operatorN;
}
```

A block is distinguished by the fact that it contains descriptions of certain program objects. For example:

```
{// This composite
operator n++;
S += n;
}
```

```
{// This is
a block
int  n = 0;
n ++;
S += n;
}
```

Blocks are most often used as function bodies. A compound statement is used whenever several statements need to be combined into one. This is necessary in conditional statements and loop statements, whose syntax dictates that only one statement is executable. More often

total, composite operator forms a branch of a conditional statement or a loop body in loop statements.

3.3. Conditional operation (?:)

This operation ternary, That There is has three operand.

Format operations:

```
operand_1 ? operand_2 :  operand_3
```

First operand Maybe have arithmetic type or be a pointer. It is evaluated for its equivalence to zero (an operand equal to zero is considered **false** , a non-zero operand is considered **true**). If the result If operand 1 is **true** , then the result of the conditional operation will be the value of the second operand, otherwise it will be the value of the third. operand. Either the second or the third operand is always evaluated. Their types may differ. The conditional operator is a shortened form of the **`if operator`** .

Example: Withdraw on screen minimum from two values.

```
#include <stdio.h>
int main ( )
{
int a = 10, b = 5, min;
                min = (b < a) ? b : a ;
printf(“ Smallest number : %d”, min);
return 0;
          }
```

Result work programs:

The greatest number: 5

Another example of using a conditional operation. It is required that some integer value was increased by 1 if its value did not exceed n, and otherwise took the value 1:

```
i = (i < n) ? i + 1 : 1;
```

3.4. Conditional operator if

It is intended to select one of two possible operators for execution depending on the fulfillment of a certain condition.

Format operator:

```
if (condition) operator 1; else operator 2;
```

Instructions is being carried out next way:

1. The value of the condition (expression) is calculated, which can have an arithmetic type or a pointer type.
2. If the value of the expression (condition) is not equal to zero (has the value **true**), the first statement following the condition is executed. If the value of the expression (condition) is equal to **false** , then the statements following the word **else are executed.**

Quite often, no action is required if a condition is false. A short form of the conditional operator is acceptable:

```
if( condition ) operator ;
```

If the logical expression evaluates to true, the statements following the condition are executed. If the expression evaluates to false, the statement immediately following the **`if statement is executed`** .

`else` keyword it is necessary to execute several statements, they are written as a compound statement.

Table 7 - Graphic interpretation of the conditional operator.

	Incomplete form records	Full form records
if- then	a) + condition – operator b) + condition – operator 1, opera or 2, …, operator n	**if-then-else** b) condition – operator 1 / operator 2 + condition – opera or 1 / operator 2 … operator n ; operator 1 / operator 2 … operator n

Note: Table 7 shows the flowcharts of possible variants of the "Branching" command (conditional operator) algorithm.

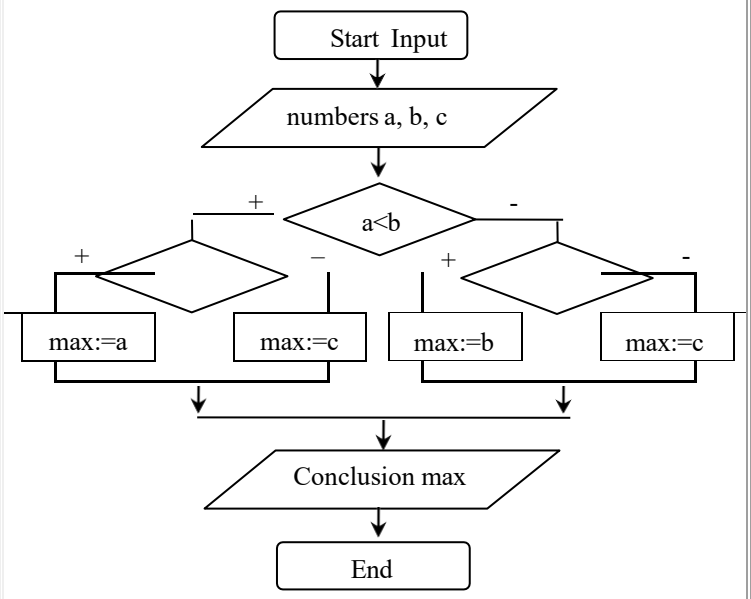
a) For simple operators;

b) for composites, when the condition is met or not met it is required to fulfill some operators : `operator 1, operator 2, …, operator n` . **In a program,** this series of operators: `operator 1, operator 2, …, operator n,` are presented as a compound operator or block, enclosed in operator brackets.

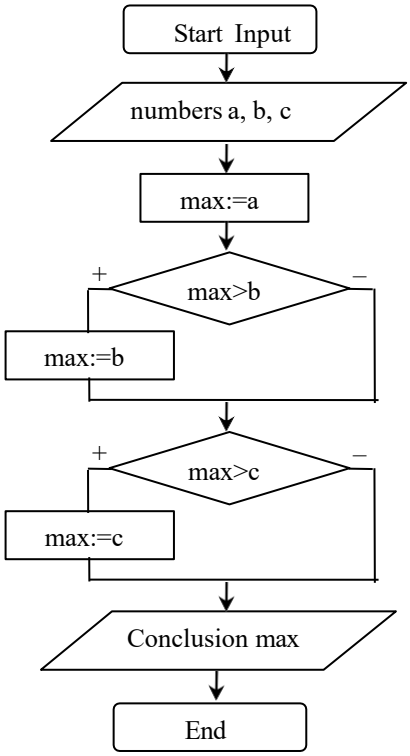
The "Branch" command structure is called nested if conditional statements are used again after the condition or **`else keyword`** . The number of nestings is arbitrary. However, the following rule applies: the **`else keyword`** always refers to the closest **`if`** .

Example. Compose program location minimum from three numbers entered from the keyboard.

1 option solutions:



2 option solutions:



Программа на C++

```
//min_of_3_numbers;
#include<conio.h>
#include<iostream>
usingnamespace std;

int main()
{
      setlocale(LC_ALL,"Rus");
      int a, b, c, max;
      cout<< "\nEnter 3 numbers through space: " ;
      cin>>a>>b>>c;
      if (a < b)
      if (a < c) max = a; // case , When a < b else
      max = c;
      else
      if (b < c) max = b; // case , When a >= b else
      max = c;
      cout<< "Minimum number = " << max;
      _getch();
      return 0;
}
```

```
//min_of_3_numbers;
#include<conio.h>
#include<iostream>
usingnamespace std;

int main()
{
    setlocale(LC_ALL, "Rus" );
    int a, b, c, max;
    cout<< "\nEnter 3 numbers separated
by spaces: " ;
    cin>>a>>b>>c;
    max = a;
    if (b < max) max = b;
    if (c < max) max = c;
    cout<< "Minimum number = " <<max;
    _getch();
    return 0;
}
```

Result work programs:

Enter 3 numbers through space:
12 5 -7
Minimum number = - 7

3.5. Operator switch

If a program needs to implement a choice between more than two options, then a multiple branching instruction (switch) can be used.

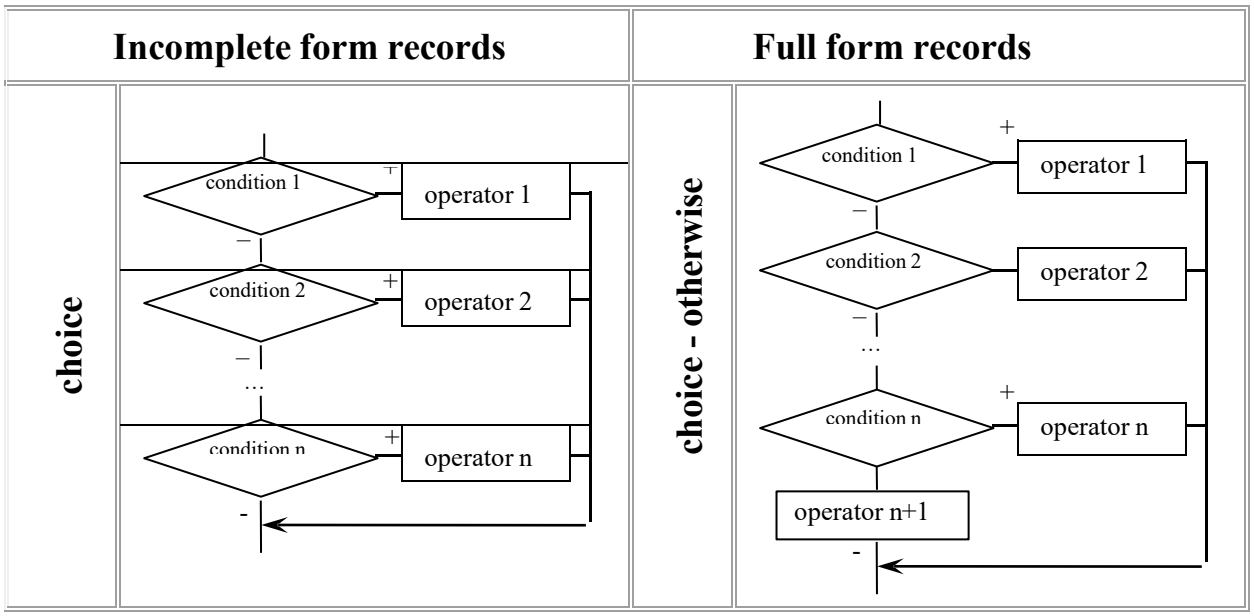
`Switch` statement A switch is used to branch the computational process into several directions. The block diagram of **`the switch operator`** is shown in Table 8.

Format operator:

```
switch ( expression )
   {
      case constant_expression_1: [operator_1];
      case constant_expression_2: [operator_2];
      …
      case constant_expression_n: [operator_n]; [
      default: operator_n + 1];
   }
```

The switch operator is a generalization of the conditional operator. Operator execution begins with the evaluation of the expression (it must be an integer), and then control is transferred to the first operator in the list marked with a constant expression whose value matches the evaluated one. *After this, if the exit from the switch is not explicitly specified, all other branches are executed sequentially.* Then control is passed to the operator following the selection operator. If the value of the expression does not match any constant expression, then operator *n* + 1 is executed.

Table 8– Operator choice (block diagram)



The expression and constant values of **`the switch statement`** must be of the same type. The type can be any integer. A list of constants can be used as a constant (e.g., **`case1: case3: case5:`** [`statement_1`];), and a compound statement can be used as an operator.

If the constants represent a range of numbers, then instead of a list, you can specify the first and last constant of the range, separating them with three dots (valid for some programming environments, for example, MinGW).

Exit from A switch is typically executed using **the break** or **return statements** . The **break statement** exits the innermost enclosing statement. its operators **switch , for ,while** and **do** . **return** statement exits the function in whose body it is written.

3.6. Operator transition goto

The unconditional jump operator is used to change the order of execution of statements.

format :

```
goto label;
```

IN body that one same functions should be present smooth one design type:

```
label: operator;
```

Operator **goto** transmits control on marked operator.

Label - This ordinary identifier, region visibility whom is the function in whose body it is defined.

The use of the unconditional jump operator is justified, as a rule, in two cases:

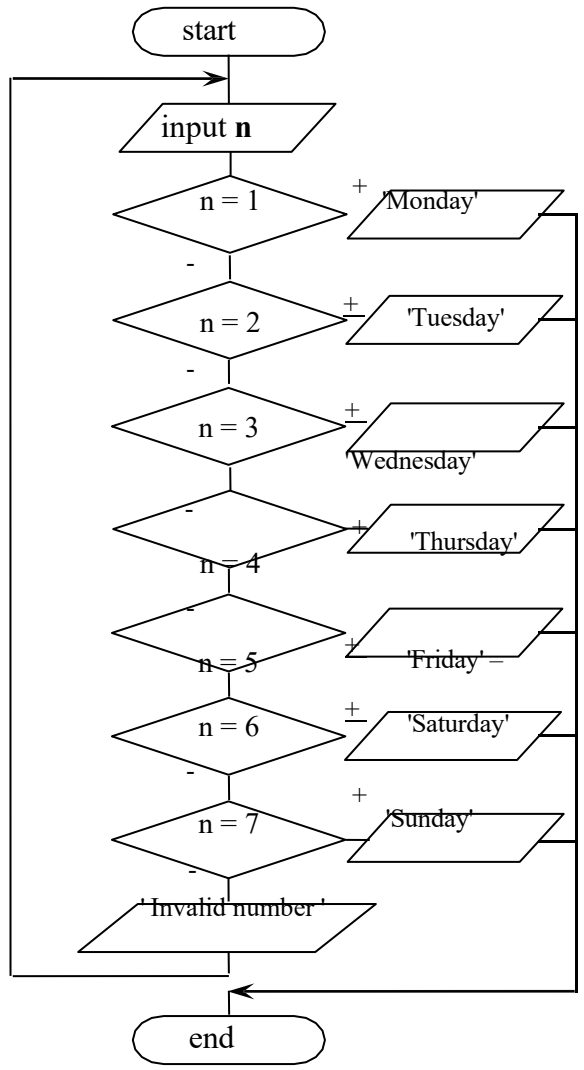
- forced exit down By text programs from multiple nested loops or switches;
- transition from several places functions V one (For example, If before exit It is always necessary to perform some actions from a function).

In all other cases, the algorithm should be brought to a structural form. form, that is, transform it so that it can be written using basic constructions.

Using an unconditional jump operator is considered bad programming style because it violates the principles of structured and modular programming, which state that all blocks that make up a program should have only one input and one output.

Example : Given number *m* ($1 \le m \le 7$). Withdraw on screen Name days weeks, which corresponds to this number.

block diagram	program

	`//switch; #include <conio.h> #include <iostream> usingnamespace std;`
	
	`int main()`
	`{`
	`  setlocale(LC_ALL, "Rus" );`
	`  int n;`
	`  mark : cout<< "\nEnter number day of the week: " ;`
	`  cin>>n;`
	`  switch (n)`
	`  {`
	`  case 1 : cout<<"\nПонедельник";break;`
	`  case 2 : cout<<"\nВторник";break;`
	`  case 3 : cout<<"\nСреда";break;`
	`  case 4 : cout<< "\n Thursday " ;`
	`  break ; case 5 : cout<< "\n Friday " ; break ; case 6 : cout<< "\n Saturday " ; break ; case 7 : cout<< "\n Sunday " ; break ; default : cout<< "\n Invalid input \n" ;`
	`goto mark;`
	`  }`
	`  _getch();`
	`  return 0;`
	`}`

Result work programs:

Enter number days weeks: 12

Incorrect input

Enter number days weeks: 2

Tuesday

Laboratory Job No. 3
Programming algorithms branching structures

PURPOSE OF THE WORK: acquisition of skills in programming computational branching processes using conditional, composite operators, operator choice (multiple branching) and the unconditional jump operator.

EXECUTION WORKS: in in accordance With option make up And implement programs.

Exercise I

Decide task With using designs branching:

1. Distance from Houses before school S km. To beginning lesson there are t *left* minutes. A student walks at speed v . Determine whether he will arrive before the bell, after the bell, or on time.
2. Given valid numbers x , y And z . Find min(x , y , z).
3. Calculate meaning functions $$f = \begin{cases} 2x\sqrt{x}, \text{ if } x > 1, \\ \dfrac{x}{5}, \text{ If } x \le 0. \end{cases}$$
4. Build V square more from three given numbers.
5. Define, There is whether among three given numbers even.
6. Withdraw on screen three given numbers V in order decrease.
7. Determine whether the point A (a_1 , a_2) belongs to the ring defined by the circles: $x^2 + y^2 = 1$ and $x^2 + y^2 = 16$.
8. Calculate meaning functions $$y = \begin{cases} \ln^3\left(x^2+1\right), \text{If } x \le -3, \\ arctg\dfrac{1}{2x+1}, \text{If } x > 3. \end{cases}$$
9. Two circles are given coordinates of the center and radii. How many points of intersection do these circles have?
10. Two vessels, one spherical with radius R 1 and the other cube-shaped with edge length A , are filled to the brim with water. Compare the amounts of water in the vessels.
11. Calculate meaning functions: $$f = \begin{cases} x\sin^2 x, \text{ if } x < 1, \\ e^x, \text{ If } X \ge 2. \end{cases}$$
12. Given three arbitrary real numbers, is it possible to construct a triangle with these side lengths?
13. Which from points A (a_1 , a_2) or B (b_1 , b_2) located further from origin ?
14. Two straight asked equations: $a_1x + b_1y + c_1 = 0$ And $a_2x + b_2y + c_2 = 0$. Determine if they intersect.

15. Will the current A (a_1 , a_2) fall into a circle of a given radius with its center at the origin ?

16. Calculate meaning functions $$y = \begin{cases} ax^2 + bx + c\ , If\ x < 1.2, \\ \frac{a}{x} + \sqrt{x^2 + 1}, If\ x = 1.2, \\ \frac{a + bx}{\sqrt{x^2 + 1}}, If\ x > 1.2, \end{cases}$$
Where a = 2.8, b = - 0.3, c = 4.

17. A rectangular hole of known dimensions is cut into a fence. Determine whether a ball of a given radius will fit through this hole.

18. Two circles are given coordinates of their center and radii. Determine whether they intersect, touch, or have no common points.

19. In the Gregorian calendar, every year divisible by 4 is a leap year, except for those divisible by 100 and not evenly divisible by 400. Determine the number of days in a year based on the year number. Thus, 1900 is not a leap year, while 2000 is.

20. Define, There is whether among four given numbers multiples five.

21. Calculate meaning functions $$y = \begin{cases} at^2 \ln t\ ,\ If\ 1 \le t \le 2, \\ 1,\ If\ t < 1, \\ e^{at} \cos t\ ,\ If\ t > 2. \end{cases}$$, Where A = -0.5, b = 2.

22. From three valid numbers a , b And c choose those, modules which more 4.

23. Given three arbitrary real numbers, is it possible to construct a triangle with these side lengths? If so, derive on the screen whether the given triangle is equilateral, isosceles or scalene.

24. Given valid numbers x And y . Find $U = \max^2 (x^2 y , xy^2)$.

25. Decide square equation.

Exercise II

Decide task With using designs choice:

1. Given a number c . Print the value this number in verbal form, taking into account its sign. Provide that -7 $\le c \le$ 7.

2. Given number m ($1 \le m \le 12$). Determine, How many days V month With number m . The year is not a leap year.

3. A natural number *n is entered from the keyboard* – the number of faces of the convex polyhedron. IN dependencies from values n withdraw on screen phrase
"Convex polyhedron has n facets", having agreed ending the word "edge" With number n. To provide, What $0 \le n \le 25$. If convex

If a polyhedron with the given number of faces does not exist, display the appropriate message.

4. Enter the number m (from 1 to 4) from the keyboard. Display the names of the months corresponding to the season with the number m (count winter) season number 1).

5. Enter a natural number n —the number of faces of a convex polyhedron—from the keyboard. Depending on the value of n , display the name of a regular convex polyhedron or report that a regular convex polyhedron with that number of faces does not exist.

   To provide, What $4 \le n \le 20$.

   Note: There are five convex regular polyhedra: tetrahedron (4 faces); hexahedron or cube (6 faces); octahedron (8 faces); dodecahedron (12 faces); icosahedron (20 faces)

6. Given a natural number n <25, display the phrase "I am n years old (year or years)" accordingly.

7. Find meaning functions: $f = \begin{cases} 2\sin^2 x, & if -1 \le x \le 1, \\ 3x^3, & If -10 < x < -1, \\ 0 & V\ the\ rest\ cases. \end{cases}$

8. Given number m ($1 \le m \le 12$). Determine, To which one time years refers to month with number m .

9. Introduced whole number $|C| < 5$. If $C > 0$, That withdraw on screen Name this C

   numbers in verbal form. Otherwise, indicate that the number is negative.

10. Enter a natural number n —the number of angles in the polygon—from the keyboard. Depending on the value of n , display the name of the polygon or indicate that a polygon with that number of angles does not exist.

    To provide, What $0 \le n \le 6$.

11. n is entered from the keyboard . Depending on the value of the remainder r when dividing n by 7, display the number n as $n = 7k + r$, where r is represented in verbal form.

12. Given a number m , determine the half-year in which month m *occurs* and the number of days in that month. Assume that $1 \le m \le 12$. The year is not a leap year.

13. Given number n . Type phrase "We successfully handed over n exams", by matching the ending of the word "exam" with the number n . Provide that $1 \le n \le 25$.

14. Determine the time of year to which the month m *belongs* and determine the number of days in this month. Assume that $1 \le m \le 12$. The year is a leap year.

15. Given a number c , print its value in words. Assume that $0 \le c \le 7$.
16. Given two numbers, D (day) and M (month), determine K , the ordinal number of the day of a leap year that has dates D and M.
17. Given three numbers D , M, and G , defining the day, month, and year, check whether they form a valid date and display the appropriate message. For example, June 31, 1990 is an invalid date. The year is not a leap year.
18. A number from 0 to 9 is entered from the keyboard. Display the name of this number in verbal form on the screen.
19. On Monday, the company is open from 9:00 AM to 4:00 PM; on Tuesday, Wednesday, Thursday, and Friday, from 8:00 AM to 7:00 PM; on Saturday, from 10:00 AM to 3:00 PM; and Sunday is a day off. Find the working hours by the day of the week you entered.
20. Find the meaning functions: $y = \begin{cases} x\sqrt{x_1 - a^2}, \ If\, x = a + 1; \\ a - \cos 2x, \ If\, 0 \le x \le a \\ \overline{x} \quad \overline{2} \\ e^x, If - a \le x < 0; \\ 1 \ V\ others\ cases\ . \end{cases}$; where $a = 4$.
21. n is entered from the keyboard . Depending on the value of the remainder r when dividing n by 5, display the number n as $n = 5\ k + r$, where r is represented in verbal form.
22. In the first year, students from four groups take a computer science exam. In the second year, group No. 1 takes an exam in the subject "Information Technology" V economy", group No. 2 - exam By discipline "Information Technology in Physics," Group 3 – exam in "Information Technology in Geography," Group 4 – exam in "Information Technology in Mathematical Research." Enter the course number and group number using the keyboard and display a message indicating which test or exam the group will take.
23. An integer *N was entered from the keyboard.* < 5. If it is positive, display its name in verbal form. Otherwise, find the square of this number.
24. Students are picking apples. The quotas are as follows: first-year students must pick at least 50 kg of apples per day, second-year students at least 60 kg per day, and third-year students at least 70 kg per day. Knowing the student's year of study can help determine whether they have met their daily quota.
25. Given a number m ($1 \le m \le 12$). Determine how many days are in the month with number m . Take into account whether it is a leap year or not.

Tasks For independent work

1. Determine *D* and *M* -date *K* -th in a row day of a leap year. Express the month verbally. *D* is the day, *M* is the month. Assume that $1 \le K \le 366$.
2. The electricity supply company charges customers according to the following tariff:
   - 7 r. for 1 kWh for first 250 kWh;
   - 17 p. per kWh if consumption exceeds 250 but does not exceed 300 kWh;
   - 20 r. for kW/h, If consumption over 300 kW/h. The

   consumer consumed *n* kW/h. Calculate the fee.
3. Schoolchildren take long jump tests. If the jump is over 2.50 meters, the score is a "5," and if it's between 2 meters and 2.5 meters, the score is a "5." Score "4"; from 1.5 m to 2 m - score "3"; if less than 1.5 m - "2". Assign a score to the student if the length of his jump is known. Note: 1 m = 100 cm.
4. When purchasing goods worth from 200 to 500 rubles, a 3% discount is provided, when purchasing goods worth from 500 to 800 - a 5% discount, when purchasing goods worth from 800 to 1000 rubles - a 7% discount, Over 1,000 rubles – 10% discount. The buyer purchased 8 rolls of wallpaper at a price of *X1* and two cans of paint at a price of *X2* . How much did he pay?
5. Students harvest tomatoes. For harvesting up to 50 kg per day, the work is paid at a rate of 30 rubles per 1 kg; for harvesting from 50 to 75 kg per day, the rate is 50 rubles per 1 kg; for harvesting from 75 to 90 kg per day – 65 rubles per 1 kg; for harvesting over 90 kg per day – 70 rubles per 1 kg plus a 120 ruble bonus. The student harvested *X* kg in a day. Calculate his earnings.
6. Elementary school students schools are renting out equipment readings. IN 1st grade speed A reading speed of up to 20 words per minute is considered low, 21 to 40 words per minute is average, 41 to 50 words per minute is above average, and more than 51 words per minute is high. In second grade, a reading speed of up to 30 words per minute is considered low, 31 to 60 words per minute is average, 61 to 70 words per minute is above average, and more than 71 words per minute is high. Use the keyboard to enter the class number and the number of words the student reads per minute. Assess the reading speed.
7. Students picking apples. First-year students have to work up to 50 kg per day. paid for from calculation 30 kopecks for 1 kg; at collection from 50 to 75 kg V day - 50 kopecks per 1 kg; when collecting from 75 to 90 kg per day - 65 kopecks per 1 kg; when collecting over 90 kg per day - 70 kopecks per 1 kg plus a 20 ruble bonus. For second-year students, when collecting up to 60 kg per day, work is paid at the rate of 35 kopecks per 1 kg; when collecting from 61 to 80 kg per day - 55 kopecks per 1 kg; when collecting from 81 to 95 kg per day – 65 kopecks per 1 kg; when collecting over 95 kg per day – 70 kopecks per 1 kg plus a 30 ruble bonus.

   Student collected *X* kg apples for day. Define his earnings.
8. An integer number *A is specified from the keyboard* > -5. For a positive single-digit number, determine whether it is even; for a positive two-digit number, determine whether it is even. multiple whether it is 3; For positive three-digit numbers

determine whether it is a multiple of 5; for a number greater than 999, determine whether it is a multiple of 10. If the number $A < 0$, then display its name in words form.

9. Schoolchildren are taking long jump tests. For fifth-grade students, the standards are as follows: if the jump is longer than 2.50 meters, the score is 5; if it's between 2 and 2.5 meters, the score is 5. score 4; from 1.5 m to 2 m - score 3; if less than 1.5 m - 2. Standards for students 6 class: if length jump more 3.0 m, That grade - 5, If from 2.5 m to 3.0 - grade 4; from 2.0 m to 2.5 m - grade 3; If less 2.0 m - grade 2. Students from other classes do not take the jump test.

   Give a grade to a student if his class number and jump length are known.

10. Schoolchildren are taking long jump tests. For second-graders, the minimum jump length must be at least 1.2 meters; for third-graders, at least 1.5 meters; for fourth-graders, at least 1.7 meters; and for fifth-graders, at least 2.0 meters.

    Knowing number class And length jump student, define, handed over whether the standard is met.

11. *N < 100* from the keyboard. If it is single-digit, display its name in verbal form. If the number is two-digit, report this and determine the remainder when dividing it by 2, 3, or 5.

12. The electricity supply company charges customers according to the following tariff:
    - ✓ 7 r. for 1 kW/h for first 250 kW/h;
    - ✓ 17 r. for kW/h, If consumption over 250, But Not exceeds 300 kW/h;
    - ✓ 20 r. for kW/h, If consumption over 300 kW/h.

    There are three categories of consumers. The Category 1 tariff accounts for 75% of the total cost, the Category 2 tariff accounts for half the total cost. The rest pay the full price.

    The consumer has consumed *n* kWh. Calculate the fee knowing his category.

13. *N* is entered from the keyboard. < 10000. Report whether it is one-digit, two-digit, three-digit, or four-digit. In each case, determine whether it is even or odd.

14. Elementary school students take a reading proficiency test. The standards are as follows: in first grade, a child should read at least 20 words per minute; in second grade, at least 50 words per minute; in third grade, at least 60 words per minute; and in fourth grade, at least 70 words per minute. By entering the student's grade number and the number of words read per minute on the keyboard, you can determine whether the reading proficiency test has been passed.

15. Given three integers defining a date: year, month, and day. Assuming the year is not a leap year, determine the date of the next day.

Control questions

1. What such composite operator?

2. What is it? full (incomplete) form teams branching (block diagram)?
3. What algorithm execution teams branching?
4. What is the execution algorithm? plural commands branching (choice)? Flowchart.
5. Which operators comparisons are used V C++?
6. What it's called simple condition? Bring examples.
7. What such composite condition? Bring examples.
8. What logical operators are allowed when creating complex conditions?
9. What result applications operator logical "AND" (&&)?
10. What result applications operator logical "OR" (||)?
11. What result applications operator logical negation (!)?
12. What general view (format) instructions "Branching"?
13. What is the algorithm for performing the conditional (ternary) operation (?:)? Give an example.
14. What general view (format) instructions "Choice"?
15. Maybe whether operator branching contain inside myself other branching?

Example execution laboratory work

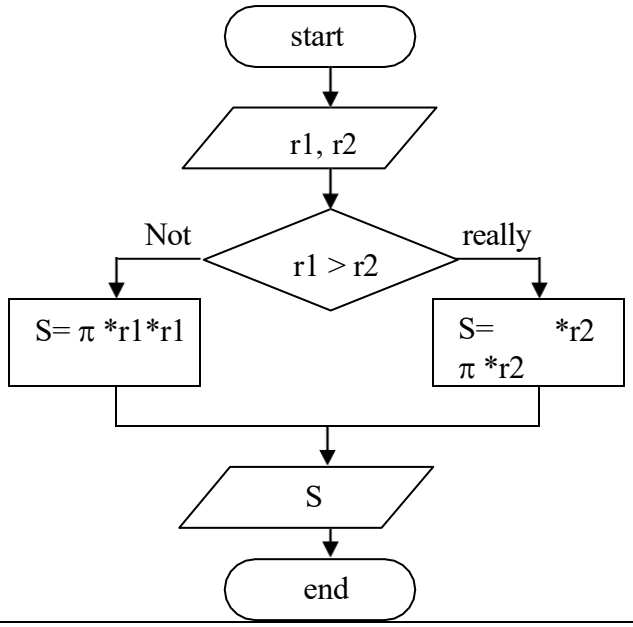
Exercise I. Two circles asked radii. Define which from of these circles will have a larger area. Display the value of the larger area on the screen.

Solution

1. Mathematical model

The greatest square will at circles With the greatest radius.

Arguments: r1, r2 material type - radii circles.
Results: S material type - meaning the greatest area.

2. Algorithm	**3. Program**
	


```
#include <iostream>
#include <conio.h>
#include <math.h>

usingnamespace std;

int main()
{
      constdouble pi = 3.14159265359;
      double r1, r2, S;
      cout<<" Введитерадиусыr1 иr2: ";
      cin>>r1>>r2;

      if (r1>r2)
            S = pi*pow(r1, 2.0);
      else
            S = pi*pow(r2, 2.0);
      cout<< " S = " <<S;
```

	_getch();
	return 0;
	}

4. Result work programs

```
Enter radii r1 And r2: 6  3 S
= 113.04
```

Exercise II. Dan number months *m* . Withdraw on screen Name this month.

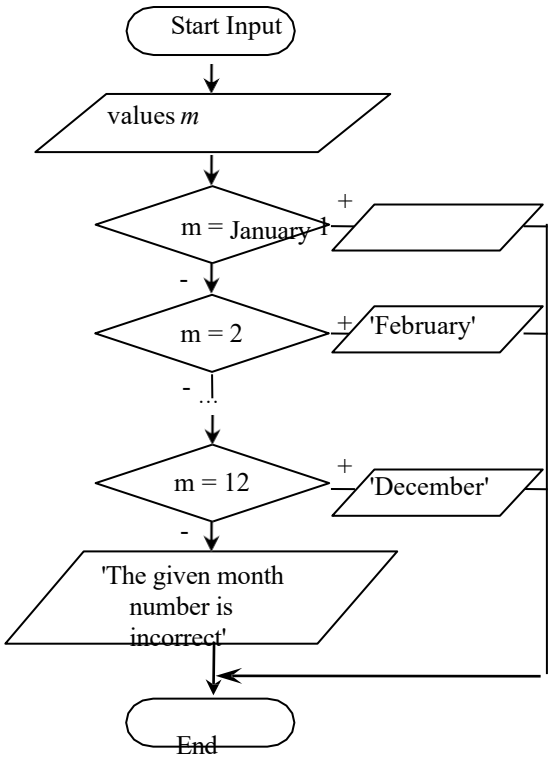
Solution

1. Mathematical model

For solutions tasks necessary use instructions choice.

Arguments: ***m*** integer type - number months.
Results: text message.

2. Algorithm



3. Program

```
#include <iostream>
#include <conio.h>

usingnamespace std;

int main()
{
    int a;

    cout<< "Enter number months: " <<endl;
    cin>>a;

    switch (a)
    {
    case 1: cout<<"Январь"<<endl; break;
    case 2: cout<<"Февраль"<<endl; break;
    case 3: cout<<" Март "<<endl; break;
    case 4: cout<<"Апрель"<<endl; break;
    case 5: cout<<" Май"<<endl; break;
    case 6: cout<<" Июнь"<<endl; break;
    case 7: cout<<" Июль"<<endl; break;
    case 8: cout<<" Август"<<endl; break;
    case 9: cout<<"Сентябрь"<<endl;break;
    case 10 : cout<<"Октябрь"<<endl; break;
    case 11 : cout<<"Ноябрь"<<endl; break;
    case 12 : cout<<"Декабрь"<<endl; break;
    default : cout<<" Ошибкаввода "<<endl;
    }
    _getch();
    return 0;
}
```

4. Result work programs

```
Enter number months: May
5
```

4. Cycles

A loop is a type of construct used to perform calculations that are repeated many times.

The block for which the loop is organized is called **the loop body** . The remaining statements control the process of repeating the calculations: these include initial settings, checking the loop continuation condition, and modifying the loop parameter (Fig. 1). One pass through the loop is called **an iteration.**

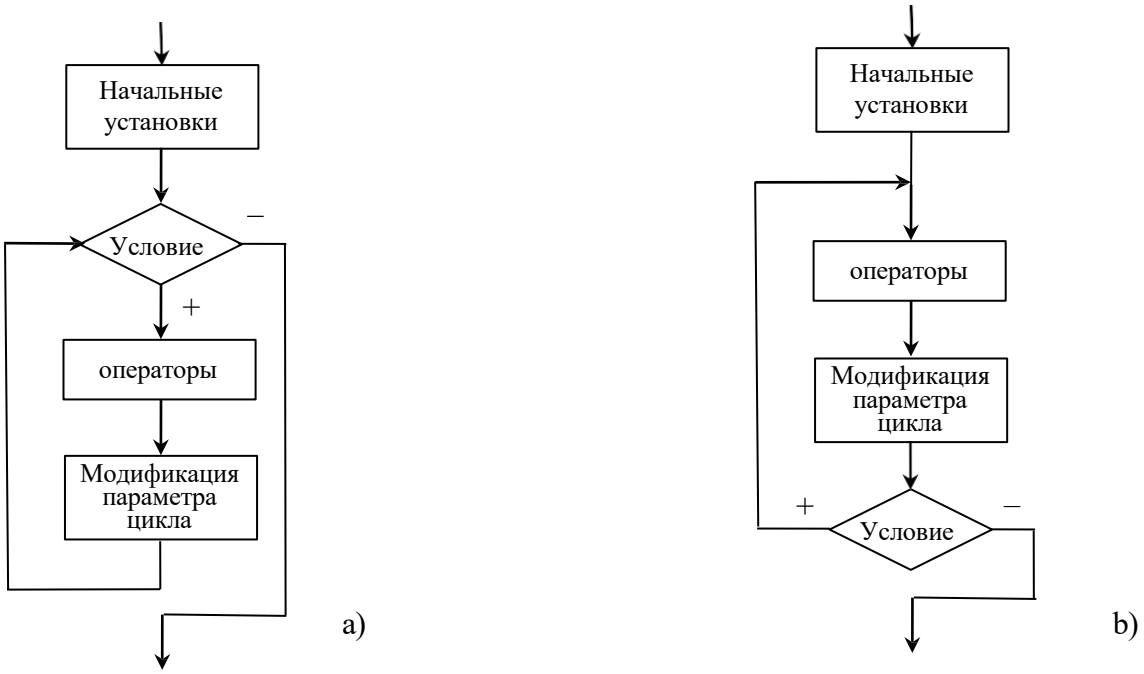


Figure 4.1 - Structural schemes execution cycle

Initial settings are used to set the values of the variables used in the loop before entering it.

The loop continuation condition is checked at each iteration either before the loop body (then we speak of a loop with a precondition, Fig. 4.1a) or after the loop body (a loop with a postcondition, Fig. 4.1b). The difference between them is that the loop body with a postcondition is always executed at least once, after which a check is made to see if it needs to be executed again. Checking the necessity execution of the cycle With precondition done before the body cycle, That's why Maybe, that it will never be fulfilled.

A loop parameter is a variable used to test the loop condition and is forced to change at each iteration, typically by the same amount. If the loop parameter is an integer, it is called the loop counter.

Three operators are provided for implementing cyclic computations: a loop with a precondition, a loop with a postcondition, and a loop with a parameter. The first two operators are used when the number of loop iterations is unknown. If the number of loop iterations can be determined before the loop begins, the loop with a parameter is used.

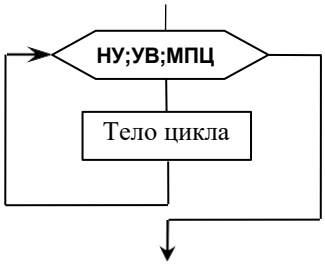
4.1. Operator cycle With parameter (for)

A loop with a parameter (a loop with a counter, an arithmetic loop, or a "for" loop) is a loop with a predetermined number of repetitions. It is suitable only for programming cyclical fragments in which the initial and final values of the loop's repetition counter are known before the loop is executed.

Format operator:

```
for (initialization; expression; modifications) operator;
```

Graphical interpretation of the **for loop operator**



WELL - initial installations (initialization)
UV– conditional expression
MPC modification parameter cycle

Initialization is used to declare and assign initial values to the variables used in the loop. Multiple statements, separated by commas, can be written in this section.

For example, So:

```
for ( int i = 0, j = 2; …
```

or

```
int k, m;
for (k = 1, m = 0 …
```

Region actions variables, announced V parts initialization cycle, is a cycle. Initialization is performed once at the beginning of the loop execution.

The expression defines the loop execution condition: if its result, cast to type **bool** , is **true** , the loop is executed. A loop with a parameter is implemented as a loop with a precondition.

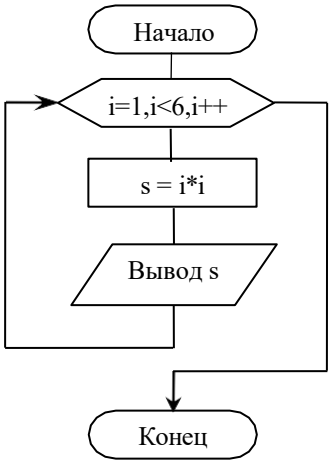
Modifications are performed after each loop iteration and are typically used to change the loop parameters. Multiple statements can be written in the modification part, separated by commas. A simple or compound *statement* represents the loop body. Any part of the **for statement** can be omitted (but the semicolons must remain in place). places!).

The value of the counter variable can be used in the loop body; Changing its value is also allowed, but is considered bad programming style.

Note: A loop with a parameter does not cause a loop, since it assumes a finite number of repetitions.

For all loop operators, exit from the loop is carried out either as a result of the natural termination of the loop operator, or using transition operators (goto – unconditional transition, continue – transition to the next iteration) and exit (break – exit from the loop).

Example: Withdraw on screen table squares first five numbers.




```
//numbers_and_them_squares
#include<conio.h>
#include<iostream>
usingnamespace std;

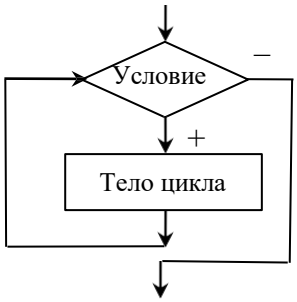
int main(){
      setlocale(LC_ALL,"Rus");
      cout<<"\nЧисла и их квадраты:\n ";
      for (int i=1;i<6;i++)
      {
            int s=i*i; cout<< "\n"
            <<i<< "^2 = " <<s;
      }
      _getch();
      return 0;
}
```

4.2. *Operator cycle With precondition* (while)

Format operator:

```
while ( expression ) operator ;
```

Graphic interpretation operator:



while statement allows you to perform the same actions repeatedly depending on the value of an expression (condition). The expression must be arithmetic or convertible to arithmetic. The expression is evaluated before each loop iteration.

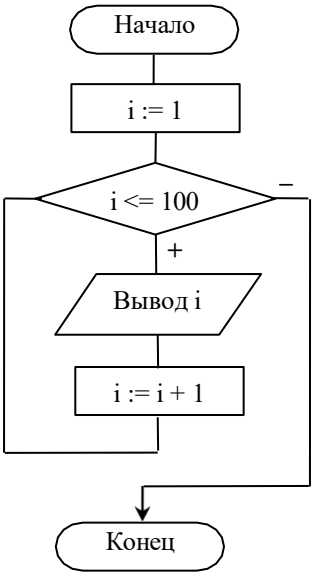
Job operator

false upon entry , the loop is not entered and control is transferred to the statement immediately following the loop statement. If the condition evaluates to true (not false), the loop is entered and the loop body, represented by a simple or compound statement, is executed once. Once the end of the loop is reached, control is transferred back to its header, where the expression is evaluated again. If the expression still evaluates to **true ,** the loop body is executed again, and so on until the condition evaluates to **false .** If it will always be equals **true** , then the cycle will endless, i.e. a loop will occur.

Notes:

– in order for the loop body to be executed at least once, the expression (condition) must be true before the loop body is executed;
– In order for the loop to complete, the values of the variables included in the condition must change in the loop body.

Example: Withdraw on screen numbers from 1 to 100.




```
//Schet_ot_1_do_100;
#include<conio.h>
#include<iostream>
usingnamespace std;

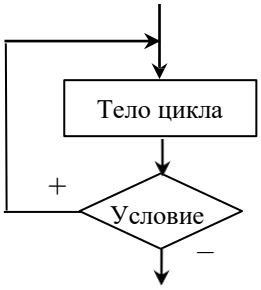
int main(){
    setlocale(LC_ALL,"Rus");
    cout<<"\nЧисла от 1 до 100:\n ";
    int i=1; //initial meaning parameter while loop
    (i<=100) // we're sorting through first 100
    numbers
    {
       cout<< " " <<i<< "  " ; // conclusion on screen
       values i i++; // Increment the loop parameter by 1
    }
    _getch();
    return 0;
}
```

4.3. *Operator cycle with a postcondition* (dowhile)

Format operator:

```
do
operator;
while ( expression );
```

Graphic interpretation operator



Job operator

First, the simple or compound statement that makes up the loop body is executed, and then the expression (condition) is evaluated. If it is true (not **`false`**), the loop body is executed again. The loop terminates when the expression evaluates to **`false`** or when any control transfer statement is executed within the loop body. The expression type must be arithmetic or convertible to arithmetic.

Notes:

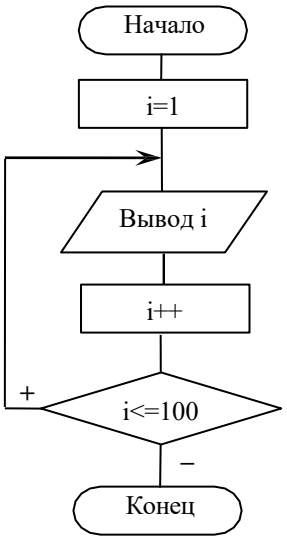
– the sequence of statements between **`do`** and **`while`** will always be executed at least once;

– In order for the loop to complete, the values of the variables included in the condition must change in the loop **body** .

The difference operator cycle **`while`**
from operator cycle **`do...while`**

IN operator **`while`** body cycle Maybe Not be carried out no once, If the logical expression will immediately turn out to be false.

Example: Withdraw on screen numbers from 1 to 100.



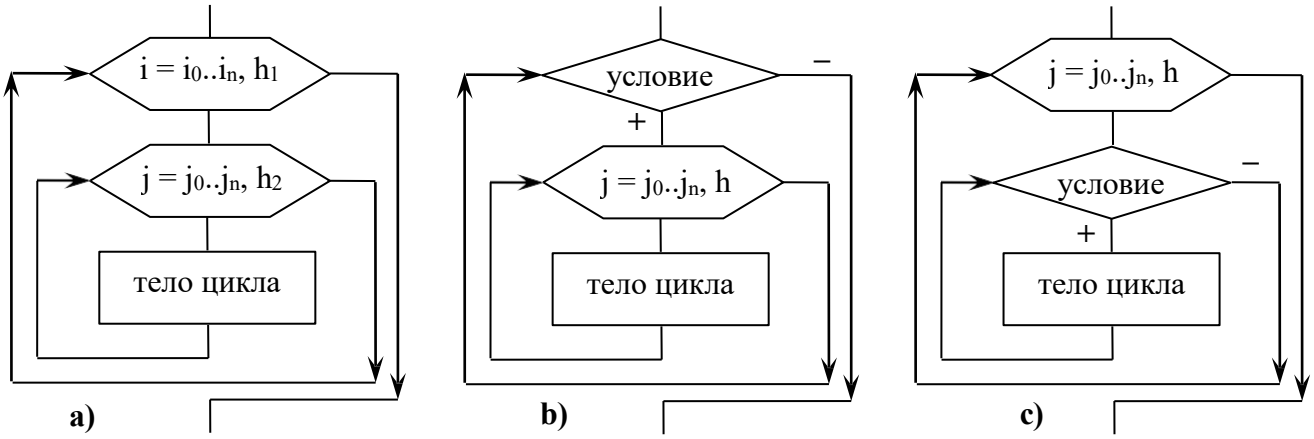

```
//Chet_ot_1_do_100;
#include <conio.h> #include
<iostream> usingnamespace
std;

int main(){ setlocale(LC_ALL, "Rus"
   ); cout<< "\n Numbers from 1 to
   100:\n " ;
   int i=1; //initial meaning parameter do { loop
      cout<< " " <<i<< "  " ; //conclusion on screen
      values i i++; // Increment the loop parameter
      by 1
   } while (i<=100) //we're iterating first 100 numbers
   _getch();
   return 0;

}
```

4.4. Nested cycles

The loop body can contain any operator, including the operator A loop structure containing a nested loop is called *a multiple loop* . The number of nested loops can be arbitrary. If a loop contains one nested loop, it is called *a double loop* (Figure 4.2).



Drawing 4.2 - Block diagrams of some from possible options implementation double cycle

A loop that contains a nested loop is called *an outer loop* . In a double loop, nested cycle is called *internal.* On Figure 4.2.a shows a flowchart of an algorithm in which a loop operator with a parameter (**for**) is used to implement both the outer and inner loops. This type of double loop is used when working with two-dimensional arrays (tables). In Figure 4.2.b, a loop operator with a precondition (**while**) is used to implement the outer loop, and a loop operator with a parameter (**for**) is used to implement the inner loop. In Figure 4.2.c, the outer loop is represented by a loop operator with a parameter (**for**), and the inner loop is represented by a loop operator with a precondition (**while**).

Internal variable cycle Always is changing faster, than external. This means that for each value of the outer loop variable, all values of the inner variable change.

The outer and inner loops can use any type of C++ loop statements (**`while`**, **`dowhile`**, **`for`**) (Figure 4.2).

Example: Given a number of the form $\overline{ab}$, where $a = 1..2$, $b = 0..9$. Display all numbers $\overline{ab}$ on the screen .

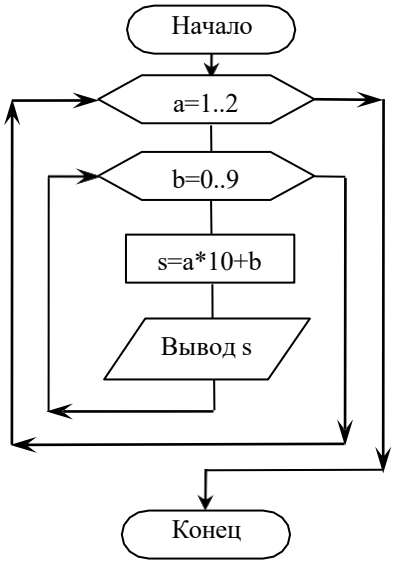
1. **Mathematical model**

Let's write it out All these numbers V row:

10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29

First, the units are incremented from 0 to 9, while the tens digit remains unchanged. Only when the units digit is complete are the tens digits incremented by 1. After this, the units digits are incremented again, from 0 to 9. In variable a – will be tens, in b – units.

2. Block diagram



3. Program

```
#include <iostream> #include
<conio.h> #include <math.h>

usingnamespace std;

int main()
{
	int s;

	for(int a=1;a<=2;a++)
		for(int b=1;b<=9;b++)
		{
			s = a*10+b;
			cout<<" "<<s;
		}

	_getch();
	return 0;
}
```

Результат работы программы

10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29

Laboratory Job No. 4
Programming algorithms cyclical structures

TARGET WORKS: acquisition skills programming computational repetitive processes using cyclic operators.

EXECUTION WORKS: V in accordance With option make up And implement programs.

Exercise I

Using operator cycle With parameter, decide proposed task:

1. Calculate the sum and product of the first n natural numbers. Allow *n to be entered* from the keyboard.
2. Calculate $S = \sum_{n=1}^{k} (-1)^{n} n^{2} \quad k \geq 10.$,
3. Calculate amount first n members rows With general member $a_k = \frac{k+1}{\sqrt{k^2+2}}, k = \underline{1,n}.$
4. Determine the average height of girls and boys in a class. There are n students in the class ($n \geq 15$).
5. Entering 4 grades for each student in a group in a loop, calculate the number of students who do not have grades 2 and 3. There are *n* students in the group. Provide for keyboard input of *n* .
6. By entering four grades earned by students during a session in a cycle, determine the average group score for all exams. There are *n* students in the group. Allow for keyboard input *of n* .
7. Given *n* triples of numbers a, b, c, enter them one by one from the keyboard in ascending order and interpret them as the lengths of the sides of a triangle. Determine how many triples can be used to construct a triangle.
8. The computer receives the results of the 200m swimming competition, in which *n* athletes ($n > 10$) are participating, one by one. The display screen shows the best result after entering the result of the next athlete.
9. *n* points ($n \geq 10$) are entered into the computer one by one . Determine how many of them fall within a circle of radius *r* centered at point (*a* , *b*).
10. First-grade students are assigned a glass of milk (200 ml) if they weigh less than 30 kg. Determine how many liters of milk are needed daily for a class of *n* students. After weighing, each student's weight is entered into the computer.
11. The list number and height of 10th-grade students are entered into the computer. Display the list numbers of those students whose height is over 170 cm.

12. *N athletes* compete in a 100-meter sprint . By entering their queue numbers and results into a computer, determine how many of them met the standard and print their numbers (the standard is 13.2 seconds).
13. *n points* are entered into a computer one by one . Determine how many of them fall within a ring with an inner radius of r_1 , an outer radius of r_2 , and a center at the origin .
14. *of n points* are entered into the computer one by one . Determine how many of them belong to the figure bounded by the x-axis and the arc of the sine wave constructed for an argument with values from 0 to π .
15. A circle of radius *r* centered at the origin is given. By sequentially entering the coordinates of *n* points that are the centers of other circles of the same radius into a computer, determine how many of these circles intersect the given circle.
16. Find all three-digit natural numbers whose sum of digits is equal to their product.
17. Find the sum of positive integers greater than 20, less than 100 and multiples of 3.
18. Calculate amount $With = 1 + \frac{1}{3^3} + \ldots + \frac{1}{n} + \frac{1}{2^3}$, where $n \geq 10$.
19. By entering the grades received by students during the exam period in a cycle of 4, determine the number of failing students in the group. There are *n* students in the group. Allow for keyboard input *of n* .
20. A cylinder with a volume of 1 cubic unit has a height of *h* . Determine the radius of the base for values *of h* from 1 to 6 in 0.5 unit increments.
21. Withdraw on seal subsequence With common member $a_k = \sin\left(k^2 + b \right)$, Where k = 0, 1, ..., K . Provide input of K and b from the keyboard. Calculate the number of elements satisfying the condition: $0 \leq a_k \leq 0.5$.
22. *N points* are entered into the computer one by one . Determine how many of them belong to the figure bounded by the ordinate axis and the right semicircle $x^2 + y^2 = r^2$.
23. *of n points* are entered into a computer one by one . Determine how many of them belong to the figure bounded by the line $|x| + |y| = 1$.
24. Create a program to calculate the values of a function $p(x) = \frac{1}{16}\left(231x^6 - 315x^4 + 105x^2 - 5 \right)$ at change x 0 to 1 With step by step 0.1.
25. Calculate meaning expressions $\frac{(x-2)(x-4)(x-6)\ldots(x-64)}{(x-1)(x-3)(x-5)\ldots(x-63)}$ For arbitrary valid numbers X from areas definitions this expressions.

Exercise II

In two ways (using a loop operator with a precondition and a loop operator with a postcondition) for a given value x, find the sum of the series S accurate to within a term of the series whose absolute value is less than eps = 0.0001. Compare this sum with the value of the control function y .

1. $S=\sum_{n=0}^{\infty}\frac{x^n}{n!}=1+\frac{x}{1!}+\frac{x^2}{2!}+...;\ x=1.5;\ and=ex.$

2. $S=\sum_{n=0}^{\infty}\frac{x^n\ln^n 3}{n!}=1+\frac{x\ln 3}{1!}+\frac{x^2\ln^2 3}{2!}+...;\ x=1;\ y=3^x.$

3. $S=\sum_{n=0}^{\infty}x^n=1+x+x^2+x^3+...;\ x=0.7;\ y=\frac{}{1-x}.$

4. $S=\sum_{n=0}^{\infty}\frac{(-1)^n x^n}{2\cdot 2^n}=\frac{1}{2}-\frac{x}{4}+\frac{x^2}{8}-...;\ x=1.2;\ and=\frac{1}{2+x}.$

5. $S=\sum_{n=1}^{\infty}\left(\frac{1}{n}+\frac{3^{n-1}}{2^n}\right)x^{n-1}=\frac{5}{6}+\frac{13}{36}x+\frac{}{216}x^2+...;\ x=-0.8;\ y=\frac{5-2x}{6-5x+x^2}.$

6. $S=\sum_{n=0}^{\infty}\frac{(-1)^n x^{2n}}{(2n)!}=1-\frac{x^2}{2!}+\frac{x^4}{4!}-...;\ x=0.3;\ y=\cos x.$

7. $S=\sum_{n=1}^{\infty}(-1)^{n-1}\frac{x^n}{n}=x-\frac{x^2}{2}+\frac{x^3}{3}-...;\ x=0.4;\ y=\ln(1+x).$

8. $S=\sum_{n=1}^{\infty}(-1)^{n+1}\frac{\sin(nx)}{n}=\sin x-\frac{\sin 2x}{2}+...;\ x=-\frac{\pi}{2};\ y=\frac{x}{2}.$

9. $S=\sum_{n=0}^{\infty}\frac{x^{2n+1}}{(2n+1)!}=x+\frac{x^3}{3!}+...;\ x=2;\ y=\frac{e^x-e^{-x}}{2}.$

10. $S=\sum_{n=0}^{\infty}(-1)^n\frac{x^{2n+1}}{2n+1}=x-\frac{x^3}{3}+\frac{x^5}{5}-...;\ x=0.6;\ y=arctg\ x.$

11. $S=\sum_{n=0}^{\infty}(-1)^n\frac{x^{2n+1}}{(2n+1)!}=x-\frac{x^3}{3!}+\ldots;\ x=1.4;\ y=\text{without } x.$

12. $S=\sum_{n=0}^{\infty}\frac{x^{2n}}{(2n)!}=1+\frac{x^2}{2!}+\frac{x^4}{4!}+...;\ x=0.5;\ y=\frac{(and^x+e^{-x})}{2}.$

13. $S=\sum_{n}^{\infty}\frac{\sin(2n-1)x}{2n-1}=\sin x\pm\frac{\sin 3x}{1}+...;\ x=\frac{\pi}{16};\ and=\frac{\pi}{4}.$

3 2 4

14. $S=\sum_{n=1}^{\infty}(-1)^{n+1}\frac{\cos nx}{n}=\cos x-\frac{\cos 2x}{2}+\ldots;\ x=\frac{\pi}{8};\ and=\ln\left(2\cos\frac{x}{2}\right).$

15. $S=\sum_{n=1}\frac{\cos nx}{n}=\cos x+\frac{\cos 2x}{2}+\ldots;\ x=\pi;\ and=-\ln\left(2\text{without}\frac{x}{2}\right).$

16. $S=2\sum_{n=1}^{\infty}\frac{x^{2n-1}}{2n-1}=2\left(x+\frac{x^3}{3}+\frac{x^5}{5}+\ldots\right);\ x=-0.4;\ y=\ln\frac{1+x}{1-x}.$

17. $S=\sum_{n=1}^{\infty}\frac{1}{(2n-1)x^{2n-1}}=\frac{1}{x}+\frac{1}{3x^3}+\frac{1}{5x^5}+\ldots;\ x=3;\ y=\frac{1}{2}\ln\frac{x+1}{x-1}.$

18. $S=\sum_{n=1}\frac{1}{(2n-1)}\left(\frac{x-1}{x+1}\right)^{2n-1}=\frac{x-1}{x+1}+\frac{1}{3}\left(\frac{x-1}{x+1}\right)^3+\ldots;\ x=0.6;\quad y=\frac{1}{2}\ln x.$

19. $S=\sum_{n=0}^{\infty}(-1)^n\frac{x^{2n}}{n!}=1-\frac{x^2}{1!}+\frac{x^4}{2!}-\frac{x^6}{3!}+\ldots\ x=\ ;\ -0.7;\quad y=e^{-x^2}.$

20.. $S=\sum_{n=0}^{\infty}(n+1)x^n=1+2x+3x^2+4x^3+\ldots;\ x=-0.7;\ y=\frac{1}{(1-x)^2}.$

21. $S=\sum_{n=0}^{\infty}\frac{x^{2n+1}}{2n+1}=x+\frac{x^3}{3}+\frac{x^5}{5}+\frac{x^7}{7}+\ldots;\ x=0.35;\quad y=\ln\sqrt{\frac{1+x}{1-x}}.$

22. $S=\sum_{n=1}^{\infty}\frac{(-1)^n}{n}\left(x+1\right)^{2n}=-\left(x+1\right)^2+\frac{1}{2}\left(x+1\right)^4+\ldots;\ x=-1.2;\ y=\ln\frac{1}{2+2x+x^2}.$

23. $S=1-\sum_{n=1}^{\infty}(-1)^{n+1}\frac{1\cdot 3\cdot 5\cdot\ldots\cdot(2n-3)}{2\cdot 4\cdot 6\cdot\ldots\cdot 2n}x^n=1+\frac{1}{2}x-\frac{1}{2\cdot 4}x^2+\frac{1\cdot 3}{2\cdot 4\cdot 6}x^3-\ldots;$
$x=-0.8;\ y=\sqrt{1+x}.$

24. $S=1+\sum_{n=1}^{\infty}\frac{1\cdot 3\cdot\ldots\cdot(2n-1)}{2\cdot 4\cdot\ldots\cdot 2n}x^{2n}=1+\frac{1}{2}x^2+\frac{1\cdot 3}{2\cdot 4}x^4+\frac{1\cdot 3\cdot 5}{2\cdot 4\cdot 6}x^6+\ldots;\ x=0.64;$
$y=\frac{1}{\sqrt{1-x^2}}.$

25.. $S=\sum_{n=1}\frac{\text{without}\,nx}{n}=\text{without}\,x+\frac{\text{without}\,2x}{2}+\ldots;\ x=\frac{\pi}{3};\ the=-\frac{x}{2}.$

Task III

1. Which quantity warmth will be required for evaporation 100 g water at 100, 110, …, 200°C?

2. Determine internal energy 1 G helium at temperature 10, 15, …, 50°C.

3. IN vessel volume 1 cm^3 located 10^{16} molecules gas. Define gas pressure in a vessel at a temperature of 0, 1,…,10°C.
4. What is it? average quadratic speed molecules oxygen at 10, 12, …, 20°C?
5. How many warmth stand out at condensation 100, 200, …, 900 g water vapor?
6. m^3 boiler contains 25 kg of superheated steam. What is the steam pressure if its temperature is 200, 220, …, 300°C?
7. What is it? average quadratic speed molecules hydrogen at temperature 10, 15, …, 50°C?
8. Determine internal energy 1 G helium at temperature 20, 22, …, 40°C.
9. In a vessel with a volume of 480 cm^3 at a pressure of $2.5 \cdot 10^4$ N/m2 is the gas. Calculate the number of gas molecules for temperatures of 10, 11, …, 20°C.
10. How many warmth will be required For evaporation 200 g water at temperature 100, 110, …, 200°C?
11. What is the root mean square velocity of carbon dioxide molecules at a temperature of 0.5, …, 30°C?
12. Determine the root mean square velocity of hydrogen molecules at 0, 5,…, 50°C.
13. In a vessel with a volume of 450 cm^3 At a pressure of 2.5 x 10^4 N/m^2 there is a gas. Calculate the number of gas molecules for temperatures of 10, 12,…, 20°C.
14. Determine the internal energy of 1 g of hydrogen at a temperature of 20, 21, …, 30°C.
15. Determine the average kinetic energy of rotational motion of hydrogen molecules contained in 1.0 mol at 10, 11, …, 20°C.
16. The density of air under normal conditions is 1.3 g/l. What is the density of air at a pressure of 4.0 x $10^{5?}$ N/m^2 and at a temperature of 50, 60, …, 100°C?
17. Determine the internal energy of 1 g of carbon dioxide at a temperature of 10, 15, …, 50°C.
18. A flask contains a gas under standard conditions. How many molecules of the gas does the flask contain if its capacity is 0.1, 0.2, …, 0.5 liters?
19. What is it? average quadratic speed molecules hydrogen at 0, 2, …, 20°C ?
20. What is the root mean square velocity of oxygen molecules at temperature 0, 2, …, 10°C?
21. In a vessel with a volume of 400 cm^3 There is a gas at a pressure of 2.5 x 10^4 $N/m^{2\cdot}$ Calculate the number of gas molecules for temperatures of 0, 1,…, 10°C.

22. How many warmth will be required For evaporation 150 G water at temperature 100, 105, …, 150°C?
23. Determine internal energy 1 G helium at temperature 20, 21, …, 30°C.
24. What is the root mean square velocity of carbon dioxide molecules at a temperature of 0.5, …, 30°C?
25. Determine the internal energy of 1 g of oxygen at a temperature of 10, 15, …, 50°C.

Control questions

1. What basic algorithmic constructs can be implemented using the C++ language?
2. What it's called cycle?
3. What such body cycle?
4. What such loop parameter ?
5. What such iteration?
6. What such looping?
7. Which types cycles do they exist?
8. Which instructions are used For implementation cycles in C++?
9. Which instructions is used For implementation cycle With parameter?
10. Can whether use V body cycle variable, being loop parameter ?
11. Which instructions is used For implementation cycle With precondition?
12. From what depends quantity repetitions bodies cycle With precondition?
13. Which instructions is used For implementation cycle With postcondition?
14. Which from designs repetitions can bring To looping?
15. How works cycle with counter?
16. IN what case V body cycle are used operator brackets?
17. What are differences between cycles With pre- And With postcondition?

Example execution laboratory Work Task I.

Create a program for tabulating the function

$$y = e^{x} \frac{2.42x^{2} - 6.47x + 1.03}{x - 5.8}$$ on segment [-4, 4] With step by step 0.2.

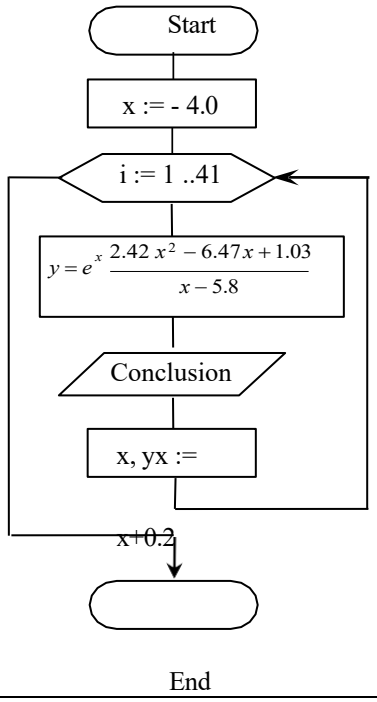
Solution

1. Mathematical model

Before in total, we will determine number repetitions bodies cycle: $S = \frac{4-(-4)}{0.2} + 1 = 41.$

Arguments: variable x of real type. **Results:** value given functions y(real type). **Intermediate values:** integer type counter.

2. Algorithm	3. Program

	`#include<iostream>`
	`#include<conio.h>`
	`#include<math.h>`
	
	`usingnamespace std;`
	
	`int main()`
	`{`
	`  double x = -4, y;`
	`  for(int i=1;i<=41;i++)`
	`  {`
	`  y = exp(x)*(2.42*pow(x, 2.0)-`
	`  6.47*x+1.03)/(x-5.8);`
	`  cout<< " x = " <<x<< " y = "`
	`  <<y<<endl; x+=0.2;`
	`  }`
	`  _getch();`
	`  return 0;`
	`}`

4. Result work programs

```
…
x = - 3.6  y = - 0.161865
x = - 3.4  y = - 0.185016
…
x = - 2.2  y = - 0.373639
…
```

Exercise II. For x = 0.47 find amount rows

$$y = x + \left(\frac{1}{2}\cdot\frac{x^3}{3} + \frac{1\cdot 3}{2\cdot 4}\cdot\frac{x^5}{5} + \ldots + \frac{1\cdot 3\cdot 5\cdot \ldots\cdot(2\nu)}{2\cdot 4\cdot 6\cdot \ldots\cdot 2n}\cdot\frac{x^{2n+1}}{2n+1} + \ldots \right)$$

accurate to a term of the series smaller (in absolute value) eps = 0.0001, using a loop operator with a precondition. Compare the resulting sum with the value of the function for control $y = \arcsin x$.

Solution

1. Mathematical model. Let p1 denote successive products odd numbers.Through p2 - sequential works even numbers.

In C++ (`math.h`), there is `arcsin` function `x` , therefore To calculate the control function, we use the relation: `y = asin(x).`

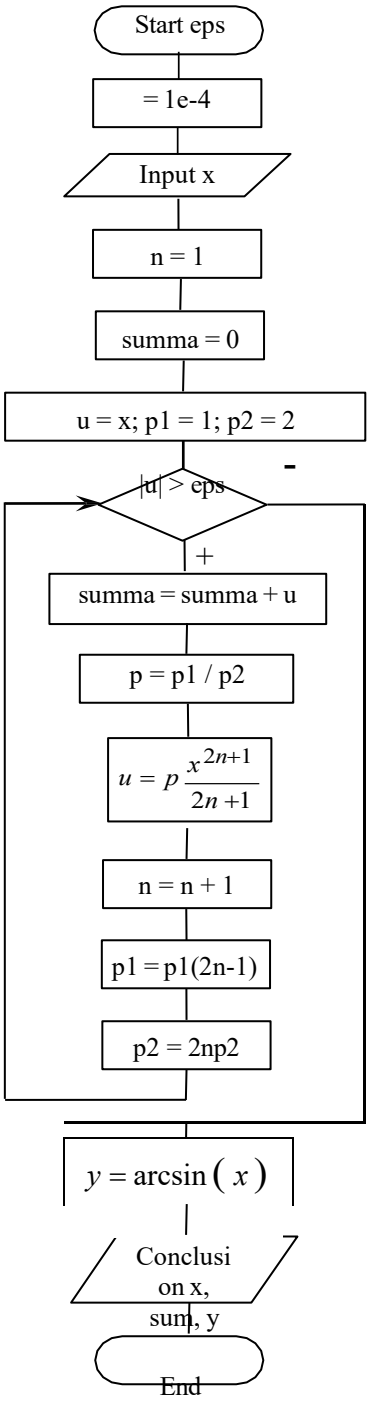
Arguments: `x` - variable material like, `-1 < x < 1` .

Results: `summa` - sum members rows material type;
`y` is the value control functions (real type).

Intermediate quantities:

eps – a constant with which the absolute value of the current member of the series is compared: the members of the series represent a decreasing sequence, therefore, according to the condition, the calculation of the sum of the members of the series will continue until the absolute value of the current member of the series is less than eps ;
p1 – consecutive works odd numbers, material type;
p2 – sequential works even numbers, material type;
p - private, received at division p1 on p2 , material type;
n - number member rows, whole type; u - current member rows, computable By formula $u = p\frac{x^{2n+1}}{2n+1}$, thing type;

2. Algorithm	3. Program




```
// calculation amounts members
rows #include <iostream>
#include <conio.h> #include
<math.h>

usingnamespace std;

int main()
{ // description And initialization given accuracy
      double eps = 0.0001;
      // description variables, input X, initial
      // installations
      double x, u, p1, p2, p; cout<<
      "Enter the value of x: " ;
      cin>>x;
      int n = 1;
      double summa = 0;
      u = x;
      p1 = 1;
      p2 = 2;
      // organization calculations amounts members
      series while (abs(u) > eps)
      {
            summa += u;
            p = p1/p2;
            u = p*pow(x, 2*n+1)/(2*n+1);
            n++;
            p1 = p1*(2*n-1);
            p2 = p2*2*n;
      }
      // calculation values control functions double y
      = asin(x);
      // conclusion results
      cout<< " Sum this rows at x = " <<x<< ", with an
      accuracy of 0.0001, S = " << summa<<endl;
     cout<< " Meaning control functions, at = " <<y;
      _getch();
      return 0;
}
```

4. Result work programs

```
Enter the value x: 0.47
Sum this rows at X = 0.4700 precision to 0.0001, S = 0.4893 Control
function value, y = 0.4893
```

Conclusion: Accuracy to 0.0001 meaning amounts rows And meaning control functions coincide, therefore, the calculation of the sum of the terms of the series is programmed correctly.

Note **:** To write a program using a loop with a postcondition, it is enough V given below program **the while** `(abs(u)> =` construct `eps)` `{ body cycle }` replace on **do{** `body cycle` **} while** `(abs(u) >= eps);`

5. Subroutines

5.1. Concept subroutines

When solving problems, we often encounter situations where the same commands must be executed for different sets of input data. In some cases, a repeat command can be used to organize repetitive calculations. Sometimes, it makes sense to isolate repetitive command sequences into a separate block, placing it outside the main program, and then calling this block at the appropriate location by specifying the appropriate set of parameters. Such a block, which implements a complete stage of data processing and can be accessed from various parts of the main program, is called **a subroutine.** A subroutine is designed as a closed program section with a clearly defined input and output. Using subroutines saves RAM, since memory is allocated to store variables used only within the subroutine for the duration of its execution.

Most programming languages contain the following types of subroutines:
- built-in subroutines;
- library subroutines;
- subroutines, created programmer.

In C++ (as in C), subroutines are implemented as functions. A distinction is made between library (standard) functions and functions created by the programmer.

Library (standard) functions

C++ provides standard functions that have specific names and perform standard actions. These functions are defined in the language and bundled into header files or libraries. To use a library function, you must include the corresponding header file in your program. To call such a function, simply In the body of the main or user-defined function, specify its name and list of actual parameters.

Functions, created programmer

C++ allows a programmer to define his own function and then use it in the same way as library functions.

5.2. Formal And factual parameters

A subroutine (or, in C++, a function) is composed formally. The variables used within it are called **formal** because they do not exist in the same capacity as ordinary variables and constants.

Formal parameters are variable identifiers that pass information from the main program to the subroutine and back. Their names are used in the body of the subroutine, but in reality, it will process factual parameters, transmitted at call, which

may have completely different names. When a subroutine is called from the main program, the formal parameters are replaced **by actual ones** , establishing a one-to-one correspondence between them that satisfies one condition: the corresponding parameters must match in order and type. The subroutine block is executed for the actual set of parameters.

Actual parameters are information about what values to pass to the subroutine's arguments and what program names to pass the subroutine's result values to.

Description subroutines asks formal diagram processing information, while the subroutine call operator configures this formal scheme for the specific processing of actual data.

5.3. Local And global variables

A variable is a named memory location that stores data of a specific type. A variable has a name and a value. The name is used to refer to the memory location where the value is stored. A variable's value can be changed during program execution. Any variable must be declared before use.

A variable's declaration, in addition to its type and storage class, explicitly or by default defines its scope. The storage class and scope depend not only on the declaration itself but also on its placement in the program text.

The scope of an identifier is the part of the program in which it can be used to access the associated memory location. Depending on the scope, a variable can be **local** or **global** .

If a variable is defined inside a block (remember that a block is delimited by curly braces), she is called **local** , region her actions – from declaration point to the end of the block, including all nested blocks. If a variable is defined outside of any block, it is called **global** and its scope The action is considered to be the file in which it is defined, from the point of description to its end.

The memory class determines the lifetime of the scope of a software object (in particulars variable). If Class memory Not specified obvious in this way, It is determined by the compiler based on the context of the declaration.

Time life Maybe be constant(in flow execution block).

Region the visibility of an identifier is called part program text from which normal access to the memory region associated with the identifier is permitted. Most often, the visibility scope coincides with the action scope. An exception is when a variable is declared in a nested block. with the same name. In this case, the outer variable in the nested block is invisible, even though the block is within its scope. However, this variable, if global, can be accessed using the scope access operator "::".

For tasks class memory are used next specifiers:

- **auto –** automatic variable. Memory for it is allocated on the stack and, if necessary, initialized each time the statement containing it is executed her definition. Liberation memory is happening at exit From the block in which the variable is declared. Its lifetime is from the moment of its declaration until the end of the block. This specifier is not used for global variables, and for local variables it is accepted by default, so specifying it explicitly doesn't make much sense.
- **extern** – means that the variable is defined elsewhere in the program (in another file or further down the text). This is used to create variables that are accessible in all program modules in which they are declared.
- **static** – a static variable. Lifetime is constant. Initialized once during the first execution of the statement containing the variable definition. Depending on the location of the declaration statement, static variables can be global or local. Global static variables are visible only within the module in which they are declared.
- **register** – similar to auto, but memory is allocated in processor registers whenever possible. If the compiler doesn't have this capability, variables are treated as auto.

```
int a;                  // 1 global variable a
int main()
  {
     int b;             // 2 local variable b
     extern int x;      // 3 variable x is defined elsewhere
     static int c;      // 4 local static variable with a= 1;
                        // 5 assignment to a global variable
     int a;             // 6 local variable a
     a= a;              // 7 assignment to a local variable
     ::a= 3;            // 8 assignment to a global variable
     return 0;
  }
int x=4;          // 9 defining and initializing x
```

In this example, the global variable **a** is defined outside of blocks. Memory for it is allocated in the data segment at program start, and its scope is the entire program. Its visibility scope is the entire program, except for lines 6-8, since the first of these defines a local variable with the same name. Its scope begins at the point of its declaration and ends when the block is exited. Variables **b** and **c** are local, their scope is the block, but their lifetimes differ: memory for **b** is allocated on the stack upon block entry and freed upon exit, while variable **c** is located in the data segment and exists for the entire duration of the program.

If the initial value of variables is not explicitly specified during definition, the compiler assigns global and static variables the zero value of the corresponding type. Automatic variables are not initialized.

A variable name must be unique within its scope (for example, there cannot be two variables with the same name in one block).

To determine the scope of identifiers, one must adhere to rules:

- every identifier must be declared before it is used ;
- region actions identifier is block, V which He described;
- All identifiers V block should be unique, those. Not repeat;
- one And That same identifier Maybe be differently defined V each individual block.

If the identifier of a user-defined function matches the name of a standard function, the latter is not available in the scope of the user-defined function, i.e. the standard function is ignored, and the user-defined function is executed.

5.4. Functions

A **function** is understood as a set of operators that jointly perform a certain action with a specific purpose, designed in accordance with the requirements of a programming language.

WITH concept functions V language C++ connected three the following components:

– description functions;
– prototype;
– call functions.

A function description consists of two parts: a header and a body. The function description is written as follows:

```
result_type function_name ([parameter_list]) // header
  {
/* declarations and operators
     or, in other words, the body of the function*/
  }
```

Here, `result_type` *is* the return type. If the type specifier is missing, the function is assumed to return an integer (**`int`** **`). If the function does not return any value, the void`** specifier is used in place of the type **`.`** The parameter list must specify a type for each parameter. If there are no parameters, the list may be empty or have **the void specifier** .

The function body is a sequence of declarations and statements describing a specific algorithm. An important statement in the function body is the return statement: **`return`** **[expression];. The `return`** statement has a dual purpose. It ensures immediate return to the calling function and can be used to pass the calculated value to the function. A function body may contain multiple **`return statements`** **,** but it may also contain none. In the latter case, return to the calling program occurs after the last statement in the function body is executed.

A function prototype can be specified before calling the function instead of a declaration functions For Togo, to compiler could execute check

Argument and parameter types correspond. A function prototype is the same form as a function header, ending with " **;** ". Function parameters in a prototype may *have names* , **but the compiler doesn't need them** .

The compiler uses a function prototype to compare argument types with parameter types. The C language does not provide automatic type conversion when arguments do not match their corresponding parameter types; thus, C provides strict type checking. With a prototype, called functions do not have to be located in the same file as the calling function. This means that function body declarations can be placed in another file, which must be included in the source file using the **`#include` directive** .

A function call can be written as a statement if the function *has no* return value, or as an expression if *it does have* a return value.

IN first case operator has next format:

```
function_name ([argument_list]);
```

For example: `Print(“HelloWorld!”);`

In second case expression is being recorded next way:

```
Variable = function_name (argument_list);
```

For example: `z=max(x,y);`

The value of the evaluated expression is the return value of the function. The return value is passed to the function caller and is the result of its execution.

The number and types of actual parameters (arguments) must match the number and types of the function's formal parameters (parameters). When a function is called, the actual parameters (arguments) are substituted for the formal parameters (parameters). Thus, the function declaration defines a formal information processing scheme, while the function call operator configures this formal scheme for the specific processing of actual data.

Below is given example functions With returnable meaning.

Example 1. Create a program containing a call to the function for calculating the maximum of two numbers.

Possible solution this tasks has view:

```
//description functionsmax located V this file below bodies main functions #
include <conio.h>
# include <iostream>
usingnamespace std;

// description of the user-defined function
int max ( int a, int b) // function header
{
     int c;                //
     c=(a>b)?a:b;          // function body
```

```
      return c ;            //
}

void main()
{
      setlocale(LC_ALL,"Rus");
      int x,y,z;
      cout << "\n one by one enter X And at \n";
      cin >> x >> y;
      z=max(x,y);           // call
      functions cout << "z=" << z;
      return ;
}
```

Description functions located V one file With main program.

The function call is the expression on the right side of the operator assignment `z = max(x,y);` , at execution whom values arguments `x` And `y` are substituted instead of parameters `a` and `b,` respectively (passing parameters to a function by value). After executing the function body, the return value is passed to the function caller and assigned to the variable `z` .

In the program discussed above, the function has a return value. There are tasks that don't require a return value. Such an example is given below.

Example 2. Usage functions without returned values.

Create a program that includes a function, When accessed, the character typed on the keyboard is displayed on the screen. Pressing the `Enter key` terminates the program.

```
      ...
      void print(char);      // function prototype

      void main()
      {
      char x;
      cout<<”\nEnter the characters.”<<” Enter – end of input”;
      do
            {
            x = getch();
            print (x);
            }
      while (x! == 13);
      return 0;
      }
      void print(char a) { cout <<a << “\t“; }
```

IN functions `print()` absent returned meaning, That's why The function is called using `the print(x); function call operator` . As a result of calling the function, the character entered from the keyboard using the `x = getch() operator will be displayed on the screen` .

Laboratory Job No. 5
Usage functions For solutions applied tasks

PURPOSE OF THE WORK: to consolidate theoretical knowledge and acquire skills in programming functions.

EXECUTION WORKS: master theoretical material, execute general for all task I and in accordance with the option, create a program (task II), implementing the user function.

Exercise I

Study order announcements And descriptions functions:

- implement (debug and run programs) examples 1, 2, paragraph 5.4 given in the theoretical information;
- design V notebooks, having written down condition, block diagram And result operation of the relevant programs (examples 1, 2, paragraph 5.4).

Exercise II

IN in accordance With option make up And implement program:

1. Three right triangles are given by their legs. Determine which one has the larger area.
2. Three segments are given vertex coordinates. Determine whether any of them are of equal length.
3. On average, three ships arrive at the port per day. What is the probability that Will 2 ships arrive at the port? 4 ships? The probability is calculated using the formula: $P = 3^k e^{-3} / k!$.
4. Find the arithmetic mean of the real roots of three quadratic equations.
5. Three different blocks, each shaped like a cube with a given edge, are given. A regular quadrangular pyramid (the base being the cube's edge) is cut from each block. Calculate the volume of material wasted .
6. Given valid numbers A And b . Calculate,

$$g(1.2, a)+g(b, a)-g(2a-1, ab), \text{ где } g(x, y)=\frac{\left(x^2+y^2\right)}{x^2+2xy+3y^2+4}$$

7. edge lengths of an arbitrary triangular pyramid are given . Calculate the total surface area of this pyramid.
8. Calculate the volume of a "snowman" consisting of three balls with arbitrary radii.

9. Each of the three rings is defined by an inner and outer radius. What is the area of the largest ring?
10. Calculate meaning expressions a! + b! + c! + d!.
11. Calculate the area of a "Christmas tree" consisting of four triangles with known sides.
12. Find minimum meaning among roots three square equations (each equation is defined by three coefficients).
13. Given valid numbers a And b. Calculate

$$\max (u + v^2, 3.14),$$

Where $u = \max(a, b)$, $v = \max(ab, a + b)$, A max - subroutine For calculating the maximum value for two given numbers.

14. The radii of three spheres are given. Calculate the arithmetic mean of their volumes .
15. Dana arithmetic progression a_n, a_1 – first her element, d - difference. Find the second, third, fourth, and seventh elements and their sum.
16. Given valid numbers s And t . Calculate $f(t, -2s, 1.17) + f(2.2, t, s - t)$,

$$f(x, y, z) = \frac{2x - y - \sin z}{5 + |z|}.$$

17. Find the largest of five given numbers using the find largest of two numbers routine .
18. Calculate meaning functions:

$$y = \varphi(3) + \varphi(7.6) \cdot (\varphi(11.2) - \varphi(5)), \text{ If } \varphi(x) = \sqrt{2x^2 - 4x + e^x}$$

19. Calculate meaning functions

$$f(x) = 2\sin x - 4.2\cos x + e^x \quad \text{And} \quad g(x) = \frac{2x^2}{10\sqrt{5x}}$$

For values $X = 0.1;\ 0.35;\ 0.53$.

20. Calculate meaning functions $x = 2f(15) - 4f(-1)\, g(11) + g(32)$, If $f(x) = 10x + e^{2x}$; $g(x) = \sqrt{2x + x^3}$.
21. From Cuba with side A sawed out cube with side b ($a > b$). Which percent material went to waste?
22. Each one from three circles given coordinates center And coordinates of a point on a circle. What is the greatest circumference of the circle?
23. Find the smallest from four given numbers, using subroutine finding the smaller of two numbers.
24. Given valid numbers a And b. Calculate

$$\min (2u - v^2, 4.5),$$

where $u = \min(a, b)$, $v = \min(a/b, b)$, and min is a subroutine for calculating the minimum value for two given numbers.

25. Calculate big material roots square equations a x^2 + b x + With = 0 and t x^2 + p x + q = 0 if they exist. In Otherwise, display the corresponding message on the monitor screen.

Control questions

1. What such subroutine?
2. How subroutine is possible implement V C++?
3. Which parameters are called formal?
4. Which parameters are called factual?
5. What is a variable?
6. What such region actions identifier?
7. What such local variable? What such global variable?
8. What is it? region actions local identifiers?
9. What is it? region actions global identifiers?
10. What such function?
11. How is different announcement functions from her definitions (descriptions)?
12. List composite parts descriptions functions.
13. What such prototype functions?
14. How is described function user V program?
15. What such returned meaning functions?
16. How is being carried out call functions from basic programs?

Example execution laboratory work

Exercise. Asking With keyboards coordinates peaks triangle, determine whether it is equilateral.

Solution

1. Mathematical model

A triangle is equilateral if the lengths of its sides are equal. We find lengths sides triangle, using subroutine finding the distance between two points.

Arguments: a1, b1, c1, a2, b2, c2 whole type - coordinates vertices of the triangle.

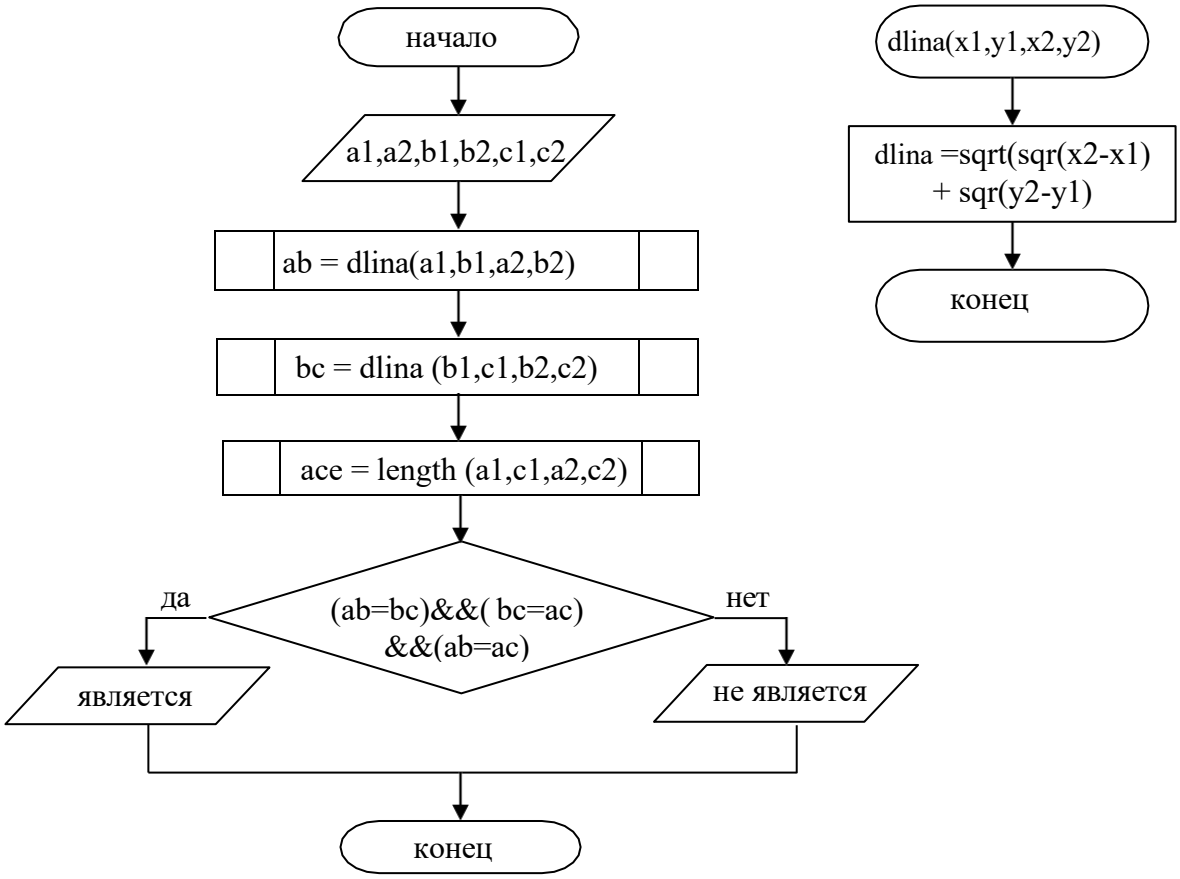
Results: text message.

Intermediate quantities: a b , b s, ace material type - lengths sides of the triangle.

Distance between two dots A(x1,y1) And B(x2,y2) we find By formula: AB =

sqrt(sqr(x2 - x1) + sqr(y2 - y1).

2. Algorithm – main program: – subroutine:



3. Program

```
#include<iostream>
#include<conio.h>
#include<math.h>
constdouble eps = 0.01;
usingnamespacestd;

double Dlina (double x1,double y1,double x2,double y2){
	return (sqrt((x1-y1)*(x1-y1)+(x2-y2)*(x2-y2)));
}

int main(){
	cout<<endl<<"Задайте координаты точек A, B, C"<<endl;
	double a1, a2, b1, b2, c1, c2;
	cin>>a1>>a2>>b1>>b2>>c1>>c2;

	double ab, bc, ac;
	ab = Dlina(a1, b1, a2, b2);
	bc = Dlina(b1, c1, b2, c2);
	ac = Dlina(a1, c1, a2, c2);

	if ((fabs(ac-bc)<eps) && (fabs(bc-ab)<eps) && (fabs(ac-ab)<eps))
		cout<<endl<< " Triangle is equilateral " ;
	else  cout<<endl<< "The triangle is not equilateral" ;
	_getch();
	return 0;
}
```

4. Результатработыпрограммы:

```
Задайте координаты точек A, B, C
   0 0 6 6 5 1

   Треугольник не является равносторонним
```

6. Iteration And recursion

6.1. Concept iterative process

An iterative process can be defined as a computational process based on on repetition sequences operations, at which on Each iteration step uses the result of the previous step. Examples of iterative algorithms include algorithms for calculating the factorial of a natural number, the Fibonacci sequence, the sum of the terms of a sequence with a given number of terms, and so on. Iterative processes are usually programmed using loop constructs.

Example Computing factorially number *n* .

```
int factor ( int n)
{
   int f=1;
   for ( int i = 2; i <=n; i++)f *= i;
   return f;
}
```

6.2. Concept recursion

At the same time, repeating the sequence of actions can be This can also be accomplished using another method based on the principle of the possibility of explicitly including a construct of the same type in programming language constructs. In particular, in the C++ programming language, any function can call itself. Such a call is called **recursion.** A function that calls itself is called **recursive** .

In general, an algorithm for solving a problem is called *recursive* if it is expressed in terms of solving a simpler version of the same problem and, in addition, it provides a solution to the simplest version of the problem.

When calling functions recursively in C++, a stack is used. organizing storage of local parameters.

A stack is the simplest dynamic data structure. Elements are added to and retrieved from the stack from one end, called the top of the stack. This is said to be that the stack implements the LIFO (last-in-first-out) service principle.

Stacks are widely used in system software, compilers, and various recursive algorithms.

Differences between iteration And recursion:

1) iteration requires a loop and auxiliary quantities, while recursion does not require auxiliary quantities and is usually easier to understand, shorter and more visual than iteration;
2) iteration requires less memory space and less computer time than recursion, which requires stack management costs;
3) at implementation recursion great probability overflow stack.

Therefore, when choosing a method for describing a function, one should first decide whether to prioritize program efficiency or compactness. However, **recursions should be avoided where there is an obvious iterative solution.** However, there are many good examples of recursion applications (constructing graphical images of various patterns, working with binary trees).

When describing a function recursively, the latter must always contain at least one alternative non-recursive branch of the algorithm with condition for the termination of calculations, ending with the return statement of the result.

Distinguish two forms recursive subroutines:

- **straight recursion** ;
- **indirect recursion** .

IN **first case** function contains call this same functions.

Indirect recursion occurs when two or more functions call each other. In the case of two functions, one function calls another function, which in turn calls the original function. To prevent looping, one or both functions must check a termination condition.

Example: Create a recursive function that calculates the factorial of a number *n* using the formula (remember that 0! = 1):

$$n! = \begin{cases} 1, & n = 1; \\ n \cdot (n-1)!, & n > 1. \end{cases}$$

```
long fact (longn)
{
     if(n==0 || n == 1)return 1;
     return(n*fact(n-l));
}
```

That same the most Can write down Briefly speaking:

```
long fact ( long n)
{
     return (n> 1) ? n*fact(nl) : 1;
}
```

Example: Print sequence, consisting of *n* letters A *and n* letters B.

```
void abn ( int n)

{
     cout<<”A”;
     if (n > 1) abn(nl);
     cout<<”B”;
     return
}
```

Laboratory Job No. 6
Programming recursive algorithms

TARGET WORKS: consolidation skills programming With using recursive functions.

EXECUTION WORKS: V in accordance With option **,** using straight recursion, compose And implement program.

Exercise

1. Describe the recursive function NOD(A, B) of integer type that finds the largest general divider (NOD) two whole positive numbers A and B, using algorithm Euclid: GCD(A, B) = GCD(B, A mod B), if B ≠ 0; GCD(A, 0) = A. WITH help this functions find GCD(A, B), GCD(A, C), GCD(A, D), if the numbers A, B, C, D are given.
2. For n = 12 find numbers Fibonacci. Numbers Fibonacci: F(0) = 1, F(1) = 1 F(n) = F(n - 2) + F(n - l)
3. Given whole numbers m And n, Where $0 \le m \le n$, calculate, using recursion, number combinations C(n , m) By formula: $C_n^0 = C_n^n = 1$, $C_n^m = C_{n-1}^m + C_{n-1}^{m-1}$ at $0 \le m \le n$. Taking advantage of formula $C_n^m = \frac{n!}{m!(n-m)!}$ Can check right result.
4. Describe a recursive function that, given a real x and an integer n, calculates the value x^n according to the formula:

$$x^n = \begin{cases} 1, n = 0, \\ \frac{1}{x^{|n|}}, n < 0, \\ x \cdot \left(x^{n-1}\right), n > 0. \end{cases}$$

5. Given a sequence of positive numbers, each ending in a negative number, calculate the number of numbers and their sum using recursion.
6. Write a recursive function K (n) that returns the number of digits in a given natural number n using the formula:

$$K(n) = \begin{cases} 1, & If \ n < 10, \\ K(n \ div \ 10) + 1, & if \ n \ge 10. \end{cases}$$

7. Let us define a function S (n) that calculates the sum of the digits of a given natural number:

$$S(n) = \begin{cases} n, & If \ n < 10, \\ S(n \ div \ 10) + n \bmod 10, & If \ n \ge 10. \end{cases}$$

Describe a recursive function DigitSum(K) of integer type that finds the sum of the digits of an integer K , not Using the loop operator, find the sum of the digits for five given integers.

8. Given a vector X of n real numbers, find the minimum element of the vector using an auxiliary recursive function that finds the minimum among the elements of X, starting with the *nth* .

9. Write a recursive function to find the binomial coefficients (for a given $M \geq i \geq j > 0$, calculate all C_i^j):

$$C_m^m = \begin{cases} 1, \text{If } n = m \text{ or } m = 0; \\ C_{m-1}^{m} + C_{n-1}^{m-1}. \end{cases}$$

10. Write a program to calculate the Ackermann function for all non-negative integer arguments m and n:

$$A(m,n) = \begin{cases} A(0,n) = n+1, \\ A(m,0) = A(m-1,1), m > 0, \\ A(m,n) = A(m-1, A(m,n-1)), m, n > 0. \end{cases}$$

11. For given integration boundaries a and b, calculate the value of the definite integral of the following form:

$$\int e^{ax} \cos^n x \, dx = \begin{cases} \dfrac{e^{ax}\cos^{n-1} x\,(a\cos x + n\sin x)}{a^2+n^2} + \dfrac{n(n-1)}{a^2+n^2}\int e^{ax}\cos^{n-2} x\,dx, n \geq 2, \\ \dfrac{-e^{ax}(\sin x + a\cos x)}{a^2+n^2}, n = 1, \\ \dfrac{e^{an}}{a}, n = 0. \end{cases}$$

12. For given integration boundaries a and b, calculate the value of the definite integral of the following form:

$$\int \cos^n x\,dx = \begin{cases} \dfrac{\cos^{n-1} x \sin x}{n} + \dfrac{n-1}{n}\int \cos^{n-2} x\,dx, n > 2, \\ \dfrac{x}{2} + \dfrac{1}{4}\sin 2x, n = 2, \\ \sin x, n = 1. \end{cases}$$

13. For given integration boundaries a and b, calculate the value of the definite integral of the following form:

$$\int \text{without}^{n}\, xdx = \begin{cases} \underline{1} - \dfrac{\sin^{n-1} x\cos x}{n} + \dfrac{n-}{n}\int \text{without}^{n-2}\, xdx, n>2, \\ \text{without } x, n=2, \\ -\cos x, n=1. \end{cases}$$

14. For given integration boundaries *a* and *b,* calculate the value of the definite integral of the following form:

$$\int \frac{dx}{\sin^{n} x} = \begin{cases} -\dfrac{1}{n-1}\cdot\dfrac{\cos x}{\sin^{n-1} x} + \dfrac{n-2}{n-1}\int \dfrac{dx}{\sin^{n-2} x}, n\geq 2, \\ \lambda\nu\, tg -, n=1, \\ x, n=0; \end{cases}$$

15. For given integration boundaries *a* and *b,* calculate the value of the definite integral of the following form:

$$\int \frac{dx}{\cos^{n} x} = \begin{cases} \dfrac{1}{n-1}\cdot\dfrac{\sin x}{\cos^{n-1} x} + \dfrac{n-2}{n-1}\int \dfrac{dx}{\cos^{n-2} x}, n\geq 2, \\ \ln tg\left(\dfrac{\pi}{4} + \dfrac{x}{2}\right), n=1, \\ x, n=0. \end{cases}$$

16. For given integration boundaries *a* and *b,* calculate the value of the definite integral of the following form:

$$\int x^{n} e^{ax} dx = \begin{cases} \dfrac{x^{n} e^{ax}}{a} - \dfrac{n}{a}\int x^{n-1} e^{ax} dx, n>1, \\ \dfrac{e^{ax}}{a^{2}}(ax-1), n=1. \end{cases}$$

17. For given integration boundaries *a* and *b,* calculate the value of the definite integral of the following form:

$$\int x^{n} a^{mx} right = \begin{cases} \dfrac{x^{n} a}{n\ln a} - \dfrac{n}{mx\ln a}\int x^{n-1} a\, dx, n>1, \\ a \qquad ready^{mx} \\ l \end{cases}$$

$$\left| \begin{array}{l} m\ln a \\ -\dfrac{a^{mx}}{m\left(\ln and\right)^{2}}, \ n=1. \end{array} \right.$$

18. For given integration boundaries *a* and *b,* calculate the value of the definite integral of the following form:

$$\int \ln^n xdx = \begin{cases} x\ln^n x - n\int \ln^{n-1} xdx, n>1, \\ x\ln x - x, n=1. \end{cases}$$

19. For given integration boundaries *a* and *b,* calculate the value of the definite integral of the following form:

$$\int x^m \ln^n xdx = \begin{cases} \frac{x^{m+1}}{m+1}\ln^n x - \frac{n}{m+1}\int x^m \ln^{n-1} xdx, n>1, \\ x^{m+1}\left[\frac{\ln x}{m+1} - \frac{1}{(m+1)^2}\right], n=1. \end{cases}$$

20. For given integration boundaries *a* and *b,* calculate the value of the definite integral of the following form:

$$\int \left(\frac{dx}{a^2 + x^2}\right)^n = \begin{cases} \frac{1}{2(n-1)a^2}\left[\left(\frac{x}{a^2+x^2}\right)^n + (2n-3)\int \left(\frac{dx}{a^2+x^2}\right)^{n-1}\right], n>1, \\ \frac{1}{a}arctg\frac{x}{a}, n=1. \end{cases}$$

21. For given integration boundaries *a* and *b,* calculate the value of the definite integral of the following form:

$$\int author^n xdx = \begin{cases} \frac{tg^{n-1}x}{n-1} - \int tg^{n-2} xdx, n\geq 2, \\ -\ln\cos x, n=1, \\ x, n=0. \end{cases}$$

22. For given integration boundaries *a* and *b,* calculate the value of the definite integral of the following form:

$$\int ctg^n xdx = \begin{cases} -\frac{ctg^{n-1}x}{n-1} - \int ctg^{n-2} xdx, n\geq 2, \\ \ln\sin x, n=1, \\ x, n=0. \end{cases}$$

23. For given integration boundaries *a* and *b,* calculate the value of the definite integral of the following form:

$$\int x^m \sin ax = \begin{cases} -\frac{x^m}{a}\cos ax + \frac{m}{a}\int x^{m-1}\cos axdx, m\geq 1, \\ \end{cases}$$

$\underline{\cos ax}\ a$

$, m =$
0.

24. For given integration boundaries *a* and *b,* calculate the value of the definite integral of the following form:

$$\int x^m \cos ax = \begin{cases} \frac{x^m}{a}\sin ax + \frac{m}{a}\int x^{m-1}\sin axdx, m \geq 1, \\ -\frac{\sin ax}{a}, m = 0. \end{cases}$$

25. Write a recursive function that multiplies two integers, one of which is non-negative, without using the multiplication operation.

Control questions

1. Which process is called iterative?
2. What way are being implemented iterative processes?
3. What algorithm? it's called recursive?
4. Which function it's called directly recursive? Indirectly recursive?
5. What such stack?In Which sequences is happening filling a stack and popping elements from the stack?
6. What should Necessarily be present V body recursively described function?
7. List differences between iteration And recursion.
8. What will happen, If recursive algorithm will call myself myself "infinite" number once?
9. How prevent infinite execution recursive algorithm?
10. Right whether, What solution tasks, feasible recursive algorithm, can be expressed using iteration?

Example of completing laboratory work Assignment.

For given borders integration a And b calculate meaning certain integral next type:

$$\int x^n e^{cx} dx = \begin{cases} \frac{x^n e^{cx}}{c} - \frac{n}{c}\int x^{n-1} e^{cx} dx, n > 1, \\ e^{cx}\left(\frac{x}{c} - \frac{1}{c^2}\right), n = 1; \end{cases}$$

Solution

1. Mathematical model

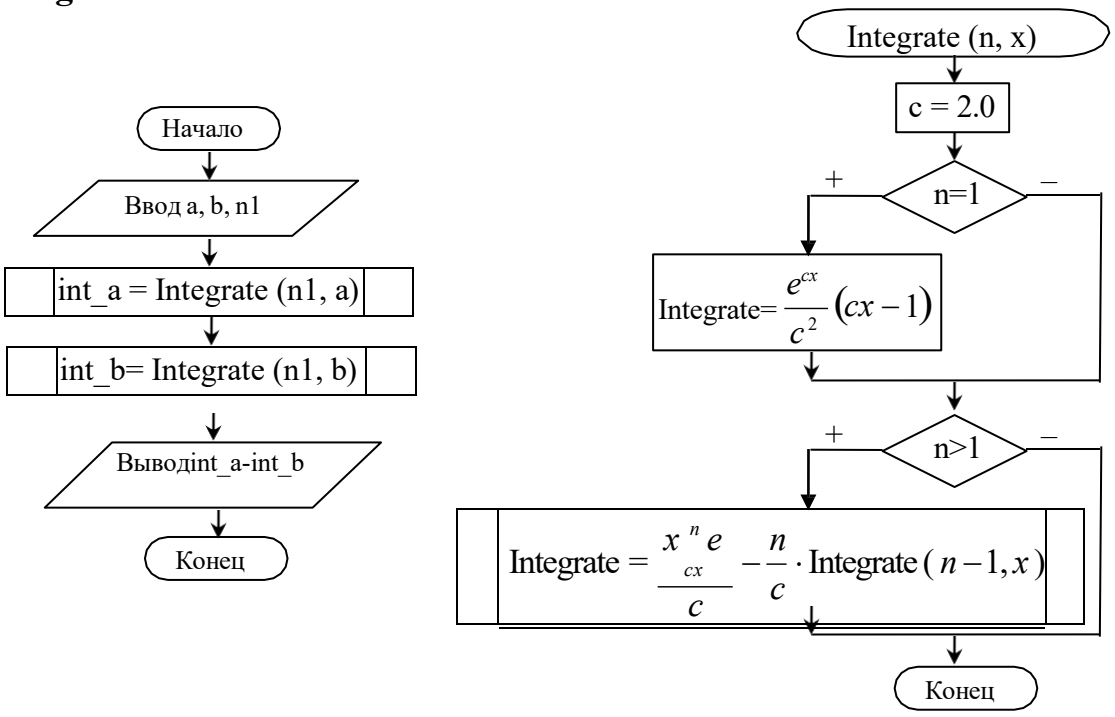
Arguments: boundaries integration a, bwhole type; degree

n1 of an integer type.

Results: difference int_b - int_areal type.

Intermediate quantities: the value of the integral for the lower boundary of integration int_a; the value of the integral for the upper boundary of integration int_b (real type).

2. Algorithm



3. Program

```
//calculation values certain integral #include <iostream>
#include<conio.h>
#include<math.h>

usingnamespace std;

double Integrate(int n, double x);

int main()
{
      double b, a;
      int n1;
      cout<< "Enter A And b, Where b more a : " ;
      cin>>a>>b;
      cout<<endl;
      cout<< " Enter degree n : " ;
      cin>>n1;
      cout<<endl;

      double int_a, int_b;
      int_a = Integrate(n1, a);
      int_b = Integrate(n1, b);
```

```
	cout<< "Meaning certain integral at A = " <<a<< " And b = " <<b<< "
	equals " <<int_b-int_a;
	_getch();

	return 0;
}

double Integrate( int n, double x)
{
	double c = 2.0; // meaning constants A is selected arbitrary
	if (n == 1) return (exp(c*x)/(c*c)*(c*x-1));
	if (n > 1) return (pow(x, n)*exp(c*x)/c-(n/c)*Integrate(n-1, x));
}
```

4. Result work programs

```
Enter a And b, Where b more a: 2 4

Enter the power of n: 3

The value of the definite integral at a = 2 and b = 4 equals 67328.1534
```

7. One-dimensional arrays

7.1. Concept structured type data

Synonyms names "base type" are names "standard", "predefined." The implementation of these types is defined in the language standard, and the programmer has no right to change the predefined ones. Derived type (defined by the programmer) is this type, which should be declared by the user before use. Derived types include arrays, enumerations, functions, structures, references, unions, and classes. It is useful to remember that the type of a data item determines its placement in memory and the set of operations.

A structured type is a set of elements—data—combined by the programmer into a single block in accordance with the specifics of the problem being solved. The data included in the block can be of the same or different types. all block data has general Name. Every element is different from others either by a serial number (index) – in an array, or by their name and, possibly, type - V structure.

Any of the structured data types are characterized by the multiplicity of elements that form this type. Variable or a constant of a structured type is always has several components. Each of these components, in turn, can belong to a structured type, which allows us to talk about possible nesting types.

In C++, the following structured types can be distinguished: arrays, structures, associations, classes, files.

7.2. Concept description type array

When using simple variables, each memory area for storing data has its own name. If a group of values of the same type require similar operations, they are given a single name, and are distinguished by their ordinal number. This allows for a compact recording of multiple operations using loops. A finite named sequence of identical values is called **an array** . The declaration of an array in a program differs from the declaration of a simple variable by the presence after the name of the square brackets, in which the number of array elements (size) is specified.

General view descriptions array:

```
data_type array_name [number_of_elements 1]…[number_of_elements n ];
```

Here:

1) `data_type` is the type of the array elements. This can be any standard or user-defined data type. For example, you could define an array of coordinates of points on a plane, in which case one array element contains two coordinates of a single point. You could define an array of student information, in which case one array element contains a variety of information. O student, For example, surname, Name, age, address And others. IN these

cases For definitions type one element array is used also a derived type, such as an array or structure;

2) `array_name` – correct identifier;

3)`number_of_elements` - This constant, given obviously or named (constant expression). **Variables cannot be used!** ;

4) **`n`** – dimension array.

At **`n`** = 1 description corresponds one-dimensional array (vector),

`n` = 2 - two-dimensional array (matrix),

`n` > 2 - multidimensional array.

On dimension arrays restrictions Not is imposed.

Dimension array Always equal quantity indices (dimensions). **Size** is

usually understood as the number of elements in an array.

Example fixed array on C/C++

```
float a[10]; // description one-dimensional array from 10 material
numbers int Array[10];  // one-dimensional array of integers with
size 10
                // numbering of elements from 0 to 9 double
Array[12][15]; // Two-dimensional array material numbers double
                      // size accuracy 12 by 15. Numbering
                      // by rows from 0 to 11, by columns from 0 to 14
```

Note: When declaring arrays, square brackets are a syntax element and do not indicate that the construct is optional.

Array elements are numbered starting from zero. Initialization values for arrays are written in curly brackets. Values are assigned to elements in order. If there are more elements in the array, the elements for which the values not specified, reset to zero:

```
int b[5] = {3,2, 1}; //b[0] = 3,b[1] = 2, b[2] = 1, b[3] = 0, b[4] = 0
```

Note: Each array element can be accessed directly. To do this, you must specify the array name and the element number (index) in squares. in brackets.

The array size, along with the type of its elements, determines the amount of memory required to allocate the array, which is done at compile time. Therefore, the size can only be specified by a positive integer constant or a constant expression. If the size is not specified when declaring an array, an initializer must be present, in which case the compiler will allocate memory equal to the number of initialization values.

7.3. One-dimensional arrays

A one-dimensional array is an array with one parameter (dimension) characterizing the number of elements in the one-dimensional array. **The dimension is `n` = 1.**

Figure 7.1 shows the structure of a one-dimensional integer array `a`. The size of this array is 10 elements (cells).

- 5	2	0	- 6	12	56	8	23	4	- 3

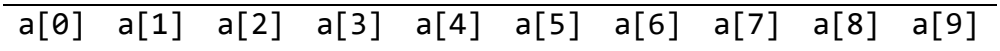


Figure 7.1 - One-dimensional array V C++

The maximum index of a one-dimensional array `a` is equal to 9, but the size of the array is 10 cells, since the numbering of the array cells is always starts with 0. A cell index is a non-negative integer that allows you to access each cell of the array and perform any actions on it (the cell).

```
int a[10]; //an example of declaring the array shown in the figure 7.1,
           // where int is an integer data type ;
           //     a is the name of a one-dimensional array;
           //     10 is the size of the one-dimensional array a.
```

Typically, when describing an array, its size is specified as a named constant, for example:

```
constant n = 6;
int a[n], b[n]; //two one-dimensional arrays of 6 integers are described
```

No operations are defined on an array as a single structure. All actions are performed on individual array elements. Moreover, all operations defined for variables of the same type are valid on array elements.

To contact element array after name array indicated element number in square brackets, for example: `a[4],b[1].`

IN next example is being counted sum elements array.

```
#include <iostream>
int main()
{
      const int n = 10;
      int i, sum;
      int marks[n] = {3, 4, 5, 4, 4};
      for ( i = 0, sum = 0; i<n; i++) sum += marks[i];
      cout <<”Суммаэлементов: ” << sum;
      return 0;
}
```

It is preferable to specify the size of arrays using *named constants,* as is done in the example, since with this approach, to change it, it is enough to adjust the value of the constant in just one place in the program.

Note: When accessing array elements, automatic checking for index out-of-bounds is not performed, which can lead to errors.

7.4. Main actions over elements arrays

1. Initialization array: assignment to everyone element initial values:

a) initialization :

```
int a[6] = {0, 5, -7, 100, 15};  //a[0] = 0, a[1] = 5, a[2] = - 7,
                                 //a[3] = 100, a[4] = 15, a[5] = 0
```

Initialization of a one-dimensional array is performed within curly braces after the equal sign, with each array element separated from the previous one by a comma. The number of constants must correspond to the number of array elements. Otherwise, if the array size is greater than the number of initializers, then elements for which values are not specified are zeroed.

```
int a[]={5,-12,9,10}; //initialization array without defining its size
```

In this case, the compiler will automatically determine the size of the one-dimensional array. The array size can be omitted only during initialization; a standard array declaration requires specifying the array size. To programmatically determine the number of elements in such an array, use the `sizeof operator`:

```
int len; // number elements
len = sizeof(month) / sizeof (int);
```

So, when defining an array, you can initialize its element values by defining a list of initialization values. For example, a year always has 12 months, and the number of days in each month is known, so this structure can be defined as an array:

```
int month[12] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
```

Equivalent initialization is simple **assignment** type:

```
month[0] = 31;        // January
...
month[11] = 31;       // December
```

b) input elements array from the keyboard:

```
constant n = 10; int
a[n];
for (i = 0; i<n; i++)
     {
          cout<< ” Enter ” <<i<< ” element :
          ” ; cin>>a[i];
     }
```

V) formation array With help generator pseudo-random numbers:

```
#include <time.h>     // for time(0)
#include <stdlib.h>   // for srand() and rand()
...
constant n = 10; int
a[n], b[n];
srand(time(0)); // Written once at the beginning
for ( i = 0; i<n; i++) a[i] = rand()%10; //a[i] ∈ [0, 10)
for (i = 0; i < n; b [i++] = rand()%51-25); //a[i] ∈ [-25, 25]
```

`srand()` function is used to update the generation base when using the `rand()` `function` , which generates pseudo-random numbers from the interval [0, RAND_MAX], where RAND_MAX is a constant containing the largest possible meaning For like, selected V quality bases generation. IN example,

The above generates elements of type `int` and the value RAND_MAX = 32767

d) calculation elements array By formula:

```
for ( i = 0; i<n; i++) a[i] = 6 * i - 2;
```

2. Outputting the array to the screen:

```
cout<< ” ArrayA: ” <<endl;
for ( i = 0; i<n; i++) cout<<a[i]<< ”\t” ;
```

3. Processing array

When working with one-dimensional arrays, there are a large number of algorithms that are considered typical, i.e., frequently encountered in practical tasks: array input/output, calculating the values of array elements, finding the sum and/or product of array elements, finding the maximum (minimum) element of an array, ordering array elements, adding (inserting) new elements to an array, deleting data from an array, etc.

Example: Given an array of real numbers with n elements, find the element with the largest absolute value and divide all the array elements by the resulting value. Display the array on the monitor screen after processing.

Possible text programs:

```
#include <math.h>
#include <iostream>

using namespace std;

int main()
{
   const int n=5;
double x[n], max;
int i;     // further set an array of n real numbers
cout << "Input " << n << " numbers:" << endl;
for (i = 0; i < n; i++)
      cin >> x[i];

// Finding the maximum absolute value of an array element

// Let's assume that x[0] is the maximum
   // By module element array: max
= fabs(x[0]);

// Now let's try to refute ourselves:
for(i = 1; i<n; i++) //iterate All elements array With first if(fabs(x[i])
>max) // compare with the current maximum
max = fabs(x[i]); // we are performing

reassignment cout<< "max=" <<max<<endl;  //

Maximum found

for (i = 0; i<n; i++)
x[i] /= max;                    // Divide all elements by max

cout<< "Array:" <<endl;         // Printing an array
for (i = 0; i<n; i++)
cout<<x[i] <<endl;

return 0;
```

```
}
```

Laboratory Job No. 7
Usage numerical one-dimensional arrays

PURPOSE OF THE WORK: mastering methods of describing an array, acquiring skills in organizing input-output and array processing.

EXECUTION OF WORK: master theoretical material, execute general for all task I and in accordance with the option, create a program (task II), if necessary, implementing a user function.

Exercise I

Study order descriptions, input-output And processing arrays:

1. Type and debug a program to find the sum of array elements in odd positions. Document the program in your notebook, recording the conditions, code, flowchart, and results, and explain the purpose of operators 0-4.

```
# include <time.h>
# include
<stdlib.h> int
main()
{
    setlocale(LC_ALL,”Rus”);
    srand(time(0));            // Оператор 0
    const int N=10;
    int Arr[N];
    int sum=0;                 // Оператор 1
    for (int i=0;i<N;i++)
        {
        Arr[i]= rand()%100;    // Оператор 2
        cout<<Arr[i]<<"\t";
        }
    for ( int i=1;i<N;i+=2)    // Operator 3
        sum+=Arr[i];           // Operator 4
    cout<<"\n Sum ="<<sum<<endl;
    return 0;
}
```

2. Type and debug a program to find the sum of array elements that are multiples of 3. Format it in your notebook, writing down the condition, code, flowchart, and result, and explain the purpose of operators 0-3.

```
#include<time.h>
#include<stdlib.h>
int main()
{
    time_t t;
    srand((unsigned)time(&t));         // Оператор 0
    const int N=10;
    int Arr[N];
    int sum=0;                         // Оператор 1
    for (int i=0;i<N;i++)
        {
```

```
        Arr[i]=rand()%50;              // Оператор 2
        cout<<Arr[i]<<"\t";
          }
    for (int i=1;i<N;i++)
    if (Arr[i]%3==0) sum+=Arr[i]; // Оператор 3
    cout<<"\nSumma="<<sum<<endl;
    return 0;
}
```

3. Type and debug a program to find the maximum element of an array of N integers, with the output of the numbers of the largest elements. In your notebook, write down the condition, code, flowchart and result of the work, explain the purpose of operators 1-6.

```
# include <time.h>
# include
<stdlib.h> int
main()
{
    srand(time(0));
    const int N=10;
    int Arr[N];
    int index[N]={0};              // Оператор 1
    int max, k=0;
    for (int i=0;i<N;i++){
            Arr[i]=rand()%100-50; // Оператор 2
            cout<<Arr[i]<<"\t";
            }
    max=Arr[0];                    // Оператор 3
    for (int i=1;i<N;i++)
    if (Arr[i]>=max)       // Оператор 4
                    {
                    max=Arr[i];
                    index[k]=i+1;  // Оператор 5
                    k++;
                    }
    k=0;
    cout<<"\nMax="<<max<<"\nИндексы: ";
    while(index[k]>0)              // Оператор 6
            {
            cout<<index[k]-1<<"\t";
            k++;
    }
    return 0;
}
```

Task II

IN in accordance With option make up And implement program:

1. Given two array different sizes. Define, which elements first array and how many times they occur in the second array.
2. Array contains 2 *n* numbers. From amounts first *n* his elements subtract the sum of the last *n* elements.

3. Transpose the array, i.e. by $a_1, a_2, ..., a_n$ form $a_n, a_{n-1}, ..., a_1$. In the resulting array, find the index of the minimum element.
4. From a given integer array, remove all duplicate elements, leaving only their first occurrences, i.e., from a given array, obtain a new array consisting of distinct integers.
5. Replace negative numbers in the array with their squares, leaving the rest unchanged.
6. In a given array, find the arithmetic mean of positive numbers, the arithmetic mean of negative numbers, and the number of zeros.
7. In an array of 2 n numbers, find the sum of the squares of the elements with even indices and the sum of the cubes of the elements with odd indices.
8. From the numbers $a_1, a_2, ..., a_n$ select those that are greater in absolute value than the given one numbers c, and form a new array from them, preserving the order of the elements.
9. From the array of integers, create three others, the first of which contains numbers that are multiples of 5, the second contains numbers that are multiples of 7, and the third contains the remaining numbers.
10. An array of 100 random integers in the interval [0, 100] is given. Find the sum of those elements of the array that are greater than 15 but less than 45, and also calculate the number of these elements.
11. In a linear array, replace all elements with the number m (m is the index of the maximum element).
12. Given an array consisting of both positive and negative numbers, you need to write the positive numbers first, and then the negative ones. in the same order as they were located in the original array. If there are zeros, write them last.
13. Find the sum of the elements of a given array. Divide each element of the original array by the resulting value.
14. Calculate amount And difference arrays one size.
15. Find the arithmetic mean of the elements of a given array. Transform the original array by subtracting the mean from each element .
16. Given two arrays of the same size, consider them as arithmetic vectors and find their lengths and their dot product.
17. Given two arrays of different sizes, combine them into one array by including the second array between *the* k -th and (k + 1) the first element (k is entered from the keyboard).
18. Subtract the element with number k_1 from the positive elements of the given array and add the element with number k_2 to the negative elements . Replace zero elements with 1. Numbers k_1 and k_2 are entered from the keyboard.

19. Add a given number *a to the even elements of an integer array, and subtract a given number b* from the elements with even numbers .
20. Given the first term of a geometric progression and its denominator, create a one-dimensional array whose elements are the first *n* terms of this progression.
21. Form array from first 30 members sequences Fibonacci.
22. Insert the same number entered from the keyboard before each negative element of the given integer array.
23. Given an array of even size, swap its halves as follows: the first element with the last, the second with the second-to-last element, and so on.
24. Given two integer arrays of the same size, obtain a third array of the same size, each element of which is equal to the larger of the corresponding elements in these arrays.
25. Asked array from *n* whole random numbers, belonging interval [-25, 25]. Find the product of those array elements that are greater than 1 but less than 15, and also calculate the number of even array elements.

Control questions

1. Give it to me definition derivative type data, structured type.
2. Give it to me definition array.
3. What Maybe be type elements array?
4. Name, size And dimension array.
5. What is it? structure one-dimensional array?
6. Rules descriptions one-dimensional array.
7. How is being carried out access To elements one-dimensional array?
8. How is being carried out input array?
9. Which methods input array You do you know?
10. Conclusion linear array.

Example of completing task II of laboratory work Task. Array D contains 24 values atmospheric pressure for every hour V flow days. Define, which meaning atmospheric pressure was the largest and at what time it was recorded.

Solution

1. Mathematical model

The atmospheric pressure values are the elements of the array D[24], and the time values are the indices of the elements. Solving the problem boils down to finding the maximum element in the array and determining its index.

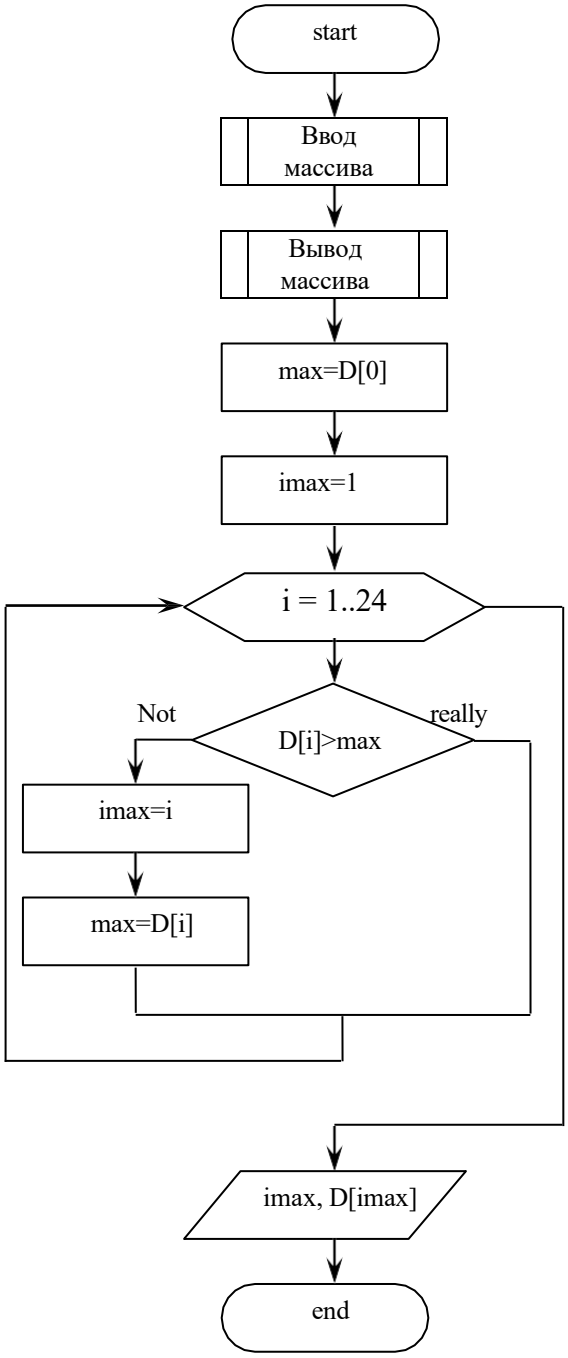
Arguments: D[24] - array integer type.

Results: imax whole type - index maximum element;

D[imax]whole type - meaning maximum element.

Intermediate value: maxwhole type - maximum meaning element array.

2. Algorithm



3. Program

```
#include <iostream> #include
<conio.h> #include <math.h>

usingnamespace std;

void Input( int mass[24]);
void Print( int mass[24]);

int main()
{
	int D[24];
	cout<< " Enter array " <<endl;
	Input(D);
	cout<< "Array pressure values :"
<<endl;
	Print(D);

	int max = D[0], imax = 1;
	for(int i=0; i<24; i++)
	{
		if (D[i]>max)
		{
			imax = i+1;
			max = D[i];
		}
	}

	cout<< " Maximum meaning pressure
: " <<D[imax-1]<< " V " <<imax<< " h. " ;
	_getch();
	return 0;
}

void Input( int mass[24])
{
	for(int i=0; i<24; i++)	{
		cin>>mass[i];
	}
}

void Print(int mass[24])
{
	for(int i=0; i<24; i++)	{
		cout<<mass[i]<<" ";
	}
	cout<<endl;
}
```

4. Result work programs:

```
Array of pressure values :
748 743  756  748  741  760  748  743  757  748  756  756  751  760  754
  756 752  758  758  757  750  747  759  750
Maximum pressure value is 760 at 6 o'clock.
```

8. Two-dimensional arrays

So, **an array** is a data structure that allows you to store under one name a set of data of any, but only one, specific type. An array is characterized by its name, the type of stored elements, its size (the number of stored elements), the numbering of the elements, and its dimensions.

Declaring a variable as a one-dimensional array (an array with dimension 1) looks like this:

```
data_type_id [constant_expression];
```

For example, operator

```
intA[10];//declares an array with the name A containing 10 integers
```

Access To elements of this array is realized by the expression `A[i]` , where `i` - an index, which in this example is an integer in the range 0 - 9. For example, `A[0]` - the value of the first element, `A[1]` - second, `A[9]` - the last one.

Let's remember that declaring an array variable can be combined with assigning initial values to array elements. For example, the operator:

```
int A[10] = {0};// assigns noughties values to everyone elements array
int A[10] = {1,2,3,4,5,6,7,8,9,10};//A[0] = 1, A[2] = 2,…, A[9] = 10
```

If there are fewer initial values than elements in the array, the remaining elements are automatically given initial values of zero.

If an array is not initialized when it is declared, its elements have random values. Elements of such an array cannot be used in expressions until they are assigned values.

Can declare multidimensional arrays, those. arrays, elements which are arrays. For example, a two-dimensional array can be declared like this:

```
int A2[10][3];
```

This operator describes two-dimensional array, which Can introduce as a table consisting of 10 rows and 3 columns.

Like this in this way, **two-dimensional array** - This array, dimension whom equal 2. Visually, two-dimensional array - This ordinary table, with lines And columns or matrix $A = \left\{ a_{ij} \right\}_{m \times n}$. In fact, a two-dimensional array is one-dimensional array one-dimensional arrays. Structure two-dimensional array, With named `A` , size `m` on `n` shown below (see Fig. 8.1).

A[0][0]	A[0][1]	…	A[0][n-1]
A[1][0]	A[1][1]	…	A[1][n-1]
A[2][0]	A[2][1]	…	A[2][n-1]
…	…	…	…
A[m-1][0]	A[m-1][1]	…	A[m-1][n-1]

$$A = \begin{pmatrix} a_{00} & a_{01} & \cdots & a_{0\,n-1} \\ \cdots & \cdots & & \\ a_{m-1\,0} & a_{m-1\,1} & \cdots & a_{m-1\,n-1} \end{pmatrix}$$

Drawing 8.1– Two-dimensional array V view tables (left)or matrices (on the right)

On drawing 8.1 `m` - quantity lines two-dimensional array;
`n` - quantity columns two-dimensional array;
`m * n` - quantity elements array.

General view descriptions two-dimensional array:

```
data_type array_name [number_of_rows][number_of_columns];
```

IN announcement two-dimensional array, Also How And V announcement one-dimensional array, first of all, you need to specify:

- type data;
- name .

Access to the values of elements in a multidimensional array is provided through indices, each of which is enclosed in square brackets. For example, `A2[3][2]` is the value of the element located at the intersection of the fourth row and third column (remember that indices start at 0).

When a multidimensional array is initialized when it is declared, the list of values for each dimension is enclosed in curly braces. The statement below declares a three-dimensional array A3 with dimensions 4 by 3 by 2.

```
int A3[4][3][2] = {{{0,1},{2,3},{4,5}},
                   {{6,7},{8,9},{10,11}},
                   {{12,13},{14,15},{16,17}},
                   {{18,19},{20,21},{22,23}}};
```

This statement creates an array A3, the four rows of which are matrices of the form:

```
0  1      6   7      12  13     18  19
2  3      8   9      14  15     20  21
4  5      10  11     16  17     22  23
```

For example, element `A3[0][1][0]` equal 2, element `A3[3][0][1]` equal 19 and etc. If V list initialization V some from dimensions Not there is not enough data, That All further Not listed elements are considered equal zeros.

All array dimensions must be constants or constant expressions, since memory allocation instructions are generated by the compiler before program execution.

IN memory multidimensional array is located By lines.

Array rows are not separated from each other in any way, meaning, for example, a two-dimensional array is a rectangular matrix only in our imagination. When viewing an array from the beginning, the right index (column number) changes first.

An element of a two-dimensional array is addressed by specifying the row number and column number at the intersection of which it is located, for example:

```
a[1][4]        a[i][j]
```

The compiler treats the first index as a line number, no matter how it is designated in the program.

1. **Initialization** of two-dimensional arrays is similar to initialization of one-dimensional arrays:

a) initialization :

```
constant n = 3; // number of lines
constint m = 4; // number of columns
// allocate memory for the matrix with simultaneous
// initialization of elements
double a[n][m] =
{
   {3, -3.1, 4.6, 5.5},
   {2, 1.2, -10, - 2.5},
   {7.6, 3.1, 1.6, 0.5}
};
```

At initialization two-dimensional array V in the course work programs, it is necessary to organize nested loops:

b) input data from the keyboard:

```
for (i = 0; i < n; i++)
   for (j = 0; j < m; j++)
      cin>>a[i][j];
```

In accordance with the order of the cycles given here Array elements must be entered row by row.

V) formation array With help generator pseudo-random numbers; d) direct assignment (calculation using a formula):

2. **output** must be implemented in a readable format, i.e., one matrix row must be displayed on each line of the screen. To achieve this, the outer loop, in addition to the inner loop, also includes the `cout<<endl; operator` , which moves the cursor to the beginning of the next line of the screen after the current matrix row is displayed.

```
for (i = 0; i < n; i++)
{
   for (j = 0; j < m; j++)
      cout << a[i][j] << "\t ";
   cout << endl;
}
```

Example : Given a rectangular matrix of real numbers of size $n \times m$. Normalize this matrix, i.e., divide the value of all matrix elements by the element with the greatest absolute value.

Possible text programs:

```
# include <iostream> #
include <cmath>

using namespace std;

int main()
{
   const int n = 3; // number of lines
   constint m = 4; // number of columns double
   a[n][m];  // we highlight memory under matrix
   int i, j;
   double max;
```

```
    // Input matrix from keyboard
    cout << "Matriza A("<< n << "*" << m << "):" << endl;
    for(i = 0; i < n; i++)
       for(j = 0; j < m; j++)
          cin>>a[i][j];

    // Search V matrix maximum By module values max =
    fabs(a[0][0]);
    for (i = 0; i < n; i++)
       for (j = 0; j < m; j++)
          if (fabs(a[i][j]) > max)
             max = fabs(a[i][j]);
    cout<< "max=" <<max<<endl;

    // Normalize the matrix
    for (i = 0; i < n; i++)
       for (j = 0; j < m; j++)
          a[i][j] /= max;

    // Output the matrix as a table on the monitor screen
    for (i = 0; i < n; i++)
    {
       for (j = 0; j < m; j++)
          cout << a[i][j] << "\t ";
       cout << endl;
    }
    return 0;
}
```

As the program code shows, working with a matrix almost always requires double loops. Particular attention should be paid to the output of a two-dimensional array as a table.

Laboratory Job #8 Two-dimensional arrays

TARGET WORKS: acquisition skills announcements, organizations input-output and processing of a two-dimensional array.

EXECUTION WORKS: V in accordance With option make up And implement program.

Task I

1. Write a program to generate a matrix, the number of rows and columns of which is determined How random natural number V interval [3, 8], and the elements are random integers from -50 to 50. Replace non-negative numbers in the resulting matrix with one, and negative numbers with zero.
2. In an integer rectangular matrix, numbers that are multiples of a given number m entered from the keyboard are replaced by the quotient of division by m , and the remaining numbers are replaced by their remainders of division by m .
3. In a square matrix of order n , find the sum of all those elements whose index sum is equal to n .
4. Determine whether a given square matrix is symmetric about its main diagonal. Find the trace of this matrix. Enter the matrix elements from the keyboard.
5. Find the number of elements in each of the odd rows of the matrix that are equal to the given number entered from the keyboard.
6. Transform the given rectangular matrix so that all elements in its rows are written in reverse order.
7. Calculate the number and sum of positive elements in the lower left triangle of a given square matrix, including the diagonal ones elements.
8. Given real numbers a_1 , a_2 , …, a_n and a real square matrix of order n (n > 5). Form material matrix size n x (n + 1), by inserting into the original matrix between the fifth and sixth columns a new column with elements a_1 , a_2 , …, a_n .
9. Given a real square matrix of order n , transform this matrix according to the rule: make the row with number k into a column with number k , and make the column with number k into a row with the same number.
10. In this real matrix of size n x m (n > 2, m > 2) swap rows with numbers n - 2 and n - 1.
11. Positive elements of the first row of a rectangular matrix multiply by the first element of the same row, and the negative ones by its last element; do the same with the remaining rows.

12. Replace all elements of the row with number s and column with number k of the rectangular matrix with those of opposite sign (the element located at the intersection of the selected row and the selected column is not changed).
13. From a square matrix, obtain another matrix by eliminating the diagonal elements and subtracting the trace of the original matrix from each element .
14. Given an integer one-dimensional array: $a_1, a_2, \ldots, a_n$. Form a square matrix of order n , whose elements located on the main diagonal are equal to $a_1, a_2, \ldots, a_n$ *, respectively* ; elements located above the main diagonal are equal to one, and elements located below the main diagonal are equal to minus one.
15. Given two rectangular integer matrices A and B of the same size m x n . Create a matrix of the same size in which the elements are equal to 1 if the corresponding elements of matrices A and B have the same sign; - 1 if the corresponding elements have different signs, and zero if at least one of the corresponding elements is zero.
16. Write a program to display an arithmetic square on the screen: the first column and first row are filled with ones, and each of the remaining elements is equal to the sum of its neighbors above and to the left.
17. From a rectangular matrix of size m x n, delete the row with number s and the column with number k ($0 \le s < m$, $0 \le k < n$).
18. Form a one-dimensional array, each element of which is equal to the sum of the negative elements of the corresponding row of a given integer rectangular matrix.
19. Find the number of elements in each column of a rectangular real matrix that are less than the arithmetic mean of the elements of the column in question.
20. Display the maximum elements of each row of a two-dimensional array, indicating their indices.
21. Given two two-dimensional arrays containing positive and negative elements, determine which one has more positive elements.
22. An array of random integers belonging to the interval is given [0, 200]. Determine which elements are more numerous in it: those exceeding k 1 or those not exceeding k 2.
23. Asked array whole random numbers, belonging interval [-50, 50]. Find amount those elements array, which greater than a, But less than b, and also calculate the number of these elements.
24. Swap the maximum and minimum elements of a given two-dimensional array. Display the result as a matrix.

25. Given an integer matrix, form a matrix obtained by permuting the first column with the last, and the second with the penultimate.

26. From a given matrix, obtain another one in the following way: place the second row in place of the first, the third in place of the second, and so on, and place the first in place of the last.

Exercise II

Square matrix A third order formed from real numbers belonging to the range [-10, 10]. Find with accuracy up to $\varepsilon = 10^{-3}$:

1) matrix B By given formula calculations her elements b_{ij} through indices i And j ($i = 1, 2, 3; j = 1, 2, 3$);
2) amount matrices A And B ;
3) matrix, transposed matrix $A + B$;
4) matrix, reverse matrix $A + B$;
5) work matrices $A + B$ And $(A + B)^T$.

Withdraw, using auxiliary function, on screen matrices A , B , $A + B$, $(A + B)^{-1}$, $(A + B)^T$, $(A + B)(A + B)^{-1}$.

1. by = $\frac{\sqrt{i+j+2}}{2i+1}$;
2. by = $\frac{i+2y}{3+\ln(i+j+5)}$;
3. by = $\frac{25\ln(i+0.5)}{i+y+1}$;
4. by = $\frac{}{3y+1}$;
5. b_{ij} = $\frac{\cos(i+2k)}{1+\sin i}$;
6. b_{ij} = $\frac{i+\ln(4+j)}{2i+3}$;
7. b_{ij} = $0.5i + \frac{10}{j+1}$;
8. by = $\frac{}{11i+5}$;
9. by = $\frac{10+\tan(i+2j)}{\sqrt{j}+4}$;
10. by = $\frac{e^{2i+y}}{i+2+3y}$;
11. by = $\frac{\sin(i+2j)+10}{\sqrt{1+2y}}$;
12. b_{ij} = $\frac{i+6+7j}{\ln(i+5)+10}$;
13. by = $\frac{\ln(i^2+4)}{\cos i+2}$;
14. by = $\frac{\cos(i+j/2)}{\ln(7j+7)+i}$;
15. by = $\frac{\tan i+5\ln}{(5+y)+20}$
16. b_{ij} e^{-j}
17. b_{iy} = $\frac{e^{i+j}}{i+y+1}$;
18. by = $\frac{\ln(i+0.5)+2j}{\sin(i+j)+3}$;
19. by = $\frac{3+\sqrt{2y}}{i+1} + 4$;
20. by = $\frac{i+4y+10}{\sin i+\cos j+5}$;
21. by = $\frac{\sqrt{i^2+4y+1}}{5+7y}$;
22. by = $\frac{\sin i}{9+\ln(i+y+10)}$;
23. by = $\frac{\tan(2i+3)+i^2}{\sqrt{i}+1}$;
24. b_{ij} = $\frac{i+2j^4}{5+\sqrt{i^3+j}}$;

25. $b_{ij} = \frac{5\,(i^2 + 2j)}{\text{sneeze}\atop j+2}$;

26. $b_{ij} = \frac{2i^2 + \ln((j+3)}{2\,(i+j)+1}$.

Note : It is recommended to calculate the inverse matrix using the adjoint matrix, and the determinant of the matrix using Sarrus's rule.

Control questions

1. Give it to me definition array.
2. Which types data Not acceptable For components array?
3. Maybe whether left border indices array be negative?
4. Which array it's called two-dimensional?
5. Rules descriptions two-dimensional array.
6. How is being carried out access To elements two-dimensional array?
7. How is being carried out input two-dimensional array?
8. Which methods input two-dimensional array You do you know?
9. Accommodation array V memory COMPUTER.
10. Conclusion two-dimensional array.
11. Maybe whether instruction `cout<<x;` enter array `X` entirely?

Example execution laboratory work

Task: A program that, given a 3 x 4 integer matrix, determines the arithmetic mean of its elements and the number of positive elements in each row.

Solution

1. Mathematical model

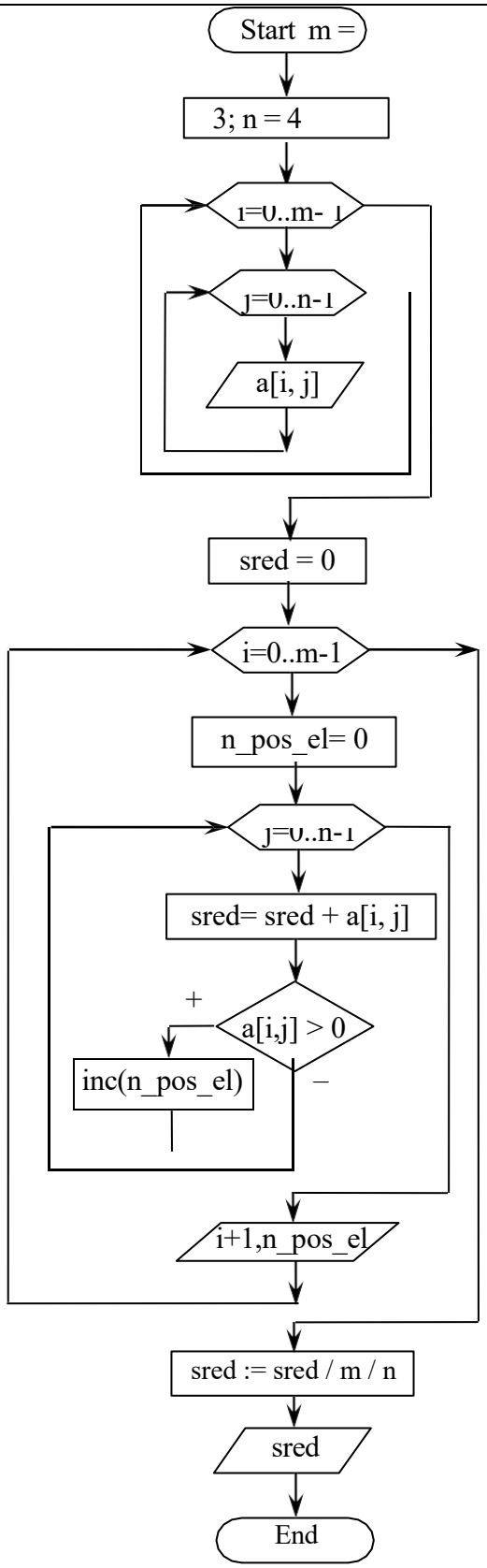
To find the arithmetic mean of an array of elements, you need to find their total sum and then divide it by the number of elements. The order in which you iterate over the array elements (row by row or column by column) does not matter. Finding quantities positive elements each lines requires scanning the matrix row by row.

Arguments: integer array; integer `m` and `n` – number of rows and columns.

Results: arithmetic mean of elements `sred` (real type); number of positive elements in each row `n_pos_el` (integer type).

Intermediate quantities: counters `i, j` (whole type).

2. Algorithm



3. Program

```
#include <iostream>
#include <conio.h>
#include <math.h>

constant m = 3, n = 4;

usingnamespace std;

int main()
{
      int a[m][n]; cout<< "Enter the array
      elements " <<endl; for ( int i=0; i<m;
      i++)
            for(int j=0; j<n; j++)
                  cin>>a[i][j];

      double sred = 0;
      int n_pos_el;
      for(int i=0; i<m; i++)
      {
      n_pos_el = 0;
      for(int j=0; j<n; j++)
      {
            sred += a[i][j]; if
            (a[i][j]>0) n_pos_el++;
      }
      cout<< "In " <<i+1<< "-th line"
<<n_pos_el<< " positive elements"
<<endl;
      }
      sred = sred/m/n;

      cout<< "Average arithmetic:"
<<sred<<endl;
      _getch();

      return 0;
}
```

4. Result of the work programs:

```
Enter the array elements
1  2  3  4  -2  0  -3  -1  1  1  1  0
B 1st line  4 positive elements B 2nd
line  0 positive elements B 3rd line  3
positive  elements  Arithmetic  mean:
0.58
```

9. Algorithms solutions tasks internal sorting and information retrieval algorithms

9.1. Complexity algorithms

Complexity theory provides a methodology for analyzing **the computational complexity** of various cryptographic methods and algorithms.

An algorithm is a formally described computational procedure that receives initial data, also called the algorithm's input or its argument, and produces the result of the computation as an output[4].

Most often, when discussing complexity, we mean *the time complexity of an algorithm* : *T.* The running time of an algorithm is the number of elementary steps it performs. Elementary steps include basic operations such as addition, multiplication, substitutions, unconditional control transfers, and subroutine calls. *T* is usually represented as a function of n , where n is the size of the input data. *The computational time function* is $T_A(n)$.

Typically, the computational complexity of an algorithm is expressed using Big *O notation* , i.e., it is described by the order of magnitude of the computational complexity. This is simply the term in the complexity expansion that grows fastest with n ; all lower-order terms are ignored. For example, if the time complexity of a given algorithm is $4n^2 + 7n + 2$, then the computational complexity is of order $n.^2$, written as $O(n^2)$ [4] .

This notation allows us to see how the size of the input data affects the time and memory requirements. For example, if $T = O(n)$, then doubling the input data will also double the execution time of the algorithm. If $T = O(^{2n})$, then adding one bit to the input data will double the running time of the algorithm.

Algorithms are usually classified according to their time or space complexity. An algorithm is called **constant** if its complexity does not depend on *n:* $O(1)$. An algorithm is **linear** if its time complexity *is* $O(n)$. Algorithms can be **quadratic, cubic** , etc. All of these algorithms are **polynomial,** their complexity is $O(n^m)$, where m is a constant. Algorithms with polynomial time complexity are called **polynomial-time algorithms.**

Algorithms whose complexity is $O(t^{f(n)})$, where t is a constant greater than 1 and $f(n)$ *is* some polynomial function of n , are called **exponential.** The subset of exponential algorithms whose complexity is $O(c)^{f(n)}$), where *c is a* constant and $f(n)$ increases faster than the constant but slower than a linear function, is called **superpolynomial.** Algorithms whose complexity is $O(n!)$, are called algorithms with factorial complexity. Thus, the following relation holds:

$$O(\log n) < O(n) < O(n \log n) < O(n^2) < O(n^3) < O(2^n) < O(n!).$$

Currently, the algorithm is considered to be practically useful only if If his temporary function is growing polynomially relatively

sizes of input data, and to a limited extent. Practical are algorithms of complexity $O(n)$, $O(n2)$. Exponential and factorial algorithms are impractical. Their complexity exceeds any polynomial estimate.

9.2. Production tasks search

In practice, we often encounter applied problems that require local or autonomous searches for desired objects. Consider the problem of searching for an element in an array.

Let an array consisting of one or more elements be given any type other than a file. Suppose that an array stores a set of n ($n \times m$) elements, and it is necessary to determine the position of a certain element in this array. Each element of the array has an index (indices), and the indices are distinct and uniquely identify the elements of the array.

The search task in this case is to find the index(es) element by a given element property.

There are two possible result search:

- the search was successful, i.e. it allowed us to determine the position of the element in the array;
- the search was unsuccessful, i.e. the end of the array was reached, but the element with the specified property was missing.

9.3. Consistent (linear) search

The simplest search method is to sequentially scan the table, one-dimensional array or lines to location element With required properties or establishing the fact that such an element does not exist. For an array whose data is not sorted, the only way to find a given element is to compare each element in the array with the given one. If an array element matches the given one, its position in the array is fixed. This algorithm is called **sequential,** or **linear,** search. The core of such an algorithm is a loop over the array index. For example, if we need to find an integer `x` entered from the keyboard in the array `inta[n]` , about which no additional information is available, the corresponding loop can be written as follows:

```
for (i = 0; i < n; i++) if (a[i] == x) k = i;
```

This way solutions possesses two significant disadvantages:

- if the value `x` occurs in the array several times, the last one will be found;
- After the required value has already been found, the array is scanned to the end, that is, `n is always performed` comparisons.

To address these shortcomings, it's necessary to stop scanning the array immediately after encountering a given number. Since the number of repetitions is unknown in this case, a loop with a precondition must be used:

```
while ((i < n) && (a[i] != x)) i++;
```

9.4. Binary search

The most efficient and widespread method for non-sequential search in one-dimensional *ordered* arrays is **binary** search, also known as *the bisection method, the dichotomy* method *, the logarithmic* search method , or *the bisection method.* The basic idea of binary search is that the searched element `x is first` compared with the middle element of the array `a[n]` . The result of the comparison either solves the problem or allows one to determine in which part of the array—the left or the right—to continue the search. After each such iteration, the search space is reduced by half and in no more than `[log2n]` comparisons we will finish the search, that is, we will either hit the element `x` , or show that `x` does not belong to the array `a[n]` . For example, for `n = 1000` Binary search is 100 times faster than sequential search, and for `n = 1,000,000` – 50,000 times. Of course, if the array is unordered, binary search cannot be applied to it. Therefore, the ordering of the elements in the array plays a key role in binary search.

9.5. Production tasks sorting data

The order in which data is arranged in the computer memory largely determines the speed and ease of execution of algorithms designed to process this data. Therefore V programming often arises task rearranging data into non-ascending or non-decreasing order. This task is called ordering or sorting the elements of a given structure, or in the simplest case, the elements of a one-dimensional array.

Task sorting connected with by many important applications, except Moreover, they serve as a good illustration of algorithm complexity analysis and thus allow one to intelligently select the best among seemingly equivalent methods. Each sorting algorithm has its own advantages and disadvantages, and algorithms should be selected based on the specific problem statement. Algorithm Selection sorting depends significantly from structure of the data being processed, so all sorting methods are divided into two classes: internal (array sorting) and external (sequential file sorting). The difference is that with internal sorting, all data is stored in the computer's RAM, while with external sorting, it is stored in external memory. Internal sorting offers more flexible options for constructing algorithms, since all data is arranged in the form of an array and appears to lie in front of user, He sees each element And has To him direct access. At external sorting access To The data is limited because it does not fit into the RAM due to its size .

Let's clarify mathematical production sorting tasks data[7].

Let us suppose that we need to order n elements $m_1, m_2, \ldots, m_n$, which we will call records. Each record m_j we will associate our key k_j, which will control process sorting. In addition to the key entry may contain and additional information, which Not influences on sorting , But Always

remains V this records. Task consists of *V location such permutations m* p_1, m_{p_2}, …,m_{p_n}, *records* m_1, m_2, …, m_n, *after which keys will be be located V non-decreasing* (*non-increasing) in order:*

$$k_{p_1} \le k_{p_2} \le ... \le k_{p_n} \quad (k_{p_1} \ge k_{p_2} \ge ... \ge k_{p_n})$$

A sorting is called **stable** if it satisfies an additional condition: records With identical keys remain V former in order, those. $p_i < p_j$, If $k_{p_i} = k_{p_j}$ And $i < j$.

Since each key identifies the corresponding record, these records may not be determined during sorting, considering only their keys (in the simplest case, when a record consists only of sortable elements **,** the concepts of record and key coincide).

When sorting internally, the method chosen should use the material sparingly. time processor operation And memory. Good ones algorithms it takes about *n* log *n time* to sort *n* records , and the measure of efficiency can be the number of required key comparisons and the number of record permutations. These numbers are functions from *n* - numbers sorted records.

When sorting arrays, we assume that permutations that move the array elements into the desired order must be performed *in the same place,* i.e., without using an intermediate array. Methods in which elements from array *A* are passed to the resulting array *B* are of significantly less interest.

9.6. Straight And fast methods internal sorting

All internal sorting methods are divided into direct and fast (or improved)[7].

Direct sorting methods require about n^2 key comparisons and are simpler and more convenient for explaining the characteristics of the underlying principles. majority sortings. Programs these methods easily understand and they are shorter than fast algorithm programs.

Improved sorting methods require a small number of operations, but these operations are usually more complex themselves, and so for sufficiently small *n,* direct methods are faster, although for larger values *of n* they should not be used.

Internal sorting methods can be divided into three classes according to the principles that define them:

1. **Insertion sort** : Elements are scanned one at a time, and each new element is inserted into the appropriate position among the previously ordered elements.
2. **Selection sort** : first, the smallest (or largest) element is selected and somehow separated from the rest, then the smallest (largest) of the remaining elements is selected, and so on.

3. **Sorting by swapping** : pairs of adjacent elements are examined sequentially; if the elements in a pair form an inversion (i.e. are not ordered), then they are swapped.

All direct sorting methods actually move each element of the array on each elementary step on one position. That's why require *O* (*n2*) such steps. Any improved methods are based on the principle of moving elements over large distances at each sorting step. This approach can significantly reduce sorting time, but complicate the execution algorithms.

Improved exchange method - Shaker sort, Hoare method (the most the fastest of the currently known internal sorting methods), an improvement of the direct inclusion sort - Shell method,
direct choice - sorting With help binary tree.

9.7. Sorting inserts

Let us consider the process of sorting by the insertion method (direct inclusion) of the array a[1], a[2],...,a[n]. First, an already sorted part, for example in non-decreasing order, a[1] ≤ a[2] ≤...≤ a[i-1] and an unsorted part a[i], a[i+l],..., a[n] are selected in the original array. Then, with step 1, starting with i = 2, alternating comparisons and moves of the element a[i], an index (key) j (1 ≤ j ≤ i -1) is determined in the sorted part of the array such that the inequalities a[jl] ≤ a[i] ≤ a[j] are satisfied. If such an index is found, the element a[i] is inserted in the corresponding place. To do this, the elements of the sorted part are shifted one position to the right: a[j], ..., a[i-1]. After this, the next value of i is moved. Otherwise, the next value of i is simply moved. With each step, the sorted portion of the array increases by one element. Clearly, a complete sort requires n - 1 steps, where n is the number of elements in the original array.

Let us demonstrate the process of insertion sorting using the example of an integer array a[1..12], consisting of the numbers 22, -14, 13, 28, 6, 41, -5, 18, 0, 35, 6, -15 (the inserted elements are marked with *, the elements of the sorted part are highlighted)

Source array:	**22**	- 14	13	28	6	41	- 5	18	0	35	6	- 15
1st step	**- 14** *	**22**	13	28	6	41	- 5	18	0	35	6	- 15
2nd step:	**- 14**	**13** *	**22**	28	6	41	- 5	18	0	35	6	- 15
3rd step:	**- 14**	**13**	**22**	**28** *	6	41	- 5	18	0	35	6	- 15
4th step:	**- 14**	**6** *	**13**	**22**	**28**	41	- 5	18	0	35	6	- 15
5th step:	**- 14**	**6**	**13**	**22**	**28**	**41** *	- 5	18	0	35	6	- 15
6th step:	**- 14**	**- 5** *	**6**	**13**	**22**	**28**	**41**	18	0	35	6	- 15
7th step:	**- 14**	**- 5**	**6**	**13**	**18** *	**22**	**28**	**41**	0	35	6	- 15
8th step:	**- 14**	**- 5**	**0** *	**6**	**13**	**18**	**22**	**28**	**41**	35	6	- 15
9th step:	**- 14**	**- 5**	**0**	**6**	**13**	**18**	**22**	**28**	**35** *	**41**	6	- 15
10th step:	**- 14**	**- 5**	**0**	**6**	**6** *	**13**	**18**	**22**	**28**	**35**	**41**	- 15
Step 11:	**- 15** *	**- 14**	**- 5**	**0**	**6**	**6**	**13**	**18**	**22**	**28**	**35**	**41**

As can be seen from For example, in the sorting process, at the first step, the first element original array counts sorted, A next for him

The element is compared with it. If this element is greater, it remains in its place; otherwise, the first element is shifted to the second position, and the second element is placed in the first position (as in our example). Next, all selected elements from the unsorted part of the array are sequentially compared with the elements of the sorted part, starting with the first, until a greater or equal element is encountered. This element and all subsequent elements of the sorted part are moved one position to the right (or down, if the array is arranged in a column), making room for the new element. This process ends when one of two conditions is met:

- the first element from the left a[j] ≥ a[i] is found, which indicates the need to insert a[i] between a[j] and a[j-1], in particular, if j=1, then a[i] is placed in the first position, which entails the need to increase the array index range by one;
- the right end of the sorted part of the array (the right barrier) has been reached, therefore the element a[i] should be left in its original place.

Note that comparing and moving an element a[i] in the sorted part of the array can be and conduct from right to left, starting from the (i-1)th element.

9.8. Sorting With help direct choice

Sorting array of size n in non-decreasing (non-increasing) order by the direct selection method is performed according to the following algorithm.

1. The element of the given array with the minimum (maximum) value (or key) is selected.
2. Selected element And first element are changing in places.
3. This process is repeated with the remaining n - 1 elements, then with n - 2 elements, etc., until there is only one, largest/smaller element left.

In total you will need n - Perform the specified sequence of actions once. During the sorting process, the sorted portion of the array will increase, and the unsorted portion will decrease accordingly.

Unlike insertion sort, where at each step only one element of the original array and **all** elements of the sorted part are considered, among which a place for insertion is sought, with direct selection, all elements of the original array are looked at to find one element with the smallest (largest) key, and the found element is placed as the next one in the sorted part of the array.

Let us remember that in all the sorting methods considered we do not use intermediate arrays.

Scheme sorting method direct choice let's illustrate on The example array discussed in the previous section. Elements that are swapped in the next step are highlighted in bold, and the minimum elements of the unsorted portion are also marked with an asterisk. The sorted portion is underlined.

Source array:	**22**	- 14	13	28	6	41	- 5	18	0	35	6	**- 15** *
1st step	<u>- 15</u>	**- 14** *	13	28	6	41	- 5	18	0	35	6	22

2nd step:	- 15	- 14	**13**	28	6	41	**- 5** *	18	0	35	6	22
3rd step:	- 15	- 14	- 5	**28**	6	41	13	18	**0** *	35	6	22
4th step:	- 15	- 14	- 5	0	**6** *	41	13	18	28	35	6	22
5th step:	- 15	- 14	- 5	0	6	**41**	13	18	28	35	**6** *	22
6th step:	- 15	- 14	- 5	0	6	6	**13** *	18	28	35	41	22
7th step:	- 15	- 14	- 5	0	6	6	13	**18** *	28	35	41	22
8th step:	- 15	- 14	- 5	0	6	6	13	18	**28**	35	41	**22** *
9th step:	- 15	- 14	- 5	0	6	6	13	18	22	**35**	41	**28** *
Step 10:	- 15	- 14	- 5	0	6	6	13	18	22	28	**41**	**35**
Step 11:	- 15	- 14	- 5	0	6	6	13	18	22	28	35	41

9.9. Sorting With help direct exchange

The algorithm of this sorting method consists of sequentially scanning the array from the beginning to the end and swapping *adjacent* elements, if they form *an inversion. Let's describe the* **non-decreasing** sorting algorithm in more detail.

Let an array a of size n be given. We begin scanning the array with the first pair of elements, a[1] and a[2]. If the first element of this pair is greater than the second, we swap them; otherwise, we leave them unchanged and compare the second pair of elements. a[2] And a[3]. If a|2] > a[3], then we change in their places And etc. On the first In this step, all pairs of array elements will be scanned: a[i] and a[i+1] for i from 1 to n - 1. As a result of this scan and the necessary exchanges, *the maximum* element will move to the end of the array and will be the sorted part. In the second step, a similar procedure is carried out from the first to (n - 1)-th element. This will move the second-largest element of the array to the second-to-last place. The sorted portion will contain two elements. These steps continue until the number of elements in the unsorted portion of the array is reduced to two. The final step is to sort the remaining two elements. After (n - 1) steps, the array will be sorted in **non-descending order.** For illustration purposes, we'll sort the array in non-descending order using the simple swap method. The sorted portion is highlighted in bold.

SourceArray:	22	- 14	13	28	6	41	- 5	18	0	35	6	- 15
1st step	- 14	13	22	6	28	- 5	18	0	35	6	- 15	**41**
2nd step:	- 14	13	6	22	- 5	18	0	28	6	- 15	**35**	**41**
3rd step:	- 14	6	13	- 5	18	0	22	6	- 15	**28**	**35**	**41**
4th step:	- 14	6	- 5	13	0	18	6	- 15	**22**	**28**	**35**	**41**
5th step:	- 14	- 5	6	0	13	6	- 15	**18**	**22**	**28**	**35**	**41**
6th step:	- 14	- 5	0	6	6	- 15	**13**	**18**	**22**	**28**	**35**	**41**
7th step:	- 14	- 5	0	6	- 15	**6**	**13**	**18**	**22**	**28**	**35**	**41**
8th step:	- 14	-5	0	- 15	**6**	**6**	**13**	**18**	**22**	**28**	**35**	**41**
9th step:	- 14	- 5	- 15	**0**	**6**	**6**	**13**	**18**	**22**	**28**	**35**	**41**
Step 10:	- 14	- 15	**- 5**	**0**	**6**	**6**	**13**	**18**	**22**	**28**	**35**	**41**
Step 11:	- 15	**- 14**	**- 5**	**0**	**6**	**6**	**13**	**18**	**22**	**28**	**35**	**41**

Sorting by the direct exchange method is also called the method "bubble" This name comes from the figurative interpretation, in which the elements of a given array are arranged vertically and, during the sorting **process** , at each step the lighter elements, like bubbles in a bath of water, rise to a level corresponding to their weight.

Laboratory work No. 9 Tasks sorting And search

PURPOSE OF THE WORK: Consolidation of theoretical knowledge and acquisition of practical skills skills By compilation algorithms And programs of various sorting methods.

EXECUTION WORKS: V in accordance With tasks points 1-4, considering option (p. 3), make up And implement program.

Exercise

1. Using a pseudo-random number generator, generate a linear integer array A[20] such that the array elements belong to the segment [-50, 50].

2. Execute V him linear search given element.

3. Sort the array using a specific method according to the option (see table).

Method sorting	Option
Insert	1, 4, 7, 10, 13, 16, 19, 22
Exchange	2, 5, 8, 11, 14, 17, 20, 23
Choice	3, 6, 9, 12, 15, 18, 21, 25

4. Execute binary search element.

Formation array, conclusion his on screen, linear search, Sorting and binary search should be formalized as corresponding functions.

Control questions

1. Concept difficulties algorithms.
2. On which classes are divided algorithms V in accordance With their temporal or spatial complexity?
3. Production tasks sorting data.
4. Straight And fast methods internal sorting.
5. Algorithm sorting array method inserts.
6. Algorithm sorting array method direct choice.
7. Algorithm sorting array method direct exchange.
8. Concept inversions.
9. Production tasks search element V array.
10. Algorithm sequential (linear) search.
11. Algorithm binary search.

10. Signposts And arrays

10.1. Concept static And dynamic variable

Previously, we discussed simple and structured data types, which are defined at the beginning of a program or subroutine and their structure remains unchanged throughout the entire program's execution. The compiler, after analysis, section variables takes away each variable corresponding The number of cells and fixes them for the entire duration of the block's execution. The size of the allocated space depends on the variable type. Such variables are created upon entering the block and cease to exist when the computational process exits the block. They do not change their size and are called static.

However, there is a class of problems in which memory must be allocated dynamically during the solution process. The most common are lists with an unknown number of objects. For such problems, dynamic variables are used. They are stored in a specific memory location, not designated by an identifier, but with a reference to this location. This type of access allows for dynamic allocation and deallocation of memory during program (block) execution. Dynamic variables are created using a reference data type. When a dynamic variable is created, a reference value is created—the address of the memory location where the newly created variable is located. variable.

10.2. Signposts

A pointer is a variable whose value is equal to the memory address of the value of another variable. In this sense, the name of the other variable refers to its value directly, while the pointer refers to its value indirectly. Referencing a value via a pointer is called indirect addressing.

Pointers, like any other variables, must be declared before they can be used. A pointer declaration looks like this:

```
type *ptr;
```

Where `type` - one from predetermined or certain user types, A
`ptr` - pointer. For example,

```
int *countPtr, count;
```

declares a variable `countPtr` of type **`int*`** (i.e., a pointer to an integer) and a variable `count` of type integer. The **`* symbol`** in the declaration refers only to `countPtr` . Every variable declared as a pointer must be preceded by an asterisk (**`*`**). If in the given example it is desired that the variable `count` was a pointer, it is necessary to write down:

```
int *countPtr, *count;
```

Symbol **`*`** V these records means operation indirect addressing. A pointer to
`void can also be declared :`

```
void *Pv;
```

This is a universal pointer to any data type. But before it can be used, it must be assigned a pointer value during operation. some specific data type. For example:

```
Pv = countPtr;
```

Pointers are widely used in C. They must be initialized either when declared or using the assignment operator. A pointer can be initialized with 0 or `NULL.` or address. Index with initial meaning 0 or `NULL` no what not points to. `NULL` is a symbolic constant defined to indicate that a given pointer points to nothing. An example of declaring a pointer with its initialization:

```
int *countPtr = NULL;
```

To assign the address of a variable to a pointer, the addressing operator `& is used` , which returns the address of its operand. For example, if there are declarations

```
int Step = 5;
int *ptr, Num;
```

then the operator

```
ptr = &Step;
```

appropriates address variable `Step` pointer `ptr` .

To get the value pointed to by a pointer, use operation `*` , usually called operation indirect addressing or the dereference operator. It returns the value of the object pointed to by its operand (i.e., the pointer). For example, continuing the example above, the operator

```
Num = *ptr;
```

assign to variable `Num` value 5, i.e. the value of the `Step variable` pointed to by `ptr` .

Operation dereferences it is forbidden apply To pointer on `void,` because For him unknown, Which size memory necessary dereference.

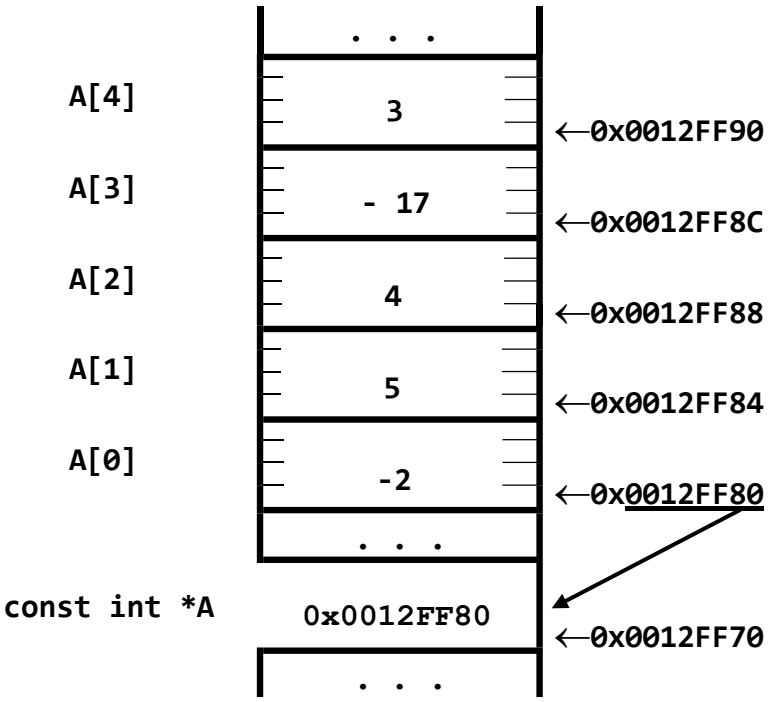
Here, it's important to note the different interpretations of the " * " symbol. If the " * " symbol appears before a variable identifier when declaring it, the meaning of this "asterisk" is to indicate that a pointer variable, not a regular variable, is being declared. In all other cases, the " * " symbol is read and interpreted by the program as "taking a value by address," i.e., an indirect addressing operation.

10.3. Interrelationship between arrays And pointers

Arrays and pointers in C are closely related and can be used almost interchangeably. An array name can be thought of as a constant pointer to the first element. array. His difference from regular pointer only in volume, What it can't be modify. For example, asked integer array `A` consisting of 5 elements:

```
int A[5]={-2,5,4,- 17,3};
```

Let's see, How this array will be accommodated V operational memory (Fig. 10.1).



Drawing 10.1 - Performance array elements V memory computer

It's worth noting that four 8-bit memory cells are allocated to store the value of each array element. Figure 10.1 on the right shows a sequence of memory cell addresses containing values that differ by four bytes. This is because the element type of this array is **int** . Moreover, four bytes are also allocated to store the address of the beginning of the array, since all pointers for 32-bit processors are 4-byte.

Pointers can be used to perform any operation, including indexing an array. Consider the following declaration:

```
int b[5] = {1,2,3,4,5}, *Pt;
```

Here an array of integers `b[5]` and a pointer to an integer `Pt are declared` . Since the array name is a pointer to the first element of the array, then You can assign the pointer `Pt` the address of the first element of the array `b` using the operator `Pt = b;`

This is equivalent to assigning the address of the first element of the array as follows: `Pt = &b[0];`

Now Can refer on element array `b[3]` With help expressions `*(Pt + 3).`

Pointers can be indexed in the same way as arrays. For example, the expression `Pt[3]` refers to the array element `b[3].`

Or, for example, the definition **`int`** `a[5];` defines an array of five elements `a[0], a[1], a[2], a[3], a[4].` If the object `*y` defined as

`int` `*at;` That operator `at = &a[0];`

assigns the address of the element `a[0] to the variable y` . If the variable `y` points to the next element of array `a` , then `y+1` points to the next element, and here is being carried out corresponding scaling For increments addresses With

taking into account the length of the object (for **the int type** – 4 bytes, **long** – 4 bytes, **double** – 8 bytes and etc.). So How itself Name array There is address his zero element, That operator `y = &a[0];` can be written in another form: `at = a` . Then the element `a[1]` can be represented as `*(a+1)` . On the other hand, if `y is` a pointer to an array `a` , then the following two entries: `a[i]` and `*(y+i)` are equivalent. Let's consider an example:

```
intmain()
{
    inta[5]={-5,0,34,12,-17};
    cout<<endl<<(a+2);  // оператор 1
    cout<<endl<<&a[2];  // оператор 2
    cout<<endl<<*(a+2); // оператор 3
    cout<<endl<<a[2];  // оператор 4
    cout<<endl;
    return 0;
}
```

So, operators 1 And 2 will be taken out one And That same meaning - address element

array With index 2. A operators 3 and 4 - one And That same meaning 34.

There is one important difference between the array name and the corresponding pointer. A pointer is a variable, and `y = a` or `y++` are valid operations. The array name is a constant, so constructions like `a = y` , `a++` cannot be used because the value of the constant is permanent and cannot be changed.

Pointers can be used as operands in arithmetic expressions, assignment expressions, and comparison expressions. However, not all operations commonly used in these expressions are permitted on pointer variables.

A limited number of arithmetic operations can be performed on pointers. A pointer can be incremented (++), decremented (--), and integers can be added to a pointer (+ or +=), subtract from it integers (- or -=) or subtract one pointer from another.

Adding pointers to integers differs from regular arithmetic. Adding 1 to a pointer means shifting it by the number of bytes contained in the variable it pointed to. Typically, such operations are applied to pointers to arrays. Continuing with the example above, where the pointer `Pt the value b` was assigned – a pointer to the first element of the array, then after executing the operator

```
Pt += 2;
```

`Pt` will point to the third element of the `b array` . The actual value of the `Pt pointer` will change to the number of bytes occupied by one element of the array, multiplied by 2. For example, if each element of the `b array` occupies 2 bytes, then the value of `Pt` (i.e. the memory address pointed to by `Pt`) will increase by 4.

Similar rules apply when subtracting an integer value from a pointer.

Pointer variables can be subtracted from each other. For example, if `Pt` points to to the first element of the array `b` , and the pointer `Pt1` – on the third, then the result of the expression `Pt1 - Pt` will be equal to 2 – the difference between the indices of the elements pointed to these pointers. AND So will, despite on That, What addresses, contained in these pointers, differ by 4 (if the array element occupies 2 bytes).

Pointer arithmetic is meaningless unless it's performed on pointers to an array. Comparing pointers with the >, <, >=, and <= operators also only makes sense for pointers to the same array. However, the relational operators == and != make sense for any pointers. Pointers are considered equal if they point to the same memory address.

10.4. Order announcements dynamic arrays

Dynamic arrays are created using the `new operation` , and the type and dimensions must be specified, for example:

```
int a=10;
int *ptr=new int [a];
```

This line creates a pointer variable to **`an int`** , allocates a contiguous area of dynamic memory large enough to hold 10 integer elements, and stores its start address in the `ptr pointer` . Dynamic arrays cannot be initialized upon creation and are not zeroed out.

Access to elements of a dynamic array is carried out in the same way as to static ones, for example, element number 5 can be accessed as `ptr[5]` or `*(ptr+5)` .

An alternative way to create a dynamic array is to use the `malloc function.` C libraries:

```
int a=10;
int *ptr=( int *)malloc(a* sizeof ( int ));
```

Operation transformations like, recorded before appeal To `malloc` functions , required That's why, What function returns meaning pointer type **`void`** `*` . Memory, reserved under dynamic array With help `new[]` , should to free oneself `delete` operator `[]` , A memory, dedicated function `malloc` - by means of functions `free` , For example:

```
delete [] ptr;
free(ptr);
```

The array size is not specified in the `delete operation` , but the square brackets are required.

10.5. Broadcast arrays V quality parameters functions

When using an array as a parameter, a pointer to its first element is passed to the function; in other words, the array is always passed by address. At this information O quantity elements array is lost, And its dimension should be passed through a separate parameter (in the case of an array of characters, that is, a string, its actual length can be determined by the value of the null character):

```
#include<iostream>

int sum(const int*mas, const int n);
int const n = 10;
```

```
int main()
  {
     int marks[n]=  {3, 4, 5, 4, 4};
     cout<<”сумма элементов массива: “<<sum(marks,n);
     return 0;
  }
int sum( constint *mas, constint n) //(the value n must be a constant)
  {
     int s =0;
     for ( int i=0; i<n; i++) s += mas[i];
     return s;
  }
```

At transfer multidimensional arrays All dimensions, If They Not known on At the compilation stage, they must be passed as parameters. Inside the function, the array is interpreted as one-dimensional, and its index is recalculated in the program. In the example below, the function calculates the sum of the elements of two two-dimensional arrays. The dimension of the array is `b` known at compile time, under array `a` memory is allocated dynamically.

```
#include<conio.h>
#include<iostream>
#include<stdio.h>
#include<stdlib.h>
usingnamespace std;
int sum(const int *a, const int nstr, const int nstb);
int main()
  {
    int b[2][2] = {{2,2},{4,3}};
      printf(сумма элементов b: %d\n”,sum(&b[0][0],2,2));
    //the array name cannot be passed to sum due to a type mismatch
    int i,j, nstr, nstb,*a;
    printf(“enter quantity lines And columns: \n");
    scanf("%d%d",&nstr, &nstb);
    //forming a matrix: allocating memory a=(
    int *)malloc(nstr *nstb* sizeof ( int ));
    // input array elements
    for (i=0; i<nstr; i++)
     for (j=0; j<nstb; j++) scanf(“%d”,&a[i * nstb+j]);
    // display the result of the sum function printf(“sum
    elements a: %d\n”, sum(a,nstr,nstb));
    //release memory
    free(a);
      _getch();
    return 0;
  }
int sum(const int *a,const int nstr, const int nstb)
  {
```

```
    int i,j,s=0;
    for (i=0;i<nstr; i++)
     for (j=0;j<nstb; j++) s+=a[i * nstb + j];
    returns;
  }
```

For Togo to work With two-dimensional array natural in this way, Can apply alternative way discharge memory:

```
#include<conio.h>
#include<iostream>
#include<stdio.h>
#include<stdlib.h>

usingnamespace std;
int sum(int **a, constint nstr, constint nstb);

int main()
{
int nstr, nstb;
  cin>>nstr>>nstb;
int **a, i, j;

//formation matrices: selection memory a
= newint * [nstr]; // nstrstringsinarray
for (i = 0; i < nstr; i++)
        a[i] = newint [nstb]; //and nstb columns

// input array elements
for (i=0; i<nstr; i++)
     for (j=0; j<nstb; j++) cin>>a[i][j];

//conclusion on screen result work functions
sum cout<<sum(a, nstr, nstb);

//release memory diverted under two-dimensional dynamic array: for
(i = 0; i<nstr; i++) //where nstr is the number of strings in the
array delete [] a[i]; // deleted the array of pointers
      delete [] a; // deleted the pointer to the array of pointers

_getch();
return 0;
}

int sum(int **a, constint nstr, constint nstb)

  {
int i, j, s=0;
for(i=0; i<nstr; i++)
     for(j=0; j<nstb; j++) s+=a[i][j];
return s;
  }
```

In this case, memory is allocated in two stages: first for a column of pointers on lines matrices, A then V cycle under every line. Memory release must be performed in reverse order.

Laboratory Job No. 10
Application arrays And pointers For solutions applied tasks

TARGET WORKS: acquisition skills organizations input-output And array processing using pointers and functions.

EXECUTION WORKS: V in accordance With option make up And implement program.

Task I

Study order descriptions And use pointers:

- dial And debug next program:

```
int main()
{
   int a=7, b=2;
   int *ptr=NULL;
   cout<<"\na="<<a<<", &a="<<&a;
   cout<<", b="<<b<<", &b="<<&b;
   ptr=&a;
   cout<<"\nДействие: ptr=&a. Результат: ptr="<<ptr<<", *ptr="<<*ptr;
   ptr=&b;
   cout<<"\nДействие: ptr=&b. Результат: ptr="<<ptr<<", *ptr="<<*ptr;
   *ptr=137;
   cout<<"\nДействие: *ptr=137. Результат: b="<<b;
   ptr++;//или ptr+=2;
   *ptr=-105;
   cout<<"\nAction: ptr++, *ptr=-105. Result: ptr="
   <<ptr<<", *ptr="<<*ptr<<endl;
   return 0;
}
```

- present result And be able to explain work programs.

Exercise II

Implement using function pointers and mechanism in in accordance With number options:

1. IN one-dimensional array, consisting from *n* material elements, calculate:
   - amount negative elements array;
   - work elements array, located between maximum And minimal elements.
2. IN one-dimensional array, consisting from *n* material elements, calculate:
   - amount positive elements array;
   - work elements array, located between maximum modulo And minimal By module elements.
3. IN one-dimensional array, consisting from *n* whole elements, calculate:
   - amount elements array, located between first And the last zero elements;
   - transform array like this in this way, to at first were located All

positive elements, A Then - All negative (elements, equal 0, consider positive); do not change the order of the array elements in the selected groups.

4. IN one-dimensional array, consisting from *n* material elements, calculate:
   - amount array elements With odd numbers;
   - compress array, having deleted from him All elements, module which Not exceeds 1.

5. IN one-dimensional array, consisting from *n* material elements, calculate:
   - maximum element array;
   - the sum of the array elements located before the last positive element.

6. IN one-dimensional array, consisting from *n* material elements, calculate:
   - minimum element array;
   - the sum of the array elements located between the first and last positive elements.

7. IN one-dimensional array, consisting from *n* whole elements, calculate:
   - number maximum element array;
   - transform the array so that the first half contains elements in odd positions, and the second half contains elements in odd positions. - elements, standing V even positions.

8. IN one-dimensional array, consisting from *n* material elements, calculate:
   - the sum of the array elements located between the first and second negative elements.
   - transform the array so that all elements are located first, module which Not exceeds 1, A Then - All rest; order of succession elements array V allocated groups Not change.

9. IN one-dimensional array, consisting from *n* material elements, calculate:
   - maximum by module element array;
   - transform the array so that the elements equal to zero are located after everyone the rest.

10. IN one-dimensional array, consisting from *n* whole elements, calculate:
    - minimum By module element array;
    - amount modules elements array, located after first element equal to zero.

11. IN one-dimensional array, consisting from *n* material elements, calculate:
    - number minimum By module element array;
    - the sum of the moduli of the array elements located after the first negative element.

12. IN one-dimensional array, consisting from *n* material elements, calculate:
    - number maximum By module element array;
    - the sum of the array elements located after the first positive element.

13. IN one-dimensional array, consisting from *n* material elements, calculate:
    - quantity elements array, lying V range from *A* to *IN* ;

- amount elements array, located after maximum element.

14. IN one-dimensional array, consisting from *n* material elements, calculate:
    - quantity elements array, equal 0;
    - amount array elements, located after minimum element.

15. IN one-dimensional array, consisting from *n* material elements, calculate:
    - quantity elements array, large *WITH* ;
    - transform the array so that all negative ones are located first elements, A Then - All positive (elements, equal 0, consider positive); the order of the array elements in the selected groups should not be changed.

16. IN one-dimensional array, consisting from *n* whole elements, calculate:
    - the sum of the array elements located between the first and last zero elements.
    - transform the array so that all positive ones are located first elements, A Then - All negative (elements, equal 0, consider positive); do not change the order of the array elements in the selected groups.

17. IN one-dimensional array, consisting from n material elements, calculate:
    - amount array elements With odd numbers;
    - compress array, having deleted from him All elements, module which Not exceeds 1. Liberated V at the end array elements fill out zeros.

18. IN one-dimensional array, consisting from n material elements, calculate:
    - maximum element array;
    - amount elements array, located to last positive element .

19. IN one-dimensional array, consisting from n material elements, calculate:
    - minimum element array;
    - the sum of the array elements located between the first and last positive elements.

20. IN one-dimensional array, consisting from n whole elements, calculate:
    - number maximum element array;
    - transform the array so that its first half contains elements, standing V odd positions, A in second half – elements, standing V even positions.

21. IN one-dimensional array, consisting from n material elements, calculate:
    - the sum of the array elements located between the first and second negative elements.
    - transform the array so that all elements are located first, module which Not exceeds 1, A Then - All rest.

22. IN one-dimensional array, consisting from n material elements, calculate:
    - maximum by module element array;
    - transform the array so that the elements equal to zero are located before by everyone the rest elements.

23. IN one-dimensional array, consisting from n whole elements, calculate:
- minimum By module element array;
- amount modules elements array, located after the first element equal to zero.

24. IN one-dimensional array, consisting from n material elements, calculate:
- number minimum By module element array;
- the sum of the moduli of the array elements located after the first negative element.

25. IN one-dimensional array, consisting from n material elements, calculate:
- number maximum By module of the element array;
- amount elements array, located after first positive element.

Exercise III

In accordance with the option, using pointers and user-defined functions, compose and implement the program:

Elements of a real one-dimensional dynamic array of size x $n > 20$ are generated by a pseudo-random number generator in the range [-10, 10]. Generate the array y using the formula and perform the specified calculations. Write the formula using mathematical symbols in a familiar form. Display the original array and the results of the problem solution. Round all numbers to hundredths. Analyze the results using well-known mathematical packages.

1. $y[i] = \exp(x[i] - i)$, $i = 0, 1, ..., n-1$, $n = 2k$. Calculate amount works $P = x[0]$ *and* $[2k-1] + x[1]$ *and* $[2k-2] + ... + x[2k-1]$ *and* $[0]$.

2. $y[i] = x[i] / \ln(\text{abs}(x[i] + 1.23))$, $i = 0, 1, ..., n-1$. Subtract the sum of the negative values of the elements of array x from the first 15 elements of array y.

3. $y[i] = \exp(x[i]) / x[i]$, $i = 1, 2, ..., n$. Calculate the sum of all products of the corresponding elements of the arrays x and y.

4. $y[i] = \ln(x[i] + 10.4) / (x[i] + 1)$, $i = 0, 1, ..., n-1$. Subtract amount positive values of the elements of the array x from the last 12 elements of the array y.

5. $y[i] = \cos(x[i]) + \sin(x[i]) + 2.6$, $i = 0, 1, ..., n-1$. Calculate amount squares of the elements of the array y, standing in odd places.

6. $n = 2k$, $y[0] = x[1]$, *and* $[1] = x[1]\,x[3]$, $y[2] = x[1]\,x[3]\,x[5]$, ..., $y[k-1] = x[1]\,x[3]\,x[5]\ ...\ x[n-1]$, $y[k] = x[0]$, $y[k+1] = x[0]\,x[2]$, $y[k+2] = x[0]\,x[2]\,x[4]$, ..., $y[n-1] = x[0]\,x[1]\ ...\ x[n-2]$.

   Subtract from amounts first k elements array *at* amount latest k its elements.

7. $y[i] = \text{sqrt}(\text{abs}(x[i]))$, $i = 0, 1, ..., n-1$. Calculate the number of elements of the array y that are greater than the arithmetic mean of the elements of the array x.

8. *in* $[0] = 5.2$, *in* $[1] = 4.3$, *in* $[2] = 0.09$, *in* $[3] = -0.5$, *in* $[4] = 7.15$; *and* $[i] = \cos(\textit{and}[i-1] + \textit{and}[i-2] + \textit{and}[i-3]) / (\text{and}[i-4] \cdot \textit{and}[\textit{Yo} - 5])$, $i = 5, 6, ..., n-1$.

Calculate work everyone negative numbers array *at* (V t h e r e i s n o array *x in this problem*).

9. $n = 2k$, *and [* 0] = *x* [0] *x* [1], *y* [1] = *x* [2] *x* [3], ..., *and* [*k* -1] = *x* [*n* - 2] *x* [*n* - 1].
Calculate amount squares elements array *at* .

10. $y[i] = x[i]^{2/3}$, $i = 0, 1, ..., n-1$. Sum the elements of array *y in* increments of 3, starting from the first element.

11. $n = 2k$, $y[i] = \text{sqrt}(\text{abs}(\cos(x[i]) + 6.7))$, $i = 0, 1, ..., n-1$.
Calculate the sum works $S = x[0]\,y[1] + x[2]\,at[3] + ... + x[n-2]\,y[n-1]$.

12. $y[i] = i \sin(x[i])$, $i = 0, 1, ..., n-1$. Subtract the product of the positive values of the *y array* from the last 14 elements of the *x array* .

13. $y[i] = \sin(\exp(x[i]))$, $i = 0, 1, ..., n-1$. Calculate quantity negative elements of the array *y* with odd indices.

14. $y[0] = \text{sqrt}(\text{abs}(x[0] - x[1]))$, $y[1] = \text{sqrt}(\text{abs}(x[1] - x[2]))$, ...,
$y[n-3] = \text{sqrt}(\text{abs}(x[n-3] - x[n-2]))$, $y[n-2] = \text{sqrt}(\text{abs}(x[n-2] - x[n-1]))$. Sum the elements of array *y* that have an index multiple of 3.

15. $at[0] = x[0]$, $y[1] = x[0] + x[1]$, ..., $y[n-1] = x[0] + x[1] + ... + x[n-1]$.
Calculate
amount cubes negative elements array *at* .

16. $n = 2k$, $y[0] = x[0] - x[n-1]$, $y[1] = x[1] - x[n-2]$, ..., $y[k-1] = x[k-1] - x[k]$. From
each element array *at* subtract average arithmetic elements of array *x* .

17. $y[i] = \exp(\cos(x[i]))$, $i = 0, 1, ..., n-1$. Calculate quantity elements of the array *y* that fall within the interval [0, 1].

18. $y[i] = 1/8\text{tg}(x[i])$, $i = 0, 1, ..., n-1$. Calculate quantity negative elements of array *x* and the sum of the inverse elements of array *y* .

19. $at[0] = x[0]$, $y[n-1] = x[n-1]$, $y[i] = (x[i-1] + x[i] + x[i+1]) / 3$. Calculate the number of positive elements of the array *y* with even indices and find their sum.

20. $y[i] = \exp(x[i]) \sin(x[i])$, $i = 0, 1, ..., n-1$. Calculate amount elements array *at* , standing on even places.

21. $y[i] = \text{tg}(2^{x[i]})$, $i = 0, 1, ..., n-1$. Calculate quantity elements array *y* , smaller in their values than the corresponding elements of the array *x* .

22. $y[i] = 3^{\cos(x[i])}$, $i = 0, 1, ..., n-1$. Calculate the number of elements of the array *y* that do not fall within the interval [0, 1].

23. $y[i] = x[i]^2 \sin(1/x[i])$, $i = 0, 1, ..., n-1$. Subtract the sum of positive values from the first 14 elements of the array *y* .

24. $y[i] = \cos(x[i]) - \text{abs}(x[i])$, $i = 0, 1, ..., n-1$. Subtract the sum of the squares of the elements of the array *y* from the sum of the squares of the elements of the array *x* .

25. *in* [0] = 4.5, *in* [1] = -3.6, *in* [2] = -8.2, *in* [3] = 6;
y [*i* = ln(abs(*y* [*i* - 1])) / sin(*y* [*i* - 2] / (*y* [*i* - 4] y[i - 3])), *and* = 4, 5, ..., *n* - 1(where

array *X* No, and the elements of the array *y* are calculated with help recurrence relations). Calculate the sum of the negative elements of the array *y* .

Control questions

1. What is this a pointer?
2. What general type of advertisement pointer? Bring example.
3. How describe some pointers on data one And Togo same like?
4. What are peculiarities initialization pointers?
5. What means `NULL` ?
6. Give an example of using the addressing operation. Explain the principle of the illustrated operation.
7. Give an example of using the "indirect addressing" operation (or dereferencing). Explain the principle behind the illustrated operation.
8. Peculiarities use symbol "*".
9. What is it? relationship between name array And pointer?
10. How ask pointer address first element array.
11. How to get the value of `the i` -th element of an array using a pointer?
12. What is it like? difference between name array And appropriate pointer?
13. What is the order of declaring dynamic arrays using the `new` operator ? Using the `malloc function` ?
14. How free memory, allotted under dynamic array?
15. What are the basic rules for passing arrays as function parameters ?

Example execution tasks III laboratory work

Task: *In accordance with the option, using pointers and user-defined functions, write and implement the following program:*

Elements of a real one-dimensional dynamic array of size *x* $n > 20$ are generated by a pseudo-random number generator in the range [-10, 10]. Generate the array *y* using the formula and perform the specified calculations. Write the formula using mathematical symbols in a familiar form. Display the original array and the results of the problem solution. Round all numbers to hundredths. Analyze the results using well-known mathematical packages.

$y[i] = (15+\exp(x[i] - i^2))/(i^2+1)$, $i = 0, 1, ..., n - 1$.

Calculate amount works elements given arrays:

$S = x[0]\, x[1] \ldots x[n - 1] + y[0]\, y[1] \ldots y[n - 1]$.

Solution

1. Mathematical model

Elements material one-dimensional dynamic array *X* size $n > 20$ let's generate sensor pseudo-random numbers V range [-10, 10].

For location elements we will use the *y* array formula:

$$y_i = \frac{15 + e^{(x_i - i^2)}}{i^2 + 1}.$$

We organize the array output using an auxiliary function that uses formatted data output with rounding up to hundredths.

Required find amount works elements given arrays:

$$S = \sum_{i=0}^{n-1} x_i + \sum_{i=0}^{n-1} y_i.$$

We will formulate the finding of the product of array elements as an auxiliary function.

Arguments: real one-dimensional dynamic array `x` ; integer `n` - quantity elements array.

Results: real one-dimensional dynamic array `y` ; sum of products of elements of given arrays `S` (double-precision real type).

Intermediate values: counter `i` (of integer type), the name of the array `lit` (of character type), the product of the elements of the given arrays `p` (of double-precision real type).

4. The result of the program (see the algorithm and program listing on the next page) **:**

```
Enter the number of array elements n = 21

x:

- 8.42  9.06   7:00   10.45   4.15  - 6.26   8.49   9.48   0.98   4.11
- 6.55 - 5.25  0.95   1.95  -6.64   -2.19   6.58   6.67  -2.11   -7.30
 5.56

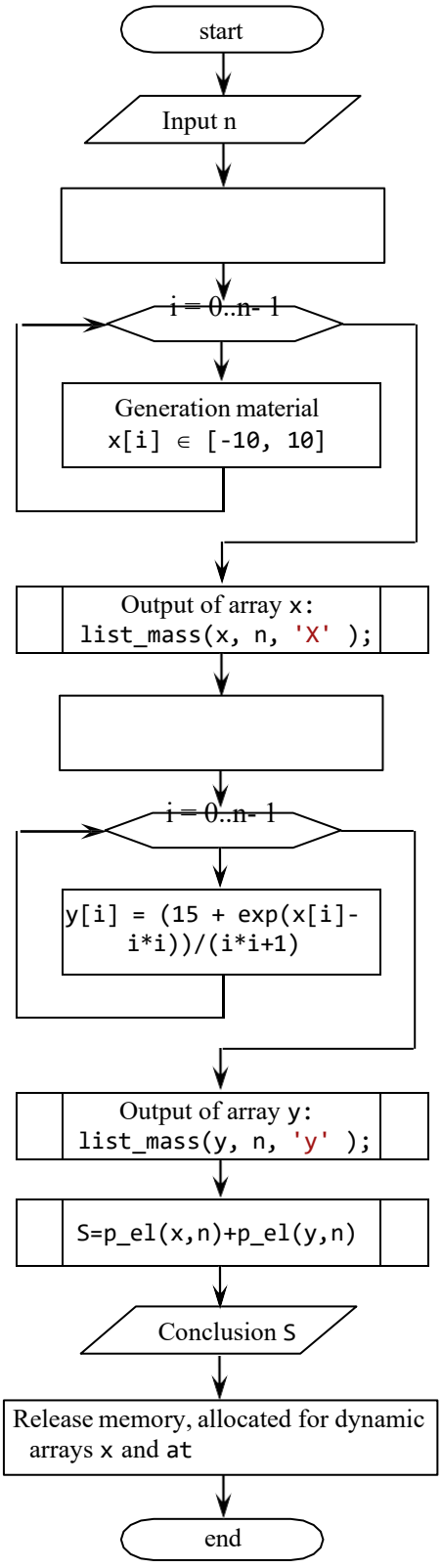
y:

15.00   1590.15  7.02   1.93   0.88   0.58    0.41   0.30   0.23   0.18
 0.15   0.12   0.10   0.09    0.08   0.07    0.06   0.05   0.05    0.04
 0.04

 S = 163630350633731.37
```

5. Analysis results With using famous mathematical packages to complete independently.

2. Algorithm



The function `double p_el( float *a, int np)` calculates work elements array.

The function `void list_mass( float *a, int nl, char lit)` implements formatted displaying a one-dimensional array on the screen

3. Program

```
#include<conio.h>
#include<iomanip>
#include<iostream>
#include<stdio.h>
#include<time.h>     // для time(0)
#include<stdlib.h>  // для srand() и rand()

usingnamespace std;

double p_el(float *a, int ns);
void list_mass( float *a, int nl, char lit);

int main()
  {
       srand(time(0));
      cout<< "Enter quantity array
      elements n = " ;
       int n;
       cin>>n;
       //forming array x: float
       *x = newfloat [n]; for (
       int i = 0; i < n; i++)

x[i] =  ( float )(rand()%2100- 1000)/100;




       list_mass(x, n, 'x' ); //output in
       arrays
       //formation massive in:
       float *y = newfloat [n];
       for ( int i=0; i<n; i++)
              y[i] = (15 + exp(x[i]-
            i*i))/(i*i+1);
       list_mass(y, n, 'y' ); //output array y
      double S = p_el(x,n) + p_el(y,n); cout<<
      "\n S = " << fixed << setprecision(2)
      <<S;
      //release memory, diverted for
      dynamic arrays:
       delete [] y;
       delete [] x;
       _getch();
       return 0;
  }

double p_el(float *a, int np)
  {
double p = 1;
for(int i=0; i<np; i++)
       p*=a[i];
return p;
  }

void list_mass(float *a, int nl, char lit)
{
       cout<<"\n"<<lit<<":\n "<<endl;
       for(int i=0; i<nl; i++)
              printf ("%5.2f\t",a[i]);
       cout<<"\n";
}
```

11. Peculiarities work with functions

11.1. Methods transmissions parameters V function

All variables declared in the function body without specifying a memory class have the **auto memory class** , meaning they are local. When the function is called, local variables are allocated memory on the stack and initialized. Control is transferred to the first statement in the function body, and execution of the function begins, continuing until a **return statement is encountered.** or the last statement of the function body. Control then returns to the point following the call point, and local variables become inaccessible. When the function is called again, local variables memory is allocated again, and therefore the old values of local variables are lost.

Function parameters are passed by value and can be considered as local variables for which memory is allocated when the function is called and initialized with the values of the actual parameters. On exit The values of these variables are lost from the function. Since parameters are passed by value, the values of variables that are actual parameters in the calling function cannot be changed within the function body. However, if a pointer to a variable is passed as a parameter, the value of that variable can be changed using the dereference operator.

Пример.

```
        /*    использование параметров по значению  */
void change (int x, int y);
void main()
  {
     int a=5,b=15;
       cout<<"\na="<<a<<", b="<<b;
       change(a,b);
       cout<<"\na="<<a<<", b="<<b;
     return;
  }
void change (int x, int y)
{
int
k=x;
x=y;
y=k;
  }
```

In this function the values of the variables `x` and `y` , which are formal parameters, are swapped, but since these variables exist only inside the `change function` , the values of the actual parameters used in when calling a function, will remain unchanged, that is, the values of the variables `a` and `b` as there were 5 And 15 respectively, So and remained. For Togo so that they change To replace the values of the actual arguments, you can use the function shown in the following example.

Example.

```
        /*    using parameters at address  */
void change ( int *x, int *y)
{  int k=*x;
             *x=*y;
             *y=k;}
```

When calling such a function, the actual parameters should be not the values of the variables, but their addresses. `change (&a,&b);.` Then the function will operate not with local values, but with the values at the addresses passed to the function as parameters, which will lead to an exchange of the values of the variables `a` and `b` .

There's another way to pass parameters to a function: by reference. Using references is very convenient and easy to use. Let's look at the same example, but using references.

Example.

```
        /*    use parameters by reference  */
       void change ( int &x, int &y)
{    int
k=x; x=y;
y=k;}
```

When calling such a function, the variable names must be used as actual parameters: `change (a,b);.` In this function, we will work with the variables themselves `a` and `b` , but through their alternative names: `x` and `y .`

Any function can return a value as a result of its execution. A function cannot return an array or a function, but it can return a pointer to any type, including an array or a function. The return type specified in the function definition must match the type in the function declaration.

A function returns a value if its execution ends with a `return statement` containing an expression. The specified expression is evaluated, converted if necessary to the return type, and returned to the caller as the function's result. If the `return statement` does not contain an expression, or if the function's execution ends after executing its last statement (without executing the `return statement`), the return value is undefined. For functions that do not use a return value, `the void type must be used, indicating the absence of a return value. If a function is defined as returning a value and the return` statement does not contain an expression upon exiting, the behavior of the calling function after control is transferred to it may be unpredictable.

11.2. Broadcast names functions V quality parameters

A function can be called through a pointer to it. To do this, a pointer of the appropriate type is declared and the function's address is assigned to it using the address-of operator:

```
void f( int a){ /*…*/ } // definition of a function
void (*pf)( int );       // function pointer
          …
pf=&f;           //the pointer is assigned the address of the function
                 //(you can write pf=f;)
pf(10);          //function f is called through a pointer pf
                 //(you can write (*pf)(10) )
```

To make programs easier to read, typedefs are used when declaring function pointers **.** Arrays of function pointers can be declared (this can be useful, for example, when implementing a menu):
//Description type PF How pointer
//on function With one parameter type **int** :

```
typedef void (*PF)( int );
```

//Description And initialization array pointers:

```
PF menu[]= {&new, &open, save};
```

// Call functions open

```
menu[1](10);
```

Here , **new** , **open** , and **save** are the names of functions that must be declared previously. Function pointers are passed into the program in the same way as parameters of other types:

```
# include <iostream.h>
typedef void (*PF)( int );
void f1(PFpf)
{     //function f1 receives V quality parameter pointer type PF
pf(5) //call the function passed via the pointer
}
void f( int i ) {cout <<i}
int main()
{
  f1(f);
return 0;
}
```

The type of a pointer and the type of the function called through that pointer must match exactly.

11.3. Overload functions

Often It happens comfortable, so that the functions, implementing one and That same Algorithms for different data types have the same name. If this name is mnemonic, that is, it conveys the necessary information, it makes the program more understandable, since only one name needs to be remembered for each action.

Using multiple functions with the same name but different parameter types is called **function overloading** .

The compiler determines which function needs to be called by type actual parameters. This process it's called **resolution**

Overloading (translated from the English word "resolution" as "clarification"). The return type of the function is not involved in the resolution. The resolution mechanism is based on a fairly complex set of rules, the essence of which boils down to using the function with the most suitable arguments and returning a message if one is not found. Let's say there are four variants of the function that determine the largest value:

```
// Returns the larger of two integers:
int max( int , int );
// Returns the longest substring :
char * max( char *, char *);
// Returns the largest of the first parameter and the length of the second:
int max ( int , char *);
// Returns the largest of the second parameter and the length of the first:
int max (char*, int);
void f(int a, int b, char* c, char* d)
{
     cout << max (a,b) << max (c,d) << max (a,c) << max (c,b);
}
```

At call functions `max` compiler chooses corresponding type actual parameters of the function variant (in the given example, all four function variants will be called sequentially).

If no exact match is found, promotions of ordinal types are performed according to the general rules (see [2] p. 38 and Appendix 3), for example, **`bool`** and **`char`** to **`int`** , **`float`** and **`double`** , etc. Then standard type conversions are performed, for example, **`int`** to **`double`** or pointers to **`void *`** . The next step is to perform user-specified type conversions (these conversions are discussed in second part [2], p. 195), as well as searching for matches due to a variable number of function arguments. If a match at the same stage can be obtained in more than one way, the call is considered ambiguous and an error message is issued.

Ambiguity Maybe appear at:

- transformation type;
- use parameters- links;
- use arguments By default.

Below given rules descriptions overloaded functions.

- Overloaded functions must be in the same scope, otherwise hiding will occur similar to identical variable names in nested blocks.
- Overloaded functions can have default parameters, but the values of the same parameter in different functions must match. Different versions of overloaded functions may have different numbers of default parameters.

Functions cannot be overloaded if their parameter declarations differ only by the **`const modifier.`** or using a reference (for example, **`int`** and **`constint`** or **`int &`**).

Laboratory work No. 11 Research ways work With functions

TARGET WORK: Improving skills in programming using pointers, exploring ways to pass parameters to a function and the order of returning parameters from a function

EXECUTION WORKS: master theoretical material, execute general For everyone exercise I And V in accordance With option to create a program (task II).

Exercise I

Research mechanism overloads And methods transmissions parameters at implementation of user-defined functions

- Implement V environment programming listings 1- 3.
- Analyze process execution programs. Convert attention on the method and location of storing the values of local variables **a, b** and **c** .
- Make written conclusions on the considered methods of passing parameters to functions 1-3.

1. Study method transmissions parameters V function By meaning

```
void Swap( int a, int b);

int main(){

      int x,y;
      cout<<"\nEnter two numbers: ";
      cin>>x>>y;
      cout<<"\nx="<<x<<", y="<<y<<endl;
      Swap(x,y);
   cout<<"\nx="<<x<<", y="<<y<<endl;
      return 0;
}
void Swap(int a, int b){
      int c;
      c=a;
      a=b;
      b=c;
      return;
}
```

2. Study method transmissions parameters V function By address

```
void Swap( int *a, int *b);

int main(){

      i n t x,y;
      cout<<"\nEnter two numbers: ";
      cin>>x>>y;
      cout<<"\nx="<<x<<", y="<<y<<endl;
      Swap(&x,&y);
      cout<<"\nx="<<x<<", y="<<y<<endl;
```

```
      return 0;
}
void Swap(int *a, int *b){
      int c;
      c=*a;
      *a=*b;
      *b=c;
      return;
}
```

3. Study method transmissions parameters V function By link

```
void Swap( int &a, int
&b); int main(){
      int x,y;
      cout<<"\nEnter two numbers: ";
      cin>>x>>y;
      cout<<"\nx="<<x<<", y="<<y;
      Swap(x,y);
   cout<<"\nx="<<x<<", y="<<y<<endl;
      return 0;
}
void Swap(int&a, int&b){
      int c;
      c=a;
      a=b;
      b=c;
      return;
}
```

4. Study order returns parameters from functions

   - implement V environment programming next listing:

```
in t* GetMem();
int main(){
      int *a=GetMem();
      cout<<"\na is "<<a;
      cout<<"\n*a is "<<*a;
      cout<<endl;
      return 0;
}
int * GetMem(){
      int *c=new int
      (5); return c;
}
```

   - Analyze the program implementation process and the results obtained ; make written conclusions.

5. Usage mechanism overloads functions

   - implement V environment programming next listing:

```
void  PrintData( int *s, int N);
void  PrintData( char *w);
```

```
int main()  {
   int *a,n; char str[50];
     cout<<"\nEnter the array size: "; cin>>n;
     a = new int[n];
   for (int i=0;i<n;i++) a[i] = rand()%101-50;
     PrintData(a,n);
     cout<<"\nEnter a string: "; cin>>str;
     PrintData(str);
   return 0;
  }
void  PrintData(int *s, int N)  {
     cout<<endl;
   for (int i=0;i<N;i++) cout<<s[i]<<"  ";
   return;
  }
void  PrintData( char *w)  {
   cout<<endl<<w;
   return ;
  }
```

- do written conclusions about use mechanism function overloading in this program.

Exercise II

In accordance with the option, compose and implement a program; explain which method of passing parameters to the function was used and why:

1. Given four natural numbers, calculate their greatest common divisor using the subroutine for calculating the greatest common divisor of two numbers.
2. Given the coordinates of the vertices of two triangles, determine which one has the larger area.
3. Find the least common multiple of four given natural numbers using the least common multiple routine for two numbers.
4. Two natural numbers are called "friendly" if each of them is equal to the sum of all dividers another, for excluding itself (for example, 284 and 220). Find all pairs of "friendly" numbers that do not exceed a given natural number.
5. Given the lengths of the sides of a triangle, calculate the lengths of the medians of a triangle whose sides are the medians of the original triangle.
6. Calculate the area of an arbitrary convex hexagon given by the coordinates of its vertices.
7. Given lengths sides triangle. Calculate average arithmetic lengths of the bisectors of this triangle.

8. Calculate the total surface area of a regular pentagonal pyramid given the edge length.

9. Compose program, which will calculate amount needed numbers n members this arithmetic progressions By any two her members, numbers which are known. *Note* : $a_n = a + d(n-1)$ $s_n = \frac{(a_1 + a_n)}{2} \cdot n$ a_1),

– the first term of the progression; d – the difference of the progression; n – the number of the taken member, s_n – the sum of the first n terms of an arithmetic progression.

10. Create a program that will calculate the sum of the required number n of terms of a given arithmetic progression, given any term of the progression whose number is known, and the difference of the progression. See the note to the problem. 9.

11. Three points are given by their coordinates on a plane. Find the area and lengths of the diagonals of a parallelogram whose vertices are these points.

12. Calculate the perimeters of three triangles and their arithmetic mean. Each triangle is given by the coordinates of its vertices.

13. Find the total surface area of a regular n -gonal pyramid along the side of the base when $n = 4$.

14. Generate members rows Fibonacci, Not superior in advance given number n . Check that if the Fibonacci number f ($k > 4$) is prime, then k is also prime.

15. Calculate the area of an arbitrary convex pentagon given by the coordinates of its vertices.

16. Create a program that will calculate the sum of the required number n of terms of a given geometric progression for any term of the progression whose number is known and the denominator of the progression.

Note : $b_n = b_1 q^{n-1}$, $s_n = \frac{b_n \cdot q - b_1}{q-1}$, *Where* $q \neq 1$, b_1 - first member progressions;

q - denominator progressions; n - number taken member, s_n - sum first n members geometric progressions.

17. Create a program that will calculate the sum of the required number n of terms of a given geometric progression for any two of its terms whose numbers are known (see note to option 16)

18. Calculate the terms of the Fibonacci series that do not exceed a predetermined number n . Check For several fours $f_i, f_{i+1}, f_{i+2}, f_{i+3}$ consecutive members rows Fibonacci justice ratios $f_{i+2}^2 - f_{i+1}^2 = f_i \cdot f_{i+3}$.

19. Program the game "Guess the number". Program using a random generator numbers chooses whole number V range from 0 to 9. Guess This

number for three attempts. After each attempts reported more or less than the stated number of the intended one.

20. Find All simple numbers V interval between natural numbers A And b ($A > 2000$, $b - A \geq 20$)

21. Given whole number A . Calculate $\frac{1.7\,t(5)+2\,t(1+a)}{6-t(a^2-1)}$, где $t(x)=\sum_{k=0}^{10}\frac{x^{2k+1}}{(2x+1)!}$.

22. How many? ways Can take away team V composition m people from n candidates? Conduct calculations at m =5, n = 8; at m =6, n = 11.

$$\left(\begin{matrix} C^{m} = \frac{n!}{m!(n-m)!} \\ n \end{matrix}\right)$$

23. n intervals asked coordinates their own ends on coordinate direct and numbered. A point x on this line is also given. Find out how many intervals the point x *belongs to* and indicate their numbers.

24. Given three pairs of complex numbers, represented in algebraic form, organize the execution of four arithmetic operations on a pair of complex numbers into a subroutine. Create a program to perform these operations for each pair of given numbers.

25. Given the coordinates of the vertices of a triangle and the coordinates of two points on a plane, determine whether they lie inside the triangle.

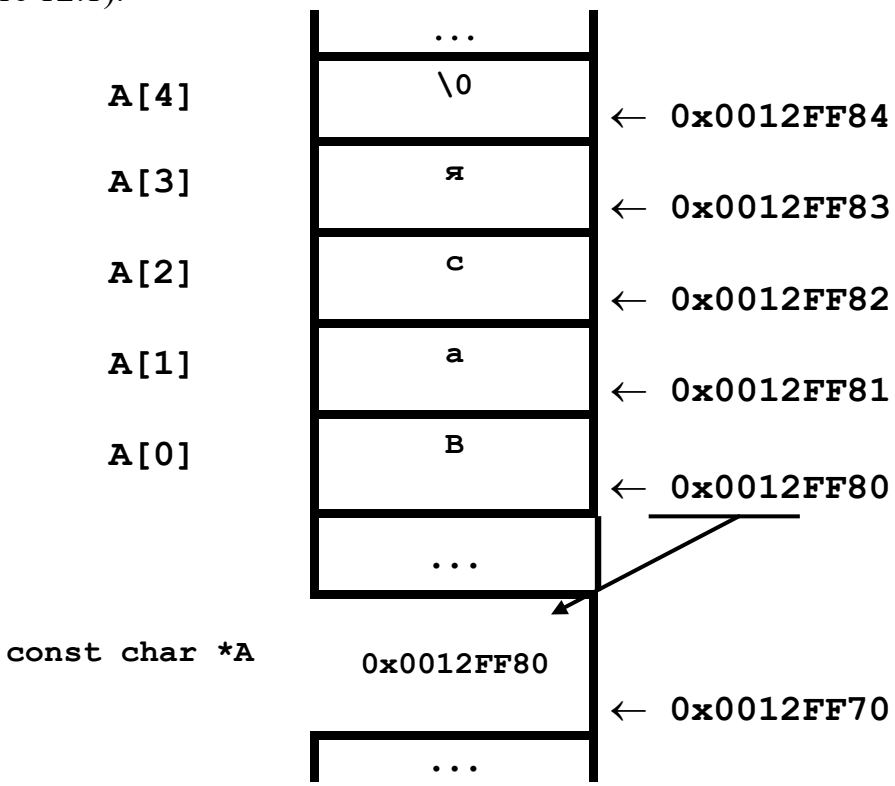
Control questions

1. List methods transmissions parameters V function.
2. Peculiarities transmissions parameters V function By meaning.
3. Peculiarities transmissions parameters V function By address.
4. What are peculiarities transmissions parameters V function By link?
5. Can a function return an array or a function as its result?
6. List methods call functions.
7. Which official word use to rename types?
8. What understand under overload functions?
9. Which process they call resolution clarification overload?
10. On how founded mechanism permissions overload? Give it to me explanations.
11. List rules descriptions overloaded functions.

12. Lineshow array elements type char

12.1. Methods performances lines V SI; implementation some typical operations on strings

C doesn't have a special string type. Strings are treated as arrays of characters terminated by a null character ('\0'). A string is accessed through a pointer to the first character in the string. The value of a string is the address of its first character. Thus, in C, a string can be said to be a pointer—a pointer to the first character of the string. In this sense, strings are similar to arrays because an array is also a pointer to its first element (Figure 12.1).



Drawing 12.1 - Performance array elements V memory computer

IN given example announced symbolic array **A** on 5 elements:

```
char A[]=”Vasya”;
```

Please note that one byte of memory is allocated for storing each character, and 4 bytes will be allocated for the pointer. It's important to note that in the example above, the array size is not specified; it is determined automatically. By length initializer strings, moreover, automatically Added end of line character.

A string can be declared either as an array of characters or as a variable of type `char *`. Each of the two declarations below is equivalent :

```
char S[] = "string";
char *Sp = "string";
```

assigns the initial value "string" to a string variable. The first declaration creates a 7-element array `S` containing the characters `'c', 't', 'p', 'o', 'To', 'A'` And `'\0'` . Second announcement creates pointer variable `Sp` ,

which points to a string containing the text `"string"` stored somewhere in the computer's memory. However, in any case, the number of stored characters is one greater than the number of significant characters due to the terminating null character.

Individual characters in a string are accessed using zero-based indices. For example, `S[0]` and `Sp[0]` - the first characters of the above declared strings, `S[1]` and `Sp[1]` - the second, etc.

In the declarations given, the length of the strings was determined automatically by the compiler. Can declare string variables given lengths.
For example, operator

```
char buff[100];
```

declares a variable `buff` that can contain a string of up to 99 significant characters plus a terminating null character.

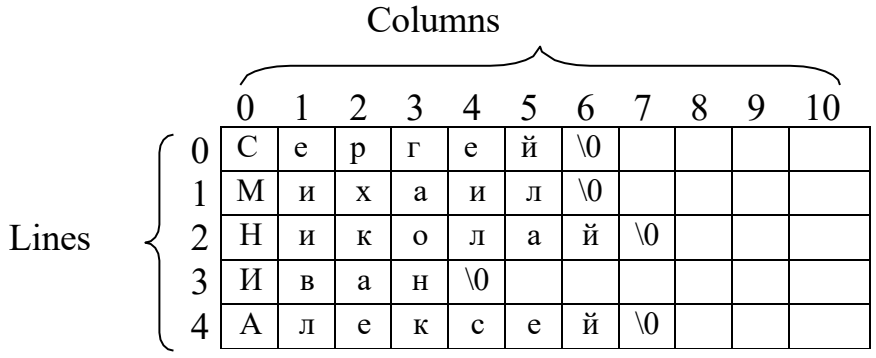
Now let's explore the most useful application of symbolic pointers: storing arrays of strings. While you can't actually create an array of strings, you can store an array of symbolic pointers, each of which contains the address of a string in memory.

Defining an array of character pointers creates a dynamic array . This array is similar to a two-dimensional table, with one exception: each row in this array can have a different length (a different number of characters per row).

For example, let's declare a character table with 5 rows and 10 columns, each row containing the same number of characters. It can be defined as follows:

```
char names [ 5 ] [ 10 ] = { { "Sergey"  },
                  { "Michael"  },
                  { Nikolai    },
      { "Ivan"     },
           { "Alexey"  }};
```

It is shown in Figure 12.2. Note that the table contains a lot of whitespace. Each row takes up 10 characters, even though the row contains much fewer characters. The empty elements contain the line terminator character because C writes this character to all uninitialized array elements. This type of table takes up too much memory.



Drawing 12.2 - Example two-dimensional symbolic array lines

To address the memory waste problem, we declare a one-dimensional array of character pointers. Each pointer contains the address of a string in memory, and these strings can have different lengths.

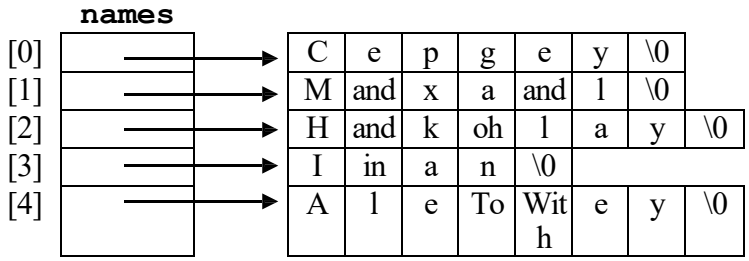
Here examples definitions (initialization) such arrays:

```
char *names [ 5 ] =
{
{ "Sergey"  },
{ "Mikhail"  },
{ "Nikolai" },
{ "Ivan"    },
{ "Alexey" }
};
```

or Just

```
char *names [  ] =
{
     { "Sergey"  },
     { "Mikhail"
     },
     { "Nikolai" },
     { "Ivan"    },
     { "Alexey" }
};
```

These are one-dimensional arrays. The asterisk before `names` indicates that this is an array of pointers. The pointers are of character type. Strings are not assigned to the array elements, but the array contains the addresses of these strings. Figure 12.3 shows this array of pointers. The strings can be located anywhere in memory. Their location is not important, since each pointer contains the address of the first element of the corresponding string. The strings do not take up extra space. memory; each line uses exactly as much space as it needs to accommodate itself and the line terminator character. This structure contains data in a jagged form.



Drawing 12.3 - Array of pointers

For conclusion first lines Can use cout operator : `cout « *names;`

```
// Print Sergey
```

For conclusion second lines is used next operator `cout:`

```
cout«*(names+1);    // Print Mikhail
```

When accessing an array element using the * operator, we access one of the array's rows. The value returned by this operator can be used anywhere a string constant or character array can be used.

Let's list some of the basic operations provided for a built-in string type (character array):

- initialization array symbols (line built-in type);
- input/output;
- Accessing individual characters of a string for reading and writing: This is done using the index operation or indirect addressing.

12.2. Main functions Si For work with lines

When working with character arrays, the task of processing them (comparing strings, concatenating strings, searching for a substring in a string, and so on) often arises. The C header file <string.h> describes many functions. for working with strings.

For example, functions:

- copying lines (**strcpy, strncpy**);
- comparisons lines (**strcmp , strncmp**);
- associations lines (**strcat , strncat**);
- search substrings (**strstr**);
- search occurrences symbol (**strchr , strrchr , strpbrk**);
- definitions lengths lines (**strlen**);
- transformations lines V number (**atoi , atoll , atof**);

and so on.

Given that strings are made up of characters, it's useful to know some functions for working with characters. The **ctype library** (header file **ctype.h**) is used to work with characters **.** It contains:

1. Functions checks on affiliation symbol set:
   - **isalmum (c) –** letters and numbers (AZ, az, 0- 9);
   - **isalfa (c) –** letters (AZ, a- z);
   - **iscntrl (c) –** managers symbols (With codes 0…31 And 127);
   - **isdigit (c) –** numbers (0-9);
   - **isgraph (c) –** printed symbols except space;
   - **islower (c) –** letters lower register (a- z);
   - **isprint (c) –** printed characters;
   - **ispunct (c) –** signs punctuation;
   - **isspace (c) – separator** symbols ;
   - **isupper (c) –** letters top register (A- Z);
   - **isxdigit (c) –** hexadecimal numbers.
2. Functions transformations characters:
   - **tolower (c) –** to the bottom register;
   - **toupper (c) –** to top register.

Lab Work No. 12 Processing symbolic arrays

TARGET WORKS: •receiving skills announcements, initialization, use of symbols arrays at processing lines in C/C++.

EXECUTION WORKS: master theoretical material, execute general For everyone exercise I And V in accordance With option to create a program (task II).

Task I

1. Studying ways formations lines V language Si

- Dial And debug program:

```
int main()
{
char Arr1[25];
char Arr2[]={"Second string!"};
char Arr3[]={'T','h','i','r','d',' ',
             's','t','r','i','n','g'};
char *Arr4="Fouth string";
cout<<"\nEnter string: \"First string\": ";
cin>>Arr1;
cout<<"\nString 1: "<<Arr1; // Оператор 1
cout<<"\nString 2: "<<Arr2; // Оператор 2
cout<<"\nString 3: "<<Arr3; // Оператор 3
cout<<"\nString 4: "<<Arr4; // Оператор 4
return 0;
}
```

- Analyze the output of the program and answer the question why **Operator 2** and **Operator 4** display their phrases on the screen in full **,** while operator 1 displays only part of it.
- Why **Operator 3** outputs all phrase lines `Arr3` and Part another line.
- What need to do, to **Operator 3** brought out only my phrase.

2. Using string functions to process character arrays

- Dial And debug program filling numerical array of character string:

```
int main(){
   char str[]="2, 75.3, 24.45, 55, 0, 0.37";
   char *ptr=str;                          //Operator 1
   float Arr[10];
   int i=0;
   printf("\nOriginal string:\n%s\n",str);
   do {
     Arr[i] = atof(ptr);                   //Оператор 2
```

```
    i++;
  if (i>9) break;
    }
  while(ptr==strchr(ptr,','),ptr++);    //Оператор 3
  printf("\nМассив чисел: \n")
  for (int s=0;s<i;s++)
  printf("%5.2f\n", Arr[s]);
  return 0;
}
```

- Analyze result work programs And answer on Question: What happens when **Operators 1-3 are executed** ?

Exercise II

1. In the given text, insert a comma and a space between words instead of spaces .
2. Given text. Insert the given sequence of characters after the *i* -th character of each word.
3. Display words from a given text in one column on the screen ending in a consonant letter, in another column - words ending in a vowel letter, in the third - all the remaining words.
4. From the given text, remove the part of it enclosed in parentheses together with the parentheses.
5. Each word text transform So, to it it was read vice versa.
6. From a given sequence of words separated by commas, remove words containing the character entered from the keyboard.
7. Create a program to count the number of words in a telegram and print a receipt for its payment.
8. Given a text of *n* characters, display only the lowercase Russian letters contained in this text.
9. Given a text in lowercase Latin letters, followed by a period. Display this text on the screen using capital Latin letters.
10. After each vowel letter *x* in a sequence of words separated by spaces, insert the letter combination *pxx* (for example, the word *order* is transformed into *porosoryasyasadorosok).*
11. Given a text, find how many double letters it contains. Replace each pair with the * symbol.
12. Define, Not using standard procedure, is whether The entered string is a correct representation of an unsigned decimal integer.
13. IN text words separated spaces. Define, what percent words in the text ends with the Russian letter *a.*
14. Asked text on Russian language. Delete from him spaces, ь, ъ.

15. Words in the text are separated by spaces. Determine what percentage of words in the text end with the Russian letter *a.*
16. Given two strings of characters consisting solely of digits (each more than 10 characters long), assume these strings contain very long integers and form a third string—the sum of these numbers.
17. Find and display double letters in the given text. Count the number of double vowels and the number of double consonants.
18. Given a square matrix whose elements are positive integers, find the square root of the number formed by the digits of the numbers on the main diagonal.
19. Given some arbitrary text, edit the text so that there is exactly one space between words and exactly two spaces between sentences.
20. Remove a character entered from the keyboard from the input string. Display the number of removed characters and the resulting string.
21. Words in the text are separated by spaces. Determine what percentage of words in the text begin with the Russian letter *"m."*
22. Withdraw on screen the most long word from given text.
23. The text is defined as follows: the first character is a number indicating the length of the first word, after the first word is another number indicating the length the second word (the length of each word is no more than 9 characters), etc. Write out *the k* -th word from the text (the number *k* is entered from the keyboard).
24. Let's define the complexity of a sentence as the sum of the number of words and punctuation marks. Determine the complexity of a given sentence.
25. IN given line replace All occurrences one substrings on another. Both substrings are entered from the keyboard.
26. Select from the text all words that contain two or more identical symbols.
27. Given a square integer matrix whose elements are positive numbers, find the square root of the number formed by the digits of the numbers on the secondary diagonal.

Control questions

1. What represents from myself line V program, written on JP C?
2. What represents by yourself line:
   A) subsequence English letters; b) a
   sequence of Russian letters;
   V) subsequence symbols PC?
3. How Maybe be described line V Si?
4. Bring example announcements lines How array symbols.
5. Bring example announcements lines How pointer variable on line.
6. What is sign endings lines?
7. How is being carried out access To elements lines?

8. Which maximum quantity symbols Maybe contain line?
9. How at description lines can be given her length?
10. During program execution, can the actual string length become larger than the length declared in the declaration section? What happens in this case?
11. List typical operations over lines.
12. List main functions processing lines.

Example execution tasks II laboratory work

Task: You are given a sentence with one space between words and **a period at the end of the sentence** . Count the number of words in this sentence and Display in a column on the screen those words that begin with the letter or symbol entered from the keyboard, and their number. If there are no such words in the entered sentence, display a message to that effect.

1. **Solution (mathematical model) .** We will enter a sentence into the character array `text[100]` . We will break this sentence into individual words and store them as elements of a two-dimensional character array `mas` . We will denote the character entered from the keyboard by `x` , the number of words in this sentence by `m (int)` , and the number of words beginning with the entered character (the result of the program) by `kol_x (int)` . Formation of the array `mas` We'll execute this using a loop with a postcondition until a period is encountered. Each word will be processed character-by-character in the loop, processing the one-dimensional array `text` until a space is encountered, incrementing the word counter by one after each space.

2. Solution **program** this tasks Maybe have next view:

```
#include <conio.h> // to work correctly with Cyrillic, you must #include
<iostream> // save the project file in DOS 866 encoding file →#include
<cstdio> // additional parameters saving →Cyrillic (DOS)

usingnamespace std;

int main() {
     char ch, x, mas[25][100], text[100]; // mas - two-dimensional array
       words cout<< "Enter a sentence ending with a period" <<endl;
       gets(text); // organize the input of all text characters
     int i=0; // character number of the word
     int j=0, m=0; // we are considering first
     word do {
                ch=text[j];
                if ((ch!= ' ' ) && (ch!= ',' ) && (ch!= '.' ))
                     {
                          mas[m][i]=ch;    //form current word i++;

                     }
else
                if
                (i!=0)
                   {
                     mas[m][i]= '\0' ;
                     m++;              //increase the word counter
```

```
                        i=0;
                      }
                j++;
        }
      while (ch!= '.' );
      cout<< "Number of words in this sentence: " <<m<<endl; cout<<
      "Enter the character with which words displayed on the screen
      should begin: " ;
       cin>>x;
      int kol_x=0; // reset the counter of searched words
      cout<< "Words starting with the character '" <<x<< "':"
      <<endl; for (i=0; i<m; i++)
           if (mas[i][0]==x)
           { // If introduced symbol V current word on first positions, then
cout<<mas[i]<<endl;;
             kol_x++;
           }
      if (kol_x==0)    cout<< "absent" ;
else
{           cout<< "Total in sentence " <<kol_x;
switch (kol_x)
                  { //organizing the output of the correct
                     ending case 1: case 21:cout<< " word, "
                     ; break ;
                     case 2: case 3: case 4: case 22: case 23: case 24:
                cout<<" слова, ";break;
                     default:cout<<" слов, ";
                  }
           }      cout<< "starting with '" <<x<< "'." ;
      _getch();
      return 0;
}
```

3. Result work programs:

1)

```
Enter a sentence
Some People see V puddle only puddle, A some see V puddle stars. Number of
words in this sentence: 13
Enter symbol, With whom should start off output on screen words: l
Words, beginning on symbol "L"
people
pudd
le
just
pudd
le
pudd
le
There are 5 words in the sentence that start with 'l'.
```

2)

```
Enter a sentence
And the sun shone brighter.
The number of words in this sentence is 4 .
Enter symbol, With whom should start off output on screen words:l Words
starting with the symbol 'l'
```

are absent

13. Lines How object special class string

13.1. Description lines V C++ And typical operations over them

To use the capabilities of the `string class in a program` It is necessary to include the header file `string at the beginning of the program` :

```
# include <string>
```

Unlike the `char type` , `string` is a class. Classes, roughly speaking, contain variables, constants, and functions (methods) for working with variables. `string` uses the only one `string` object for string handling. Being part of the C++ standard library, this object is also part of the standard namespace – `std` . Thus, to use the capabilities of the `string header file` , it is necessary to declare the standard namespace `std in the global object declaration area of the program` .

Now For creation lines enough write:

```
strings;
```

For records V line Can use operator assignment (=):

```
s="Hello";
```

Example work (input/output) With object class `string` :

```
string name;
cout<<"Enter your name"<<endl;
cin>>name;// the line is counted up to the first space
//therefore, when working with string, it is better to use
getline(cin,name);
//orcin.getline(name[, 50]); V [] specified optional parameter
cout<<"Hi "<<name<<"!"<<endl;
```

Let's look at some basic operations implemented for an object of the `string` class :

- initialization with a character array (a built-in string) or another object of type `string` ;

```
string st( "For work With Cyrillic Dos 866\n" );// or
strings0=st;// or
string s1 = " file → additional parameters saving → Cyrillic (DOS) ";
```

- copying one lines V another;

```
st2 = st3;//Using the assignment operation, we copy st3 to st2
```

- access To separate symbols lines For readings and records;
- comparison two lines for equality;

```
if ( st == st3 )
// initialization worked
```

- concatenate two strings, producing the result as either a third string or instead of one of the original ones;

```
string s1( "hello, " );
string s2( "world\n" );
string s3 = s1 + s2;// 3rd line - result concatenation the first two
s1 += s2;// or same possible addition s2 V end s1
```

We can rewrite the example above so that special characters and punctuation marks are represented by a built-in type, and meaningful words are represented by objects of the string class:

```
const char *pc = ", ";// Operation additions Maybe concatenate
string s1( "hello" );//         objects of class string
string s2( "world" );//    and with lines built-in type
string s3 = s1 + pc + s2 + "\n";
```

Expressions like these work because the compiler knows how to automatically convert objects of the built-in type to objects of the `string class` . Maybe And simple assignment built-in lines object `string` :

```
     string s1;
const char *pc = "a character array";
s1 = pc; // correct
```

13.2. Main functions class string For work with lines

Let's look at some of the basic methods (functions) provided for working with strings that are instances of the `string class`:

- `s.append(str)` - adds V end lines line `str` . Can write How `s.append(variable)` , so and `s.append("string");`
- `s.assign(str)` - appropriates line s meaning lines `str` . Likewise records `s=str;`
- **`int`** `i=s.begin()` - writes down V `i` index first element lines;
- **`int`** `i=s.end()` - similarly, but the last one;
- `s.clear()` – as the name suggests, clears a string, i.e. removes all elements in it;
- `s.compare(str)` – compares the string `s` with the string `str` and returns 0 if there is a match (in fact, it compares the character codes and returns their difference);
- `s.copy(str, len (how much), start (starting from which))` – copies from string `s` into the string `str` (`str` can be either of `the string type` or a string of `the char type`). The last two parameters are optional (you can use a function with 1, 2 or 3 parameters);
- **`bool`** `b=s.empty()` - If line empty, returns **`true`** , otherwise **`false;`**
- `s.erase(start, len)` deletes `len` elements With given `start` positions ;
- `s.find(str, start)` - is looking for line `str` starting With given `start` positions ;
- `s.insert(start,str, beg, count)` - inserts V line `s`, starting With positions `start` part of string `str`, starting from the `begcount position` symbols;
- `s.replace(start, n,str)` - beret `n` first symbols from `str` and replaces characters of the string `s` on them, starting from the `start position`;

```
replace(str.begin(), str.end(), '.', '_');//runs through range from
// begin() to end() And replaces elements, equal to the third
parameter,
// on the fourth
```

- **int** `len=s.length()` - writes down V `len` length lines;
- `s.push_back(symbol)` - adds V end strings character;
- `str=s.substr(n,m)` - returns `m` characters starting With position `n`;
- `s.swap(str)` - changes content `s` And `str` in places;
- `s.size()` - returns number elements V line.

Example research features input/output string data.

…

```
string s, s1, ss;
cout<<"\nEnter string: \"1 string\": "<<endl;
cin>>s;
getline(cin,s1);
cout<<"\nEnter string: \"2 string\": "<<endl;
getline(cin,ss);
cout<<"\nString 1: "<<s;
cout<<"\nпродолжение String 1: "<<s1;
cout<<"\nString 2: "<<ss<<endl;
string str = "Hello, world!";
cout << str <<endl;
string name,surname,text,fullname,s2,s3,user;
user = "Petya Petrov";
cout << user <<endl;
cout<<"Enter your name"<<endl;
getline(cin,name);
cout<<"Enter your surname"<<endl;
getline(cin,surname);
fullname=name;
fullname+=" "; // добавляемпробел
fullname.append(surname);
cout <<"fullname = "<<fullname <<endl;
if (fullname.compare(user)==0) // <=> if (!(fullname.compare(user)))
     cout<<"Your are good user"<<endl;
     else       cout<<"Bad user"<<endl;
cout<<"enter s1"<<endl;
getline(cin,s1);
cout<<"enter s2"<<endl;
getline(cin,s2);
s1.swap(s2);
cout<<"new s1: "<<s1<<endl<<"new s2: "<<s2<<endl;
cout<<"Enter big text with your name"<<endl;
getline(cin,text);
cout <<"old text = "<<text <<endl;
int i=text.find(name);
while (i) // <=> while (i!=0)    {
text.replace(0,name.length(),name); //replace the first
characters in text with name s3=text.substr(i,name.length());
//place the substring from text into s3
  cout<< "Replaced: " <<s3<<endl; i=text.find(name); //search for the
substring name in the string text
}
cout<< "New text:" <<endl<<text<<endl;
text.clear(); //clear the string cout<<
"text: " <<text<<endl;
```


…

Laboratory Job No. 13
Programming tasks on processing data string type

TARGET WORKS: receiving skills announcements, initialization And using objects of the `string class` C++.

EXECUTION WORKS: master theoretical material, in in accordance with option to solve the problem, create and implement a program With using functions class string.

Exercise

1. Enter a sentence with words separated by spaces and commas. Print this sentence, removing any words that appear more than once.
2. Find the set of all words that occur in each of the two given sentences.
3. Delete from this lines those words, which already met V her before.
4. Enter a string consisting only of numbers and letters. Print those groups of numbers in which the number 7 appears no more than twice (a group of numbers is a sequence of numbers surrounded by letters).
5. Enter a sentence with words separated by spaces and commas. Print the words that are grammatical precedents of other words in the sentence. If the required words are missing, report them.
6. Print the entered sentence, removing from it words that consist of less than three letters.
7. Print the entered words, other than the last one, by rearranging them as follows: move the last letter of each word to its beginning and leave only the first occurrences of each letter in each word.
8. Print the string that is obtained from the input string as follows: each digit is replaced by a sequence of characters enclosed in parentheses: “+” if the digit is even, or “-” if the digit is odd. The length of the sequence is equal to the number represented by the digit.
9. Students encrypt their notes by writing all the words backwards. Create a program that decrypts the students' messages.
10. Print the entered string, removing odd-numbered words from it. and reversing the even-numbered words. For example, the line *"the cold stream burned my face and hands"* should become *"yechuur otsil ikur."*
11. Enter a string that can only contain numbers and letters. Print those groups of letters in which the letter *A* occurs at least twice (a group of letters is a sequence of letters surrounded by numbers).
12. Enter any text consisting of several sentences. Calculate average number of words per sentence and average sentence length.

13. Given a text in which words are separated by spaces, create a list of words in the text, indicating the number of letters in them.
14. Print, starting from the last word, the inverted words of the entered sentence.
15. From the first line, select all words beginning with a vowel, and from the second line, select words beginning with a consonant. Create a third line consisting of the selected words from both lines. In the new line, words should be separated by two spaces.
16. From the entered text, print only those words that are symmetrical (for example, *radar, anna, hut)* or in which the letters are arranged in reverse alphabetical order (for example, *current, three, term).*
17. Students encrypt their notes by writing each word of the text backwards. Compose program, allowing encrypt messages in the manner described .
18. For each word in a given sentence, indicate how many times it appears in the sentence.
19. Replace the ending *ing* of each word occurring in the given sentence in English with *ed.*
20. Given a sequence of words separated by spaces, print all the words, first removing all previous occurrences of the last letter from each word.
21. From a given sequence of words separated by commas, remove words containing the sequence of characters entered from the keyboard.
22. Given a sequence of words separated by spaces, print all the words, first removing from each word all occurrences of the last letter of the previous word.
23. Display on the screen those words from the entered text that do not contain repeated letters.
24. Given a text consisting of several sentences, display all words other than the last word of the current sentence for each sentence.
25. Given a sequence of words, each of which contains from 2 to 10 Latin letters. Between adjacent words - space, for last one Word-period. If the word is odd-length, remove the middle letter; if it is even-length, move the first letter to the end of the word. Display all converted words on the screen.
26. In the given text, words are separated by spaces or commas. Display a list of words that contain at least one character entered from the keyboard.
27. Withdraw on screen display list words, having prefix.

Tasks For independent work

Next group tasks is Not mandatory For solutions And This problem relates to the field of cryptography, i.e., the encryption and decryption of texts. Solving problems in this group will improve your string processing skills and expand your knowledge of their practical applications.

1. Encrypt the text using the Gronsfeld method. The key is a finite sequence of digits written consecutively above the characters of the text being encrypted. The digit above each character is the offset (i.e., it indicates how many positions must be moved forward in the encoding table from the current character to obtain its replacement). Test the program you've written.

2. To encrypt text written using Russian letters and punctuation marks, it can be rewritten by replacing each letter with the one immediately following it in the alphabet (the letter " *я″* is replaced by the letter " *а* "). This encryption method can be generalized by shifting not one letter, but *n* letters, where *n* is a given natural number (you can imagine the letters written in a circle, like the numbers on a clock face). Write and test a program that encrypts and decrypts this text.

3. Let a message be transmitted over a certain communication channel, which has the form of a sequence of zeros and ones (or similarly, dots and dashes). Due to interference, it is possible erroneous reception some signals: zero Maybe be perceived as one and vice versa. Each signal can be transmitted three times, replacing, for example, the sequence 1, 0, 1 with the sequence 1, 1, 1, 0, 0, 0, 1, 1, 1. Three consecutive digits are replaced during decryption with the digit that appears at least twice among them. This concealment of signals increases the likelihood of correct reception of the message. Write a program to decrypt a given message.

4. The following method is designed to encrypt sequences zeros and ones (or, for example, dots and dashes). Let $a_1, a_2, ..., a_n$ - such a sequence. What is proposed as its cipher is the sequence $b_1, b_2, ..., b_n$, educated By to the next one rule: $b_1 = a_1$; $b_i = 1$ if $a_i = a_{i-1}$ and $b_i = 0$ otherwise ($i = 1, 2,..., n$).

   Using the described method, encrypt and decrypt the given sequence of zeros and ones (or dots and dashes).

5. To encrypt a text consisting of 121 letters, it can be written in a square matrix of order 11, row by row, and then read in a spiral, starting from the center (i.e., from the element with indices 6, 6). Encrypt and decrypt this text using this method. Having chosen a table, develop a method for replacing each symbol with another symbol—its cipher. Encrypt and decrypt this text using the specified method.

Control questions

1. How is a string variable declared that is an instance of the `string class` ?
2. How is being carried out access To elements lines?
3. List typical operations over lines.
4. How is being carried out initialization lines?
5. How is being carried out input/output lines, being object class `string` ?
6. How is being carried out comparison lines?
7. How is being carried out concatenation lines?
8. Maybe whether V process execution programs change actual length of the string?
9. List main functions processing lines.

Example execution laboratory Work Task: Find the sum of the digits of the entered natural number.

1. **Solution (mathematical model + verbal algorithm) .** We'll read the natural number into the string variable `stroke` . We'll reset the adder to `s = 0.` We'll calculate the string length and then loop through the following:
   - We will convert each character of the string into a number, thereby obtaining the digit of the entered number;
   - Increment the adder by the value of the resulting digit of the

   number. After the loop completes, we display the result—the sum of the digits of the number `s` . **Arguments:** `line (string)` - whole number entered With keyboard. **Results:** `s (int)` – Sum of the digits of a number

   Intermediate values: counter `i,` number of elements in row `n,` numerical performance symbol introduced lines `si` – whole type.

2. Solution **program** this tasks Maybe have next view:

```
#include <conio.h> #include
<iostream> #include
<string>

usingnamespace std;

int main()  {
        string ;
      cout<< "Enter a natural number:" <<endl;
      getline(cin,string);
      int si, s=0, n=stroka.length();
      for (int i=0; i<=n-1;i++)  {
                              si=stroka[i]-'0';
                              s=s+si;
      }
        cout<<"Сумма цифр числа: "<<s<<endl;
        _getch();
      return 0;
}
```

3. Результат работы программы:

```
Введите натуральное число:
12345678
Сумма цифр числа: 36
```

14. Структуры

14.1. Order announcements And initialization structures

Structures in C are used to logically group related data. Unlike arrays, a structure can contain data of different types.

In the C++ language, a structure is a type of class and has all its properties, but at this stage of learning the C++ language, it is enough to use structures as they are defined in the C language:

```
struct [ type_name ]
  {
       type_1 element_1;
       type_2  element_2;
       . . .
       type_n  element_n;
  } [ list of descriptors ];
```

The elements of a structure are called *the structure's fields* and can be of any type except the structure's own type, but can also be pointers to it. If a type name is missing, a list of descriptors must be provided: variables, pointers, or arrays. In this case, the structure declaration serves as a definition of the elements of this list:

```
struct // An example of a structure describing student profile data
  {
  char fam[20];
  char name[15];
  intage;
  } stud[30], *ptr;
```

This example describes a structure containing three fields: a 20-element character array named `fam` , a 15-element character array named `name` , and an integer variable named `age` , which are used to store information about a student's last name, first name, and age, respectively. The descriptor list specifies an array of 30 instances of this structure and a pointer to the structure (which also specifies an instance of the structure, but is identified not by the instance name, but by the name of the pointer storing the address of the beginning of the memory allocated for the instance of the structure, but without the instance name). If the descriptor list is missing, the structure description defines new type, Name which can be used V further along with standard types, for example:

```
struct  student
  {
       char fam[20];
       char name[15];
       int  age;
  };
int main()
  { . . .
```

```
        student  stud[30], *ptr;
. . .
        return 0;
}
```

This example describes a global data structure of the struct type with the definition of a new data type, `student` . The definition of instances of this structure is performed within the `main() function` .

Variables of structured type can also be placed in dynamic memory area, for this you need to describe a pointer to the structure and allocate space for it:

```
student  *ptr=new student;    // copy structures
student *stud=new student[N];// array of instances
// structures
```

To initialize a structure with values, the same methods are used as for ordinary data types, but with a small addition: to access a field of a structure, an instance of which was created as an ordinary variable or array, use the “point” operation, and to access a field of a structure, an instance of which was created as a pointer to a structure, use the “arrow” operation:

```
stud[1].fam=”Sidorov”;       // Access instance name
stud[1].name=”Alexander”;
stud[1].age=23;
prt->fam=”Petrov”;       // Access via pointer
ptr->name=”Semyon”;      // to the structure
ptr- >age=21;
```

In practice, the input of field values of a structure described by an array of instances is carried out using loops with a parameter:

```
for (int i=0; i<N; i++)
  {
  cout<<”\nEnter last name student: “;
  cin >>stud[i].fam;
  cout<<”\nEnter Name student: ”;
  cin >>stud[i].name;
  cout<<”\nEnter age student: ”; cin>>stud[i].age;
  }
```

In this example the number of students `is N` must be defined (introduced With keyboards) higher By text programs, to moment use.

14.2. Programming With using structures

For variables of the same structural type, **the assignment operation is defined** , which involves element-by-element copying.

By themselves structures are only means descriptions Complex data types, but their main advantage is the ease of searching and analyzing the data described in this format. For example, we need to display data on all students currently under 22 years old:

```
k=0;
```

```
for (j=0; j<N; j++)
        if ( stud[j].age<=22 )
              {
                    cout<<”\nLast name: ”<<stud[j].fam;
                    cout<<”\tName: ”<<stud[j].name;
                    cout<<”\tAge: ”<<stud[j].age; k++;
              }
  if (k==0) cout<<”\nThere are no such students!”;
```

or O students, whose surname " **Petrov** ":

```
k=0; //f=true;
for (j=0; j<N; j++)
        if ( strcmp(stud[j].fam, “Petrov”)==0 )
              {
                    cout<<”\nLast name: ”<<stud[j].fam;
                    cout<<”\tName: ”<<stud[j].name;
                    cout<<”\tAge: "<<stud[j].age; k++;
                    // f=false;
              }
if (k==0) cout<<”\nThere are no such students!”;
// if (f) cout<<”\nThere are no such students!”;
```

Proposed V comments alternative `boolf` for considered V `intk` example is a more correct implementation of the required operation.

14.3. Usage functions For work With derivatives data types

A structure can be passed to a function and returned as the function's value.

IN in accordance With modular approach To programming It is advisable to implement search questions in separate functions, for example:

```
void RequestFamily(student  *k, char  *f);

int main()

{ . . .
char *family;
. . .
cout<<”\nEnter last name For search: ”;
cin >>family;
RequestFamily(courses, family);
. . .
return 0;
}

void RequestFamily(student  *k, char  *f)

{
   k=0;
   for (j=0; j<N; j++)
        if ( strcmp(k[j].fam, f)==0 )
              {
                    cout<<”\nLast name: ”<<k[j].fam;
```

```
                    cout<<”\tName: ”<<k[j].name;
                    cout<<”\tAge: ”<<k[j].age; k++;
                }
    if (k==0) cout<<”\nThere are no such students!”;
    return ;
}
```

In this example, two pointers are passed to the function: one of type `student` , the other of type **`char`** . What are these pointers, why are they passed, and what do they mean?

The first pointer is a pointer to the `student data type` , that is, to a structure. Why is a pointer passed? Because structure instances are declared as an array of instances, and it's known that the array name is a pointer to the zeroth element of the array. Thus, we pass the entire array of structure instances to the function via a pointer.

The second pointer is a pointer to the **`char data type`** , that is, a character data type. This pointer is used to pass the string to the function to search for the student's last name.

The function returns nothing as a result (the return type **`is void`**), but we don’t need to return anything, since the function already displays a list of students we are interested in.

Laboratory Job No. 14
Application structures For solutions applied tasks

PURPOSE OF THE WORK: to acquire basic skills in declaring, initializing and using derived data types, to improve skills descriptions And use functions V programs.

EXECUTION WORKS: V in accordance With option make up And implement programs.

Exercise I

Using a structured type – **`struct`** (structure), create a program entering a list of objects from the keyboard (at least 10) On the topic specified in the corresponding assignment, 4-6 characteristics of these objects and displaying the data on the screen in the form of a rectangular table. Organize a data search based on one of the specified characteristics.

The information must be reliable. It can be obtained from reference books, encyclopedias, advertisements, books, etc.

1. Signs zodiac
2. Industrial goods
3. Products nutrition
4. States Europe
5. Russians writers
6. Homemade And wild animals
7. Flowers
8. Household technique
9. Computers And components
10. Precious stones
11. Rivers Russia
12. Types sports
13. Music
14. Materials For repairs apartments
15. Universities Russia
16. Intelligence O students groups
17. Painting eras Renaissance
18. Chemicals elements of the periodic table
19. Medicine
20. Tourism And rest
21. Transport
22. Citrus plants
23. Mountains Northern Caucasus
24. Perfumery
25. Stationery goods

Exercise II

In accordance with the option, draw up and implement a program. Provide processing data With using MENU, containing the following items: data entry, data viewing, search 1, search 2, exit.

Option 1

1. Describe the STUDENT structure, which contains the following fields: last name and initials, group number, academic performance (array of five elements).
2. Write program, performing next actions:
   - entering data from the keyboard into an array of structure instances consisting of five records;
   - search records, V which average ball student more 4.0;
   - search records, V which match numbers groups students.

Option 2

1. Describe the AEROFLOT structure, which contains the following fields: flight destination name, flight number, capacity, aircraft type.
2. Write program, performing next actions:
   - entering data from the keyboard into an array of structure instances consisting of five records;
   - search for records in which the destination matches the one entered from the keyboard;
   - search records, V which capacity airplane Not less introduced With keyboards.

Option 3

1. Describe the structure of TRAIN, which contains the following fields: train number , destination name, departure time, travel time.
2. Write program, performing next actions:
   - entering data from the keyboard into an array of structure instances consisting of five records;
   - search trains, departing after time, introduced With keyboards within the next two hours;
   - search trains, departing V paragraph appointments, introduced With keyboards.

Option 4

1. Describe the TOURIST structure, which contains the following fields: route number, name of the starting point of the route, name of the final point of the route, travel time.
2. Write program, performing next actions:
   - entering data from the keyboard into an array of structure instances consisting of five records;
   - search for routes starting at the point whose name is entered from the keyboard;
   - search for routes whose total travel time does not exceed the time entered from the keyboard.

Option 5

1. Describe structure NOTE, containing next fields: surname, Name, phone number .
2. Write program, performing next actions:
   - entering data from the keyboard into an array of structure instances consisting of five records;
   - search information O person, whose surname introduced With keyboards;
   - search information O person, whose number phone introduced With keyboards.

Option 6

1. Describe structure SIGN, containing next fields: surname, Name, Date of birth, zodiac sign.
2. Write program, performing next actions:

- entering data from the keyboard into an array of structure instances consisting of five records;
- search information O people, born Today;
- search information O people, born under led familiar zodiac.

Option 7

1. Describe the PRICE structure, which contains the following fields: product name, name of the store to which the product was shipped, and product cost.
2. Write program, performing next actions:
   - entering data from the keyboard into an array of structure instances consisting of five records;
   - search for information about stores to which the product, the name of which was entered from the keyboard, was shipped;
   - search information O goods, price which Not exceeds the cost entered from the keyboard.

Option 8

1. Describe the ORDER structure, which contains the following fields: taxpayer's last name, taxpayer's initials, taxpayer's TIN, tax amount.
2. Write program, performing next actions:
   - entering data from the keyboard into an array of structure instances consisting of five records;
   - search information O taxpayer, last name whom introduced from the keyboard;
   - search information O taxpayers, whose total tax exceeds the amount entered from the keyboard.

Option 9

1. Describe the structure of a CAR containing the following fields: vehicle make, state registration number, owner's last name, owner's initials, quarterly mileage (an array of four elements).
2. Write program, performing next actions:
   - entering data from the keyboard into an array of structure instances consisting of five records;
   - search information O owners cars, brand whom introduced from the keyboard;
   - search information about cars, at which total mileage for 2 And 3 quarters does not exceed the value entered from the keyboard.

Option 10

1. Describe structure STUDENT, containing next fields: surname, Name, patronymic, passport series, passport number, date of issue.
2. Write program, performing next actions:
   - entering data from the keyboard into an array of structure instances consisting of five records;

- search records, V which matches Name student;
- search records, V which matches month issuance passports.

Option 11

1. Describe the AEROFLOT structure, which contains the following fields: flight number, departure time, check-in start time, aircraft type.
2. Write program, performing next actions:
   - entering data from the keyboard into an array of structure instances consisting of five records;
   - search for flights for which check-in has already started in relation to the time entered from the keyboard, but has not yet ended (the end time of check-in is considered to be the departure time minus 20 minutes).

Option 12

1. Describe the WORKER structure, which contains the following fields: employee's last name and initials, date of birth, year of commencement of employment, year of employment.
2. Write program, performing next actions:
   - entering data from the keyboard into an array of structure instances consisting of five records;
   - search employees, whose labor experience Not less introduced With keyboards;
   - search employees, born V flow months, introduced With keyboards.

Option 13

1. Describe the structure of TRAIN, which contains the following fields: train number, name of the route destination, number of sleeping, compartment and reserved seat berths (array of three elements), number of available sleeping, compartment and reserved seat berths (array of three elements).
2. Write program, performing next actions:
   - entering data from the keyboard into an array of structure instances consisting of five records;
   - search for trains departing to the destination entered from the keyboard and having the number of free reserved seats specified by the user.
   - conclusion information O on the train, number whom introduced With keyboards.

Option 14

1. Describe the structure of a NOTE containing the following fields: last name, first name, meeting date, meeting location.
2. Write program, performing next actions:
   - entering data from the keyboard into an array of structure instances consisting of five records;
   - search information O meetings, appointed on date, introduced With keyboards;
   - search information O meetings, appointed V place, introduced With keyboards.

Option 15

1. Describe the PRICE structure, which contains the following fields: store name, product name, quantity of product in stock, product cost.
2. Write program, performing next actions:
   - entering data from the keyboard into an array of structure instances consisting of five records;
   - search for information about stores that have the product whose name was entered using the keyboard;
   - Search for information about stores that have a product whose name is entered using the keyboard, with a price not exceeding the value entered using the keyboard.

Control questions

1. Which types data relate To simple, A which to the composite?
2. What such structure (**`struct)`** ?
3. How are announced structures?
4. Right whether, What fields structures should have one And That same type?
5. How is different structure from array?
6. Which operations acceptable over structure V in general?
7. How is being carried out link on components structures?
8. Can whether in quality fields of the structure describe structure?

Example execution tasks I laboratory work

Assignment: Using **`the struct structured type`** , create a program to enter a list of objects (at least 10) on the topic specified in the corresponding assignment, along with 4-6 characteristics of these objects, and display the data on the screen as a rectangular table. Organize a data search based on one of the specified characteristics.

The information must be reliable. It can be obtained from reference books, encyclopedias, advertisements, books, etc.

Subject: **Planets Solar systems** .

1. Solution (information model + verbal algorithm) . So, as an example of a primitive relational database, we will provide some information about the planets of the solar system, formatted as a table. The table will consist of a header with field names, 9 records (the number of planets in the solar system), and 6 fields: the ordinal number, the name of the planet, its distance from the Sun in astronomical units (an astronomical unit is the distance from the Earth to the Sun, equal 149.6 million km), period appeals planets around Suns V light years, the presence of the planet's own satellites and the basic chemical composition of the planet's atmosphere.

Preparing such a table manually is called constructing an information model of the problem being solved. This is where you should begin completing an individual assignment.

Like this in this way, table should have next view:

No.	Name of the planet	Distance from the Sun (in a.e.)	Period of revolution (in St.G.)	Availability of a satellite	Main cast atmosphere
1	Mercury	0.387	0.21	-	Ar, No, He
2	Venus	0.700	0.61	-	CO2
3	Earth	1,000	1.00	+	N,O_2
4	Mars	1,520	1.88	-	CO_2
5	Jupiter	5,200	11.90	+	H_2, CH_4, NH3
6	Saturn	9,500	29.46	+	CH_4, H_2, No
7	Uranus	19,180	84.00	+	H_2, No, SN_4
8	Neptune	30,070	164.80	+	CH_4, H_2, No
9	Pluto	39,400	247.70	+	CH4 – methane

When compiling a data input/output program in the form of the table below, we will use a one-dimensional array `mas` of size 20, elements which have `the struct type` . The number of structures (records ⇔ planets) (the actual number of array elements) – `n` will be entered from the keyboard.

In addition to the description of variables, we will describe three more procedures with the names `list` (formation of the table header and output of all data in the form of a table on the screen), `vinput` (data entry) and `search` (search for information by planet name).

The width of each table column is determined by the values of the corresponding record fields. However, the width of the field names in the table header and the screen width must also be taken into account. Therefore, when describing `the list function` , it's important to keep in mind the possibility of formatted data output.

Then program Can introduce V next view:

2. Program

```
#include<conio.h>
#include<iomanip>
#include<iostream>
#include<string>

usingnamespace std;

struct
      {
            string planeta;
            string sostav;
            float rasst,period;
            char sputnik;
      } mas[20];
intn;
stringbuff;// variable, used V program For cleaning buffer void list();
```

```
void input();
void search();

int main()
{
      cout<< "Enter quantity planets: " ;
      cin>>n;
      system("CLS");
      vvod();
      list();
      search();
      return 0;
}
voidvvod() // Процедуравводаданных
{
 system("color 17");
 system("CLS");
for (int i=1; i<=n; i++)
{
           system("CLS");
           cout<< "   Enter data For " <<i<< " records :" <<endl;
           getline(cin,buff); // clear the buffer
           cout<< "   Planet name - " ;
           getline(cin,mas[i].planeta);
           cout<< "   Distance from the
           Sun - " ; cin>>mas[i].rasst;
           cout<<"    Периодобращения - ";
           cin>>mas[i].period;
           do
           {
                 cout<<"Наличиеспутника (+,-) - ";
                 char x = getch();
                 if ((x == '+') || (x == '-'))
                           {     cout<<x<<"\n";
                                 mas[i].sputnik=x;
                           }
                      else cout<< "\r" ;
           }
           while ( (mas[i].sputnik!= '+' ) && (mas[i].sputnik!= '-' ));
           cout<< " Main composition of the atmosphere - " ;
           cin>>mas[i].composition;
}
}
//Formation title tables And conclusion data V view tables on void list()
screen
{
      system( "CLS" ); system(
      "color 1f" );
      cout<<endl<<endl<<endl;
      cout<< "                  Table: 'SPACE EXPLORATION'" <<endl;
      system( "color 17" );
      cout<< "     ╔═══╦═══════════╦═══════╦═══ " <<endl;
```

```
	cout<< "     ╟  N  ╟    Name    ╟  Distance Period  ╟Availability  Main composition "
	                                                                          <<endl;
	cout<< "     ╟ No. ╟  planets   ╟from the Sun appeals satellite atmosphere   ╟ <<endl;
	cout<< "     ╟─────╫────────────╫──────────╫─── "" <<endl;
	for ( int i=1; i<=n; i++)
	{
	 cout<<"      ╟  "<<i<<"  ╟"<<setw(10)<<mas[i].planeta<<"
╟"<<setw(7)<<fixed<<setprecision(2)<< mas[i].rasst<<"  ╟"<<setw(7)<<mas[i].period;
	 cout<<" ╟"<<setw(5)<<mas[i].sputnik<<"    ╟"<<setw(15)<<mas[i].sostav<<"╟"<<endl;
	}
	cout<< "     ╚═════╩════════════╩══════════╩═══ " <<endl;
	system( "pause" );
}

void search() // Procedure for searching for information on the name of a planet
{
	string s;
	getline(cin,buff); // clear the buffer
	cout<< " Enter sought after Name planets:" ;
	getline(cin,s);;
	system( "CLS"
	); bool f= true
	;
	cout<<endl<<endl<<endl;
	for ( int i=1; i<=n;
	i++) if
	(mas[i].planeta==s)
	{
		cout<< "    Planet name:              "
		<<mas[i].planeta<<endl; cout<< "      Distance from the Sun:
		                                      " <<mas[i].rasst<<endl;
		cout<< "    Orbital period:           " <<mas[i].period<<endl;
		cout<< "    Presence of a satellite:  "
		<<mas[i].sputnik<<endl; cout<< "      Main composition of the
		atmosphere: " <<mas[i].socomposition<<endl; f= false ;
	}
	if (f) cout<< "  There is no such planet! " <<endl;
	system( "pause" );
}
```

When drawing table frames, you can use symbols pseudographics (Alt+code) corresponding symbol V code table ASCII typed using the numeric keypad).

The program should result in a table similar to the one shown above.

15. Files

File they call way storage information on physical device. A file is a concept that applies to everything from a file on a disk to a terminal. The C and C++ languages have different capabilities for working with file streams. The C language does not have operators for working with files. All necessary actions are performed using functions included in the standard library. They allow work With various devices, such, How disks, printer, communication channels And etc. These devices are very differ Friend from friend. However file system transforms their V united abstract logical device, ***called stream*** .

Unlike C, C++ handles file streams using stream classes based on the **<ifsream> and** <ofstream> **libraries . Let 's** look at each of these methods in more detail **.**

15.1. Approach With using opportunities language Si

IN Si exists **two type streams** : *text* (**`text`**) And *binary* (**`binary`**).

A text stream is a sequence of characters. When characters from the stream are transferred to the screen, some are not displayed (for example, carriage returns and line feeds).

A binary stream is a sequence of bytes that uniquely correspond to what is on the external device.

Before reading or writing information to a file, it must be opened and thus associated with a stream. This can be done using the **`fopen () library function`** . She beret external performance file - it complete The physical name, or, in other words, the specification (for example, `c:\my_prog.txt`), is associated with an internal logical name, which is used later in the program. **The logical name** is a pointer to the required file. It must be defined; this is done, for example, like this:

```
FILE *fp;
```

Here `FILE` is the type name defined in the standard header file `stdio.h` , `fp` is the file pointer. Calling the `fopen() function` in the program it is implemented by the expression:

```
fp = fopen("file_specification", "file_use_method");
```

Specification file (those. physical Name file And path To to him) Maybe, for example, it should look like this : `"c:\\my_prog.txt"` – for the file `my_prog.txt` on disk `c:` .

Way use file is being asked next symbols:

- `r` – (from read) open existing file For readings;
- `w` – (from write) create new file For records (If file With specified the name exists, then it will be overwritten);

- `a` – (from add) to append to a file (open an existing file to write information starting from the end of the file, or create a file if it does not exist).

Type file the flow is specified by the key `t` (text) or `b` (binary). If the mode `t` or `b` is not specified (for example, `r` , `w` , or `a`), then it is determined by the value of the global variable `_fmode` . If `fmode=0_BINARY` , then files are opened in binary mode, and if `_fmode=0_TEXT` , then in text mode. Constants `0_BINARY and 0_TEXT` are defined in the `fcntl.h file` .

Table 9 -Full table permissible modes

"r"	Opens file For readings. (Opens By default How text file.)
"w"	Creates file For records. (Opens By default How text file.)
"A"	Attaches To file. (Opens By default How text file.)
"rb"	Opens binary file For readings.
"wb"	Opens binary file For records.
"ab"	Attaches To binary file.
"r+"	Opens file For read/write. (Opens By default How text file.)
"w+"	Creates file For read/write. (Opens By default How text file.)
"a+"	Attaches or creates file For read/write. (Opens By default as a text file.)
"r+b"	Opens binary file For read/write.
"w+b"	Creates binary file For read/write.
"a+b"	Attaches or creates binary file For read/write.
"rt"	Opens text file For readings.
"wt"	Creates text file For records.
"at"	Attaches To text file.
"r+t"	Opens text file For readings.
"w+t"	Creates text file For read/write.
"a+t"	Opens or creates text file For read/write.

`the fopen()` function If an error occurs, it returns a pointer to the NULL constant. Recommended use next way discoveries file:

```
if ((fp = fopen("c:\\my_prog.txt", "rt")) == NULL) {
   puts("Failed to open file\n")
   return 0;
}
```

After endings work With file He must be closed. This is being done using the library function `fclose()` . It has the following prototype:

```
int fclose(FILE *fp);
```

If the operation is successful, the function `fclose()` returns a value of zero. Any other value indicates an error.

Let's consider standard functions for working with **files sequential access:**

1. Function putc() writes down symbol V file And has next prototype:

```
int putc( int c, FILE *fp);
```

Here fp – a pointer to the file returned by the fopen() function, - symbol to write (variable with has the **int type** , but only the low byte is used). If putc() completes successfully, returns the written character, otherwise the constant EOF is returned . It is defined in the stdio.h file. and has a value of (-1).

2. Function getc() reads symbol from file And has next prototype:

```
int getc(FILE *fp);
```

Here fp is a pointer to the file returned by the fopen() function. This function returns the character read. The corresponding value is of type **int ,** but the high byte is zero. If the end of file is reached, getc returns EOF.

3. The feof() **function** determines the end of file when reading binary data and has the following prototype:

```
int feof(FILE *fp);
```

Here, fp is a file pointer returned by the fopen() function. A nonzero value is returned when the end of the file is reached; otherwise, 0 is returned.

4. fputs() **function** writes a string of characters to a file. It differs from the function puts() only in that the second parameter must be a pointer to a file type variable.

For example:

```
fputs("Example", fp);
```

At occurrence mistakes is coming back meaning EOF.

5. The fgets() **function** reads a string of characters from a file. It differs from the gets( ) The second parameter must be the maximum number of characters to be entered plus one, and the third parameter must be a pointer to a file-type variable. The string is read in its entirety if its length does not exceed the specified number of characters; otherwise, the function returns only the specified number of characters. Let's look at an example:

```
fgets(string, n, fp);
```

The function returns a pointer to the string string upon successful completion and the constant NULL. in case of an error or reaching the end of the file.

Functions, used For work With **files direct access** (all they are described in the stdio.h file):

1. Function fseek() allows fulfill reading And recording With random access and has the following prototype:

```
int fseek( FILE *fp, long count, int access);
```

Here fp - a pointer to the file returned by the fopen() function, count is the byte number relative to the specified starting position, starting from which the operation will be performed, access is the method for specifying the starting position.

Variable access can accept next values:

0 - initial position given V beginning file;

1 - initial position counts current;

2 - initial position given V at the end file.

At successful completion is coming back zero, at error - non-zero value.

2. Function ferror() allows check right execution the last operation when working with files. It has the following prototype:

```
int ferror( FILE *fp);
```

IN case mistakes is coming back non-zero meaning, V otherwise In this case, zero is returned.

3. Function remove( ) deletes file And has next prototype:

```
int remove( char *file_name);
```

Here, file_name is a pointer to a string containing the file specification. Zero is returned on success; otherwise, a non-zero value is returned.

4. The rewind() **function** sets the current position pointer to the beginning of the file and has the following prototype:

```
void rewind( FILE *fp);
```

5. fread() **function** is used to read blocks of data from a stream. It has the following prototype:

```
unsigned fread( void *ptr, unsigned size,
                unsigned n, FILE *fp);
```

It reads n data elements, each size bytes long, from the given input stream fp into the block pointed to by ptr . The total number of bytes read is equal to n * size . Upon successful completion, the fread() function returns the number of data elements read, on error – 0.

6. fwrite() **function** is used to write blocks of data to a file. It has a prototype:

```
unsigned fwrite( void *ptr, unsigned size,
                 unsigned n, FILE *fp);
```

It appends n data elements, each size bytes long, to the given output file fp . The data is written starting at the position pointed to by the pointer ptr. If the operation completes successfully, the fwrite() function returns the number of data elements written, or, on error, an invalid number of data elements.

15.2. Approach With using opportunities language C++

A stream is an abstract concept that refers to any transfer of data from a source to a destination. C++ streams, in contrast, C-style input/output functions are based on the concept of a class. Let's consider the procedure for using C++ file streams.

To work with files in C++, you need to include the fstream header file:

```
#include < fstream >
```

File flow For readings opens next way:

```
ifstream stream_id(file_name,stream_keys);
```

For example:

```
ifstream input("input.txt", ios ::in);
```

Here, a file stream with the identifier `input is opened` for reading from a file. Such a file stream must be closed, as in C: `input.close();`

File flow For records opens So:

```
ofstream stream_id(file_name,stream_keys);
```

For example:

```
offstream output("output.txt", ios
::out);
```

This stream should also be closed:

```
output.close();
```

Table 10 — modes discoveries files

Constant	Description
ios_base::in	open file For readings
ios_base::out	open file For records
ios_base::ate	at opening move pointer to the end file
ios_base::app	open file For records V end file
ios_base::trunc	delete content file, If He exists
ios_base::binary	opening file V binary mode

How same work With such streams? Very simple:

```
int main(){
   int  n,k;
   float f;
   char str[100];
   ifstream input("input.txt",ios::in);
```

```
   ofstream output("output.txt",ios::out);
   input>>n;
   input>>k;
   n=n+k;
   output<<n<<endl;
   input>>f;
f=f+2;
output<<f<<endl;
input>>str; output<<str<<", ";
input>>str; output<<str<<"!!";
   input.close();
   output.close();
   return 0;
}
```

If original contents file `input.txt` was following:

```
5
10
- 5.56783345
Helloworld!
```

That V file `output.txt` the following will be written :

```
15
- 3.56783
Hello, peace!!!
```

Laboratory work No. 15 Research methods access To files data

TARGET WORKS: improvement skills V programming With using pointers, exploring sequential and direct file access functions.

EXECUTION WORKS: V in accordance With option make up And implement programs.

Exercise I

Job With unstructured data

To explore different methods of accessing data files, the following preparatory steps must be completed:

1. Create V Notepad next text file:

```
U me asked: How many will x Oper y ? I
don't know! And n Oper k ? Neither!
Help!
```

For example:

```
U me asked: How many will 7 * 2 ? I
don't know! And 9/4? Neither!
Help!
```

2. To you known structure file. Withdraw content file on screen, A write the results to the output file:

```
x Oper y = Res1
n Oper k = Res2
```

For example:

```
7 * 2 = 14
9 / 4 = 2.25
```

3. Initial data are taken from tables according to options:

Var.	x	Oper	y	n	Oper	k	Var.	x	Oper	y	n	Oper	k
1	15	+	4	7	*	8	***9***	23	+	37	13	*	5
2	18	-	19	18	/	4	***10***	7	-	42	37	/	6
3	9	*	6	56	-	37	***11***	34	*	3	14	-	53
4	23	/	5	31	+	29	***12***	21	/	5	11	+	77
5	7	+	23	14	/	4	***13***	12	+	25	20	/	6
6	34	-	67	11	*	3	***14***	15	-	72	54	*	2
7	21	*	2	20	+	11	***15***	18	*	4	7	+	55
8	12	/	5	54	-	32	***16***	9	/	18	18	-	81

4. Explore the possibility of solving this problem using two methods :
   - approach language Si;
   - file streams C++.

5. Draw conclusions about the advisability of using one or another method of reading or writing data to a file.

Exercise II

Job with structured data

1) Using the program obtained during laboratory work 14, implement the ability to save data to a file and then read the entered data from the file.
2) Explore the possibility of solving this problem using one of two methods:
   - approach language Si;
   - file streams C++.
3) Draw conclusions about the advisability of using one or another method of reading or writing structured data to a file.

Exercise III

IN in accordance With option write And debug program:

1. Information about four students is given. The structure is as follows: last name, year of birth, place of birth, faculty. Display information about the oldest student.

2. Information on six savings bank depositors is provided. The structure is as follows: last name, district, social status, and deposit amount. Display data on farmers. Calculate their number.

3. Information on five schools is provided. The structure is as follows: school number, year, number of graduates, and number of university applicants. Display data for the school with the highest ratio of applicants to graduates.

4. Information about five students is provided. The structure is as follows: last name, first name, patronymic, and four exam grades. Display data about students who passed the session with excellent grades.

5. Information about five workers in a workshop is given. The structure is as follows: last name, gender, salary, and length of service. Display information about the worker with the highest salary.

6. Information about four universities is given. The structure is as follows: university name, number of students, number of faculties, number of departments. Display the total number of students.

7. Information about five students is given. The structure is as follows: last name, first name, patronymic, and four exam grades. Display data on students who passed the session without failing.

8. Information on six Sberbank depositors is given. The structure is as follows: last name, district, social status, and deposit amount. Find out how many of the depositors are employees. Calculate their total deposit.

9. Dana information about five patients. The structure has type: surname, age, Gender, blood pressure. Display data on patients with high blood pressure (over 140) and calculate their number.

10. Information about five apartments is provided. The structure is as follows: owner's name, area, number of rooms, floor. Display information about apartments with an area less than 30 square meters and count their number.

11. Information about six students is provided. The structure is as follows: last name, year of birth, department, and year. Display information about students aged 17 to 19 and their number.

12. Information on six bank depositors is provided. The structure is as follows: last name, district, social status, and deposit amount. Display data on depositors in the district entered using the keyboard.

13. Information about six students is given. The structure is as follows: last name, height, weight, year of birth. Display data about the students with the shortest height and a weight between 65 and 75 kg.

14. Information is provided on five dormitory rooms. The structure is as follows: faculty, number rooms and her square, quantity living in her students. Display data about the room with the smallest area per student.

15. Information on six savings bank depositors is provided. The structure is as follows: last name, district, social status, and deposit amount. Display information on the depositor with the largest deposit amount.

16. Information on five shop floor workers is provided. The structure is as follows: last name, gender, salary, and length of service. Display information about the worker with the longest length of service.

17. Information on five athletes is provided. The structure is as follows: last name, gender, year of birth, and height. Display information about the tallest athlete over 20 years old.

18. Information on five publications is provided. The structure is as follows: name of publication, newspaper, or magazine, price of one copy, and number of copies per year. Print information on the least expensive publication.

19. Information on six of the bank's investors is provided. The structure is as follows: last name, district, social security number. Position and amount of contribution. Display data on the total contribution of investors in a given area.

20. Information about six students is given. The structure is as follows: last name, height, weight, hair color. Display data about the blondes with the highest height.

21. Information about four universities is given. The structure is as follows: university name, number of students, number of faculties, number of departments. Display data about the university with the largest number of students.

22. Information on five musical instruments is provided. The structure is as follows: instrument name, date, month, and year of manufacture. Print information about the instrument with the longest lifespan.

23. Information about five copies of a structure is provided. The structure is as follows: newspaper or magazine, publication title, cost per copy, number of copies per year. Print information about the annual subscription price for all newspaper publications.

24. Information about five workers in a workshop is given. The structure is as follows: last name, gender, salary, and length of service. Display information about the man with the highest salary.

25. Information about five books is given. The structure is as follows: book title, author, price, and number of pages. Display information about a given book (by title and author) with the minimum price.

26. The file contains information about five people in the phone book. The structure is as follows: last name, phone number, date of birth. Add information about three new contacts to the end of the list.

27. The file contains information about five Russian universities. It is structured as follows: university name, number of students, number of departments. Append information about multiple universities to the end of the file.

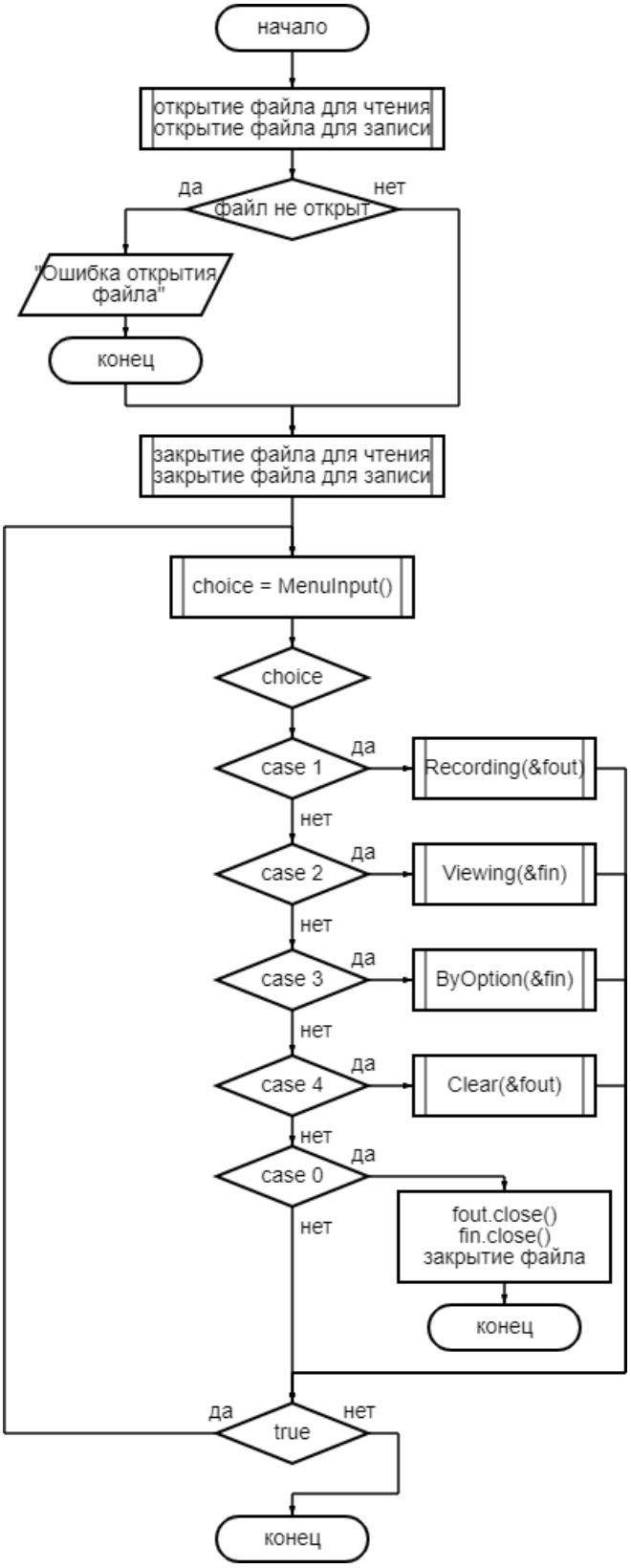
Control questions

1. Concept file V C/C++.
2. Classification files.
3. What the type can be components file?
4. Need to whether at definition file in advance indicate his length?
5. How is different file from array?
6. Concept O logical And physical name file.
7. What understood under readingfrom files What under recording V file?
8. Types access To files.
9. What such consistent file?
10. What is it? general form creation file (C)?
11. What is it? general form readings file (C)?
12. How is being carried out processing files With using approach V style C++ language ?
13. List standard functions For work With **files sequential access**
14. List library functions, used For work With **direct access files.**

Example execution tasks I laboratory work

Task : Information about five apartments is given. The structure is as follows: owner's last name, area, floor, and district. Display data on apartments on a given floor level whose area is greater than min but less than max. The user enters the min, max, and lvl values using the keyboard.

1. Algorithm.



Description used V program functions.

`int MenuInput();`	Organizes input teams V menu

`void PrintMenu();`	Outputs V console menu
`void PrintHeadTable();`	Outputs V console top Part (hat) tables
`void PrintEndTable();`	Outputs V console lower Part tables
`void Print( Apartment * ap , int * counter );`	Outputs a string with information about the structure ap. counter - counter, necessary For numbering structures V table
`void Recording( ofstream * file );`	Organizes recording information O structures from consoles to file
`void Viewing( ifstream * file );`	Organizes conclusion information about structures from file V console
`void Clear( ofstream * file );`	Produces cleaning file
`bool Check_correctly( int x );`	Checks right introduced V menu teams
`void ByOption( ifstream * file );`	Outputs information O records, corresponding task V option
`void set_color( int text , int background = 0);`	Installs color text for console
`void write_colored_text( string text , int color , int background = 0);`	Outputs on screen colored text

2. Program

```
#include <iostream>
#include <iomanip>
#include <string>
#include <fstream>
#include <conio.h>
#include <windows.h>

using namespace std;

// description ours structures
struct Apartment {
	char surname[20] {}; // surname owner
	double square = 0; // area
	int level = 0; // floor
	char district[25] {}; // район
};

int MenuInput();
void PrintMenu();
void PrintHeadTable();
void PrintEndTable();
void Print(Apartment* ap, int* counter);
void Recording(ofstream* file);
void Viewing(ifstream* file);
void Clear(ofstream* file);
bool Check_correctly(int x);
void ByOption(ifstream* file);


// function, which sets necessary color For text (very cool)
```

```
void set_color(int text, int background = 0) { // Installation flowers For text And his
	background HANDLE hStdOut = GetStdHandle(STD_OUTPUT_HANDLE);
	SetConsoleTextAttribute(hStdOut, (WORD)((background << 4) | text));
}

// function, output on screen colored text
void write_colored_text(string text, int color, int background = 0) {
	// change the text color
	set_color(color, background);
	// we are deducing
	this text cout <<
	text;
	// we are installing back color text By default (white)
	set_color(15, background);
}

// mainfunction
intmain() {
	// we do setting For work With Russian language
	setlocale(LC_ALL, "Rus");
	SetConsoleCP(1251);
	SetConsoleOutputCP(1251);

	offstream fout("example.dat", ios::out | ios::app);
	ifstream fin("primer.dat", ios::in | ios::app);

	if (!fout.is_open() or !fin.is_open()) {
		cout << "Ошибка открытия файла!\n";
		exit(0);
	}
	fout.close();
	fin.close();

	short choice;
	do {
		choice = MenuInput();
		switch (choice) {
		case 1:
			Recording(&fout);
			break;
		case 2:
			Viewing(&fin);
			break;
		case 3:
			ByOption(&fin);
			break;
		case 4:
			Clear(&fout);
			break;
		case 0:
			fout.close();
			fin.close();
			return EXIT_SUCCESS;
		}
	} while (true);

	system("pause");
	return 0;
}

// outputs V console "hat" ours tables void
PrintHeadTable() {
	cout << "                Table \"INFORMATION ABOUT APARTMENTS " << endl;
```

```
	cout                                                    <<                                                                  "
_____________________________________________________________________________" << endl;
	cout << "            |       |                         |                |             |
|" << endl;
	cout << "          |  No.  |    Last name    |  Area       |    Floor   |          District
|" << endl;
	cout                                                    <<                                                                  "
| ____| _________________| ___________| _________| ____________________|" << endl;
}

// outputs V console lower "closer" ours tables void
PrintEndTable() {
	cout                                                    <<                                                                  "
| ____| _________________| ___________| _________| ____________________|" << endl;
}

// prints line With information O structure [ap]
void Print(Apartment* ap, int* counter) {
	cout << "         |  "
		<< left << setw(3) << *counter << "|  "
		<< left << setw(15) << ap->surname << "|  "
		<< left << setw(10) << ap->square << "|  "
		<< left << setw(8) << ap->level << "|  "
		<< left << setw(18) << ap->district << "|\n";
	(*counter)++;

}

// печатаетцветноеменю
void PrintMenu() {
	write_colored_text("\n                    MENU\n", 2);
	write_colored_text(" • [1] Adding an owner \n", 14);
	write_colored_text(" • [2] Viewing owner information \n", 14);
	write_colored_text(" • [3] Search data By floor And area \n", 14);
	write_colored_text(" • [4] Clearing file\n", 14);
	write_colored_text(" • [0] Exit \n", 14);
	write_colored_text("Select an item: ", 15);
}

// organizes the introduction of a
command (numbers 0-4) int MenuInput()
{
	PrintMenu();
	int m;

	do {
		cin >> m;

		// examination on happening, If introduced
		NOT number if (cin.fail()) {
			cin.clear();
			cin.ignore(32767, '\n');

			system("cls");
			PrintMenu();
			write_colored_text("\nYou entered a non-number! Repeat attempt: \n",
			4); continue;
		}

		// If introduced number, But Not correct, we write
		about in this if (!Check_correctly(m)) {
			system("cls");
			PrintMenu();
```

```
                    write_colored_text("\nThe entered number is invalid.  Repeat  attempt:  \n",
4);
                    continue;
             }
             return m;

      } while (true);
      return m;
}

// function checks on right introduced teams (What would be included V interval 0-4)
bool Check_correctly(int x) {
      switch (x) {
      case 1:
      case 2:
      case 3:
      case 4:
      case 0:
             return true;
      default:
             return false;
      }
}

// function entries(input) fields
structures void Recording(ofstream*
file) {
      file->open("example.dat", ios::out | ofstream::app);
      system("cls");

      cout << "Get started data entry\n[Esc] - Completion data entry\nUse This after completing
the data entry into the structure.\n\n";
      Apartment home;
      do {
             cout << Owner's name: "; cin
             >> home.surname;
             cout << "Apartment area: ";
             cin >> home.square;
             cout << "Floor: ";
             cin >> home.level;
             cout << "District:
             ";
             // we delete With flow input symbol new lines
             cin.ignore(1024, '\n');
             gets_s(home.district);

             // we're writing it down all ours sturku V file
             file->write((char*)&home, sizeof(Apartment)); write_colored_text("Data
             successfully recorded! [Esc] - exit.\n", 2);

      } while (_getch() != 27); // 27 - кодкнопки Esc

      file->close();
      system("cls");
}

// is browsing And outputs information O everyone structures V file V view
void Viewing(ifstream* file) tables {
      file->open("example.dat", ios::in);
      system("cls");

      // If file empty, we are deducing message
      about in this if (file->peek() == EOF)
             write_colored_text("\nFile is empty!\n\n", 4);
```

```
	else {
		Apartment home{};
		PrintHeadTable();

		// counter for numbering structures in
		the table int counter = 1;
		while (file->read((char*)&home, sizeof(Apartment))) {
			Print(&home, &counter);
		}
		PrintEndTable();
	}

	file->close();
	system("pause");
	system("cls");
}

// cleanses file from all
information void Clear(ofstream*
file) {
	// cleaning is happening thanks to flag ::trunc
	file->open("example.dat", ofstream::out | ofstream::trunc);
	system("cls");

	write_colored_text("\nFile successfully cleared!\n\n", 2);

	file->close();
	system("pause");
	system("cls");
}

// performs conclusion information O records, corresponding tasks options
void ByOption(ifstream* file) {
	file->open("example.dat", ios::in | ios::app);
	system("cls");

	// If file empty, we are deducing message about
	in this if (file->peek() == EOF)
		write_colored_text("\nFile is empty!\n\n", 4);
	else {
		int LVL; // sought after floor
		double min, max; // interval sought after area rooms

		write_colored_text("Enter below floor And interval area, By which need to
search\n\n", 14);
		write_colored_text("Enter floor: ", 15);
		cin >> LVL;
		write_colored_text("Enter the lower bound area: ", 15);
		cin>>min;
		write_colored_text("Enter the upper limit area: ", 15); cin
		>> max;

		// flag, inspector, There is whether Although would 1 office, suitable
		under our conditions bool flag = false;
		// counter For numbering structures V table
		int counter = 1;
		Apartment home{};
		while (file->read((char*)&home, sizeof(Apartment))) {
			if (LVL == home.level && home.square > min && home.square < max) {
				if (!flag)
					PrintHeadTable();

				Print(&home, &counter);
```

```
                        flag = true;
                }
        }
        if (flag)
                        PrintEndTable();
        else
                write_colored_text("\nThere are no such entries!\n\n", 4);
}

file->clear();
file->seekg(0, ios::beg);
file->close();
system("pause");
system("cls");
}
```

Testing programs 15 laboratory

Entrance data:

Surname	**Square**	**Floor**	**District**
Chernyshev	40	3	Chernyshevsky
Lazarev	25	1	Lazarevsky
Volkova	30	1	Volkovsky
Borisova	15	2	Borisovsky
Eremin	50	1	Ereminsky

Testing

1. **Data entry**

```
Начинайте ввод данных
[Esc] - Завершение ввода данных
Используйте это после завершения внесения данных в структуру.

Фамилия владельца: Чернышев
Площадь квартиры: 40
Этаж: 3
Район: Чернышевский
Данные успешно записаны! [Esc] - выход.
Фамилия владельца: Лазарев
Площадь квартиры: 25
Этаж: 1
Район: Лазаревский
Данные успешно записаны! [Esc] - выход.
Фамилия владельца: Волкова
Площадь квартиры: 30
Этаж: 1
Район: Волковский
Данные успешно записаны! [Esc] - выход.
Фамилия владельца: Борисова
Площадь квартиры: 15
Этаж: 2
Район: Борисовский
Данные успешно записаны! [Esc] - выход.
Фамилия владельца: Еремин
Площадь квартиры: 50
Этаж: 1
Район: Ереминский
Данные успешно записаны! [Esc] - выход.
```

2. Conclusion information on screen

```
                    Таблица "ИНФОРМАЦИЯ О КВАРТИРАХ"
┌──────┬──────────────┬──────────┬──────┬──────────────┐
│  №   │   Фамилия    │ Площадь  │ Этаж │    Район     │
├──────┼──────────────┼──────────┼──────┼──────────────┤
│  1   │ Чернышев     │ 40       │ 3    │ Чернышевский │
│  2   │ Лазарев      │ 25       │ 1    │ Лазаревский  │
│  3   │ Волкова      │ 30       │ 1    │ Волковский   │
│  4   │ Борисова     │ 15       │ 2    │ Борисовский  │
│  5   │ Еремин       │ 50       │ 1    │ Ереминский   │
└──────┴──────────────┴──────────┴──────┴──────────────┘
Для продолжения нажмите любую клавишу . . .
```

3. Testing search

Testing search is happening V 6 stages V in accordance With table tests

Table tests:

No. test	Floor search	Range area For search
1	1	0-30
2	1	30-60
3	2	0-30
4	2	30-60
5	3	0-30

6	3	30-60

Expected result

1 test - data output O 2 And 3 apartment

2 test - conclusion data O 5 apartment

3 test - conclusion data O 4 apartment

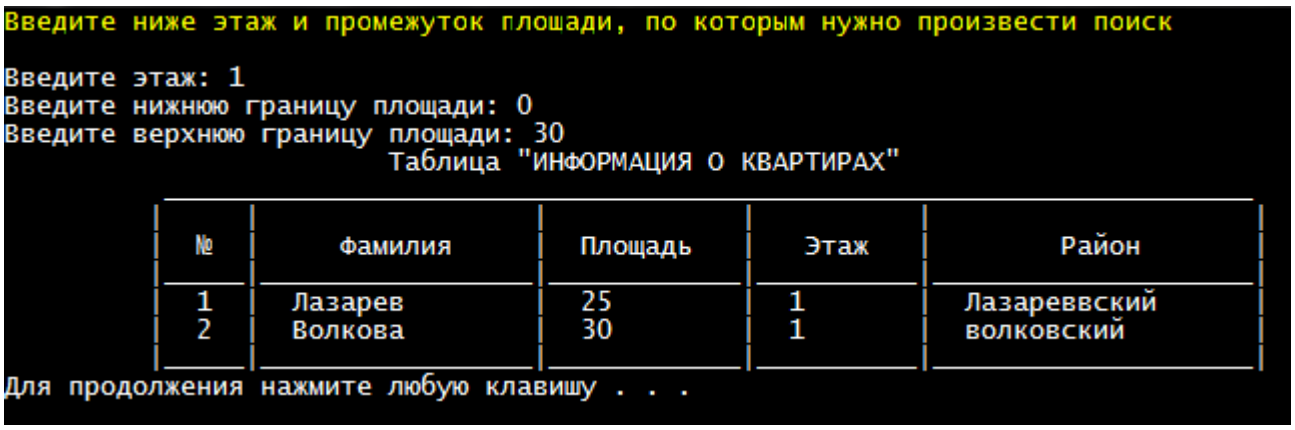
4 test - absence data, satisfying search

5 test - absence data, satisfying search

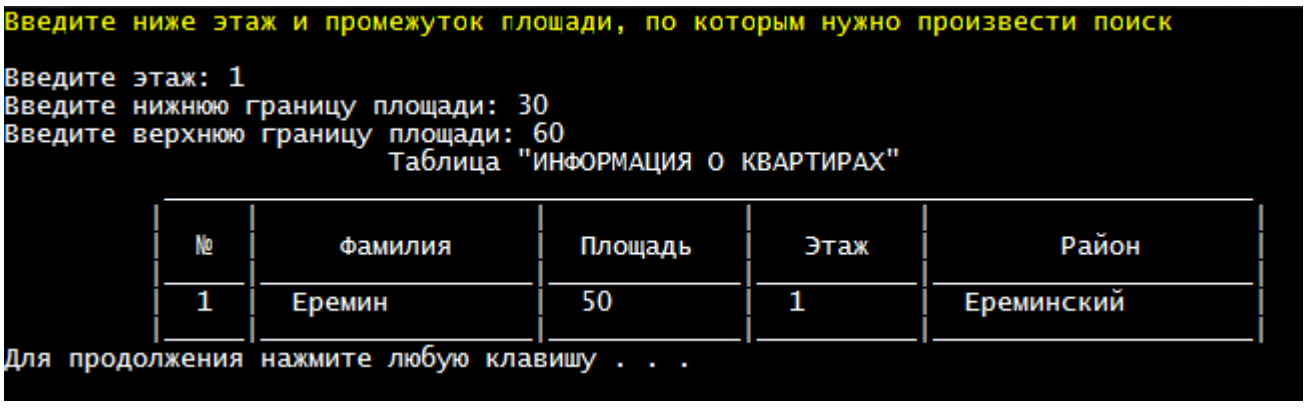
6 test - conclusion data O 1 apartment

Result testing:

1 test:



2 test:



3 test:

```
Введите ниже этаж и промежуток площади, по которым нужно произвести поиск

Введите этаж: 2
Введите нижнюю границу площади: 0
Введите верхнюю границу площади: 30
                        Таблица "ИНФОРМАЦИЯ О КВАРТИРАХ"
┌──────┬──────────────────┬──────────────┬──────────┬──────────────────────┐
│  №   │     Фамилия      │   Площадь    │   Этаж   │        Район         │
├──────┼──────────────────┼──────────────┼──────────┼──────────────────────┤
│  1   │ Борисова         │ 15           │ 2        │ Борисовский          │
└──────┴──────────────────┴──────────────┴──────────┴──────────────────────┘
Для продолжения нажмите любую клавишу . . .
```

4 test:

```
Введите ниже этаж и промежуток площади, по которым нужно произвести поиск

Введите этаж: 2
Введите нижнюю границу площади: 30
Введите верхнюю границу площади: 60

Таких записей нет!

Для продолжения нажмите любую клавишу . . .
```

5 test:

```
Введите ниже этаж и промежуток площади, по которым нужно произвести поиск

Введите этаж: 3
Введите нижнюю границу площади: 0
Введите верхнюю границу площади: 30

Таких записей нет!

Для продолжения нажмите любую клавишу . . .
```

6 test:

```
Введите ниже этаж и промежуток площади, по которым нужно произвести поиск

Введите этаж: 3
Введите нижнюю границу площади: 30
Введите верхнюю границу площади: 60
                        Таблица "ИНФОРМАЦИЯ О КВАРТИРАХ"
┌──────┬──────────────────┬──────────────┬──────────┬──────────────────────┐
│  №   │     Фамилия      │   Площадь    │   Этаж   │        Район         │
├──────┼──────────────────┼──────────────┼──────────┼──────────────────────┤
│  1   │ Чернышев         │ 40           │ 3        │ Чернышевский         │
└──────┴──────────────────┴──────────────┴──────────┴──────────────────────┘
Для продолжения нажмите любую клавишу . . .
```

4. Testing buttons "Cleaning file»:

```
Файл успешно очищен!

Для продолжения нажмите любую клавишу . . .
```

5. Examination cleaning

```
Файл пуст!

Для продолжения нажмите любую клавишу . . .
```

6. Exit from programs:

```
                МЕНЮ
 • [1] Добавление владельца
 • [2] Просмотр данных о владельцах
 • [3] Поиск данных по этажу и площади
 • [4] Очистка файла
 • [0] Выход
Выберите пункт: 0

C:\Users\Lisa\source\repos\Project3\Debug\Project3.exe (процесс
Чтобы автоматически закрывать консоль при остановке отладки, вкл
и остановке отладки".
Нажмите любую клавишу, чтобы закрыть это окно…
```

```
                МЕНЮ
 • [1] Добавление владельца
 • [2] Просмотр данных о владельцах
 • [3] Поиск данных по этажу и площади
 • [4] Очистка файла
 • [0] Выход
Выберите пункт: 0

C:\Users\Lisa\source\repos\Project3\Debug\Project3.exe (процесс
Чтобы автоматически закрывать консоль при остановке отладки, вкл
и остановке отладки".
Нажмите любую клавишу, чтобы закрыть это окно…
```

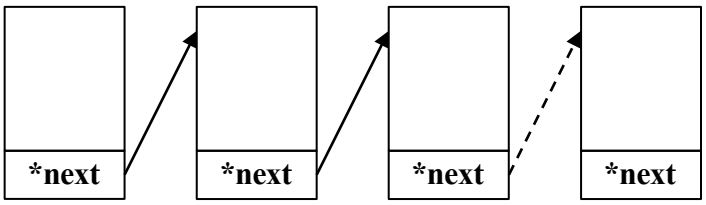
16. List

16.1 Study formations elements related list

A linked list is the simplest data type of a dynamic structure consisting of elements (nodes). Each node typically contains two fields:

- data (data can be a variable, a class object or structures, etc.)
- pointer on next knot V list.

Each element of a list contains *a key* that identifies that element. The key is usually either an integer or a string and is part of fields data. Different parts of a data field can act as a key when working with a list. For example, if a linear list is created from records containing last name, year of birth, length of service, and gender, any part of the record can act as a key: when sorting the list alphabetically, the key would be the last name, and when searching, for example, for veterans of labor, the key would be length of service. The keys of different list elements can be the same.



Drawing 16.1 - Unidirectional list

```
// description structures, allowing create linear list struct
List{
   // -- data fields --
   char name[20]; // name
   intage;            // age
   // -- connection fields -- -
   List *next;    // pointer to the next
}; //          list element
List *start;       // start - special global
                   // pointer to enter the list
                   // for an empty list should
                   // be equal to zero
```

For this list defined three operations:

- addition new element list **`Add() ;`**
- removal element from list **`Del() ;`**
- transition To to the next one element list **`Next() .`**

Let's take a look at them.

16.2 Study operations With elements related lists

1. Addition new element list **`Add() .`**

```
// function for adding elements to the list
```

```
List* Add(List *cur)    { // cur - current value
   // allocate memory for the new list element
   List *p=(List*)malloc(sizeof(List)); // p - new element if (!p)
   { // If highlight memory Not succeeded
              cout<<" Error !";
              return 0;
   }
   if(!start) { // if the list is empty
              start=p;   // That new element will first
              p->next=NULL;    // and last in the list
              returnp;
   }
   // if the list is not empty, then add the element
   // after the current one and bind the new element to
   // subsequent
   p->next=cur- >next;
   cur->next=p;     // we connect current element With
   new returnp; // return a pointer to the new element
}
```

Dynamic creation of new list elements is performed using the standard memory management function **malloc() :**

```
List *p=(List*)malloc(sizeof(List));
```

Let's take a closer look at this operator. **The sizeof(List) function** determines the size structures **List** in bytes. **The malloc() function** allocates a memory block of this size. The result of this function is converted to a pointer to a structure type. **List - (List*) ,** That There is returns pointer **p** type **List** structure address of the first byte of allocated memory.

2. Removal element from list **Del() .**

```
// function to remove a list element
voidDel(List *cur) {// cur - pointer to the element to be removed
// create a pointer to move and
// we are installing on
   start List *p=start;
   if (cur==start) {// if we delete the first element
              of the list free(start);   // free up
              memory start=start->next;// second we do
              first return ;
   }
   // execute until we reach the end of the list
   // or previous current while
   (p&&p->next!=cur)
                   p=Next(p); // move on to the next
   if(!p)     {// we arrived to end Not having found
   previous
              cout<< "Not found element";
              return ;
   }
   // we connect previous With
   subsequent p->next=cur->next;
   // we are freeing memory current
   free(cur);
```

```
    return;
  }
```

3. Transition To to the next one element list **Next()** .

```
// function transition To to the next one
element list List* Next(List *cur)      {
  // we return pointer on next element list return
(cur->next);
}
```

Laboratory work No. 16 Research related lists data

TARGET WORKS:

- Improvement skills developments programs V environment programming
- Improvement skills V programming With using pointers
- Study process formations elements related list
- Study operations With elements related lists

EXECUTION WORKS: V in accordance With option make up And Implement programs. Time to complete the work: 6 hours

Exercise I

To investigate the operating principles of the operations described in the theoretical part and, based on this, to develop graphic diagrams reflecting the operations defined for the list.

Write a program that allows you to perform work using the menu and the operations described above. With a linked list, inputting data into the list from a file, outputting the entire list, and processing the data stored in the list while solving the problem according to the variant, and then dumping the list to a file. A solution without files is possible, but the maximum rating for the result is "good."

Exercise II

IN in accordance With option write And debug program, using dynamic data structure: Singly linked list.

1. Recordings contain last name And 4 assessments. Delete from list underachievers.
2. Recordings contain last name And 4 assessments. Delete having 2, 3.
3. Recordings contain Name books, author, year publications. Delete publications with a year less than the specified one.
4. Recordings contain Name books, author, year publications. Delete books specified author.
5. Recordings contain Name, price, quantity goods. Delete records With product of a given name.

6. Records contain the name, price, and quantity of the product. Delete product batches exceeding the specified value.

7. Entries contain last name and year of birth. Add new entries so that the list is sorted alphabetically.

8. Entries contain last name and year of birth. Add new entries so the list is sorted by age.

9. Entries contain the title of the publication, newspaper or magazine, and the price per copy. Add new entries so that magazines appear first, then newspapers.

10. Entries contain the title of the publication, newspaper or magazine, and the price per copy. Add new entries so that the titles are sorted alphabetically.

11. Records contain the athlete's name, sport, and point total. Add new records so that information for each sport is arranged sequentially.

12. Records contain the athlete's last name, sport, and point total. Add new records so they are sorted by descending points total.

13. The records contain the product name, price, and quantity. Sort the list items in descending order by price using direct sorting.

14. In the solution to the previous problem (option 14), when sorting a list, exchanging two list elements is implemented as exchanging information parts. In real-world problems, exchanging information parts is inefficient due to their large size. Therefore, it is more efficient to reset the pointers in the list without affecting the information parts. Based on this observation, implement a procedure that sorts list elements in descending order.

15. The entries contain the book title, author, and year of publication. Merge the two alphabetically ordered lists into one (also alphabetically ordered), creating a new list.

16. Given an ordered list of book titles, we need to add information about a new book while maintaining the order of the list.

17. Let's assume we're given a list of students. Each list element contains: last name, first name, patronymic, year number, group number, and grades for the five exams of the last session. Assuming the students' names in the list are sorted alphabetically, delete the sublist of students starting with the last name Petrov and ending with the last name Sidorov, inclusive.

18. Let's assume we're given a list of students. Each list element contains: last name, first name, patronymic, year number, group number, and grades for the five exams of the last session. Assuming the student information in the list is sorted by year number, move the second-year students to the end of the list.

19. Consider a list of students. Each element of the list contains: last name, first name, patronymic, year number, group number, and grades for the five exams of the last session. Delete information about fifth-year students and promote the remaining students to the next year.

20. Let's assume a list of students. Each element of the list contains: last name, first name, patronymic, year number, group number, and grades for the five exams of the last session. Print information about the accrued stipend, assuming that excellent students are paid S1, students with only Bs and A's are paid S2, and students with Cs are not paid a stipend.

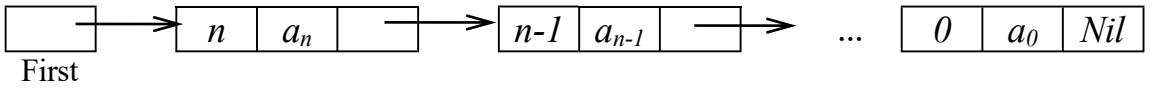
21. Let's say a file stores records with information about cars and their owners (make, traffic police number, owner's last name). For each car make, we need to create a list containing the last names of the owners of cars of that make.

22. Let's say a file stores records with information about cars and their owners (make, traffic police number, owner's last name). For each owner's last name, we need to create a list containing information about the cars owned by that owner.

23. Given a text file, create a doubly linked list, each element of which contains the number of characters in the corresponding line of text.

Exercise III

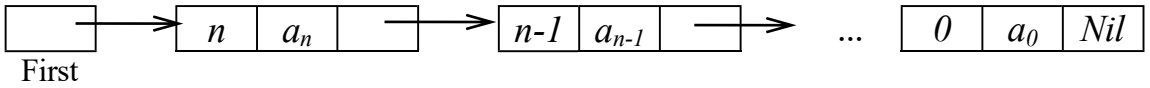
1. Polynomial $p(x) = a_n x^n + a_{n-1} x^{n-1} + ... + a_1 x + a_0$ s whole coefficients Can present it as a list (see fig.), and if $a_i = 0$, then the corresponding link is not included in the list.



Describe type data corresponding such presentation And program For working with lists of such a structure, which includes:

- procedure **(P),** which reads from the keyboard kit numbers and forms a polynomial list P.
- procedure deletions specified element from list.
- procedure inserts element V arbitrary position list
- logical function **equals(P,Q),** checking on equality two polynomial.
- procedure **output(P,x)** , which prints the polynomial p as a polynomial in the variable x.

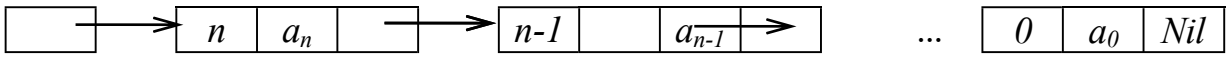
2. Polynomial $p(x) = a_n x^n + a_{n-1} x^{n-1} + ... + a_1 x + a_0$ s whole coefficients Can represent in the form list (see fig.), and if $a_i = 0$, then the corresponding link is not included in the list.



Describe type data corresponding such presentation And program For working with lists of such a structure, which includes:

- procedure **input(P),** which reads from the keyboard a set of numbers and forms a polynomial list P.
- procedure deletions specified element from list.
- procedure inserts element V arbitrary position list
- the procedure – **diff(P,Q),** which constructs the polynomial Q the derivative of the polynomial P.
- procedure **output(P,x)** , which prints polynomial p How polynomial from the variable x.

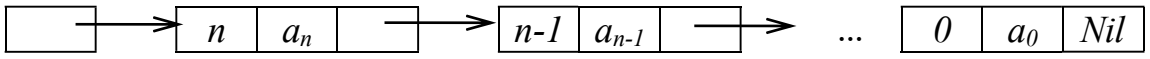
3. Polynomial $p(x) = a_n x^n + a_{n-1} x^{n-1} + ... + a_1 x + a_0$ s whole coefficients Can present it as a list (see fig.), and if $a_i = 0$, then the corresponding link is not included in the list.



Describe type data corresponding such presentation And program For working with lists of such a structure, which includes:

- procedure **input(P),** which reads from the keyboard a set of numbers and forms a polynomial list P.
- procedure deletions specified element from list.
- procedure inserts element V arbitrary position list
- procedure **addition(P,Q,R),** which builds polynomial R, How amount polynomials P and Q.
- procedure **output(P,x)** , which prints polynomial p How polynomial from variable x.

4. Polynomial $p(x) = a_n x^n + a_{n-1} x^{n-1} + ... + a_1 x + a_0$ s whole coefficients Can present it as a list (see fig.), and if $a_i = 0$, then the corresponding link is not included in the list.



Describe type data corresponding such presentation And program For working with lists of such a structure, which includes:

- procedure **input(P),** which reads from the keyboard a set of numbers and forms a polynomial list P.
- procedure deletions specified element from list.
- procedure inserts element V arbitrary position list
- function **value(P,x),** computing meaning polynomial P, V some integer point x.
- procedure **output(P,x)** , which prints polynomial p How polynomial from variable x.

5. N children are arranged in a circle. Starting from the first, they remove each *kth* child, closing the circle after each removal. Determine the order in which children are removed from the circle.

Solution this tasks describe V view programs, which

- forms annular list;
- prints names Guys V volume in order, V which They are deleted from circle.

6. IN file system catalog files organized How linear list. For Each file in the directory contains the following information:

- Name file;
- Date of creation;
- Number of file accesses; Compose

program, which provides:

- Initial formation catalog files;
- Conclusion catalog file;
- Removal files, date creation which less required;
- Conclusion file With the greatest quantity appeals.

7. Questionnaire For survey population contains two groups questions. The first group contains information about the respondent:

Age;

Floor

Education (initial, average, higher)

Second group contains actually question questionnaires, answer on which can be either "YES" or "NO"

Compose program, which:

- Provides elementary input questionnaires And forms from them linear list;
- On basis analysis questionnaires gives out answers on next questions:
  1) How many men older 40 years, having higher education, answered “YES” to the survey question;
  2) How many women younger 30 years, having average education, answered “NO” to the questionnaire question;
  3) How many men under 25 years of age with primary education answered “YES” to the survey question;
- Produces conclusion everyone questionnaires And answers on questions.

8. Create a program that contains dynamic information about the availability of buses in the bus depot.

Intelligence O each The bus contains:

- Number bus;
- Full name driver;
- Number route.

Program should provide next possibilities:

- Initial formation data about everyone buses V in the park V view list;
- When each bus leaves the depot , the bus number is entered and the program deletes the data about this bus from the list of buses,

located in the park and records this data in the list of buses on the route;
- When a bus enters the depot, the bus number is entered, and the program removes the data about this bus from the list of buses on the route, and writes the data about this bus to the list of buses in the depot;
- Upon request, it provides a list of all buses in the depot or on the route.

9. Given a polynomial (in variable x) of arbitrary degree with integer coefficients, these coefficients may be unordered by powers of x, and monomials of the same degree may be repeated (for example, $-8x^4 + 3x^5 + 2x^4 - 1$). You are required to write a program that performs the following functions:
- procedure **input(P),** which reads a set of numbers from the keyboard and forms a polynomial list P.
- deletes specified element from list.
- inserts element V arbitrary position list
- procedure, which will lead similar And will form new polynomial.
- procedure **output(P,x)** , which prints polynomial p How polynomial from variable x.

10. Create a program that contains dynamic information about books in a library.
Intelligence O each book contain:
- Number UDC;
- Full name author;
- Name;
- Year publications;
- Quantity copies V library

Program should provide next possibilities:
- Initial formation data about everyone books V library V view list;
- Addition data O books, again received V library;
- Removal data O written off books;
- Upon request, information about the availability of books in the library is provided , sorted by year of publication.

11. Compose program, which contains dynamic information O requests for air tickets.
Each one The application contains:
- Paragraph appointments;
- Number flight;
- Full name passenger;
- Date departure;

Program should provide next possibilities:

- Initial formation data about everyone applications V view list;
- Addition data O again received applications;
- Removal data O completed applications;
- By request are issued intelligence By given number flight And date departure.

12. Create a program that contains dynamic information about books in a library.
Intelligence O each book contain:
    - Number UDC;
    - Full name author;
    - Name;
    - Year publications;
    - Quantity copies V library.

Program should provide next possibilities:
    - Initial formation of data on all books in the library in the form of a list;
    - Changing the number of copies when issuing a book and inserting information about the issued book into the list of issued books or displaying a message that the book is currently issued (if the number of copies is 0);
    - When a book is returned, the program should increase the number of copies by one;
    - Upon request, information about the availability of books in the library or books on hand is provided.

13. Intelligence about student contain
    - Full name
    - names class (years training And letters), V which He studies.

Create a program that allows:
    - create a list containing information about the school's students in alphabetical order;
    - add an entry to a list in such a way that the order of the list is not changed;
    - view content list;
    - delete intelligence about student;
    - find out there are whether V school namesakes With identical names. Access to all the above procedures should be provided as a menu.

14. Intelligence O luggage passenger include intelligence
    - Full name passenger;
    - quantity things;
    - common weight things.

Create a program that provides the following functionality:
    - Initial formation of a list containing information about the baggage of all passengers;

- At admission new luggage - addition information O him V list;
- At issuance luggage - removal information O him;
- By request are issued surnames passengers, weight luggage which exceeds the norm.

Access ko to everyone the above procedures design V view menu.

15. Intelligence about student include
    - Name And last name
    - names class (years training And letters), V which He studies.

Compose program allowing:
- create a list containing information about the school's students , in alphabetical order of last names;
- add recording V list, like this in this way, to order increase alphabetically unchanged;
- view content list;
- delete indicated recording from list;
- find out there are whether V school namesakes.

Access ko to everyone the above procedures design V view menu.

16. Intelligence about student include
    - Name And last name
    - names class (years training And letters), V which He studies.

Compose program allowing:
- form queue, containing intelligence about students schools;
- add recording V queue;
- view content queues;
- delete n records from queues;
- form a queue g containing information about students in grades 9 and 10, placing information about students in grade 9A at the beginning, then 9B, etc., then 10A, 10B, etc.

Access ko to everyone the above procedures design V view menu.

17. Intelligence about exported product consist from following entries:
    - Name goods,
    - country importing, product
    - volume of the supplied batch.

Compile program allowing:
- form queue, containing intelligence O supplied goods;
- add recording V queue;
- view content queues;
- If goods delivered V country appointments - delete intelligence O them from queues;

- on request, display the names of goods exported to a given country and find the total volume of goods imported by a given country.

Access ko to everyone the above procedures design V view menu.

18. Intelligence about those on the waiting list on improvement housing conditions contain the following data:
- Full name,
- Address,
- Number queues.

Compose program allowing
- form queue, containing necessary intelligence;
- add recording V queue;
- view content queues;
- at input V exploitation N apartments print queue lucky ones to receive orders and remove them from the original queue;

Access ko to everyone the above procedures design V view menu.

19. Intelligence O goods on warehouse consist from the following entries:
- Name goods,
- price,
- quantity this species goods. Create

a program that allows:
- form list, containing intelligence O goods;
- add an entry to the list as follows: if information about a certain product is already present in the list, then increase its number, if not, add it to the end;
- view intelligence O goods, existing on warehouse;
- when selling a product, reduce its quantity; if the quantity of the product is 0, remove information about the product from the list;

Access ko to everyone the above procedures design V view menu.

20. Card index V bureau exchange apartments organized How linear list.
Information about each apartment contains:
- Quantity rooms;
- Floor;
- Square;
- Address;

Compose program, which provides:
- Initial formation card indexes;
- Input applications on exchange;
- Search the card index for a suitable option: if the number of rooms and floors are equal and the difference in area is within 10% the corresponding card is displayed, which then is being removed from list; V otherwise case the received application is included in the list;
- Conclusion the entire list.

21. Given expression V postfix form records (This recording, V which the operation sign is placed after the operands, for example,
ab → a b –

a*bc → ab*c + a*(b+c)
→ a b c + *)

1) describe procedure creation stack;
2) describe procedure inserts element V stack;
3) describe procedure deletions element from stack;
4) using described procedures, describe function, which calculates How An integer value of an expression written in postfix notation in a text file or string (use the following algorithm: the expression is scanned from left to right. If an operand (number) is encountered, its value (as an integer) is pushed onto the stack. If an operator sign is encountered, the last two elements (the operands of the given operation) are popped from the stack, the operation is performed on them, and its result is pushed onto the stack. At the end, only one number remains on the stack—the value of the entire expression).

Option 22

1) describe procedure creation stack;
2) describe procedure inserts element V stack;
3) describe procedure deletions element from stack;
4) describe a procedure that prints the contents of a text file, writing out the characters of each line in reverse order (you can use the following algorithm: read the characters of the text file and push them onto the stack until the end of the line is reached, then remove elements from the stack and write them to the screen. Do this for each line of the file).

23. IN text file (or line) recorded without mistakes formula of the form:
<formula>::=<digit>| M(<formula>,<<formula>)| m(<formula>,<formula>) where
M denotes the maximum function, and m - minimum.

1. describe procedure creation stack;
2. describe procedure inserts element V stack;
3. describe procedure deletions element from stack;
4. Using the procedures described, describe a procedure or function that calculates (as an integer) the value of a given formula (for example, M(5, m(6,8)) = 6).

Control questions

1. How differ static And dynamic quantities?
2. What such pointer? What such link?
3. Which types pointers to you are known?

4. How is determined address variable?
5. How to allocate memory for a dynamic variable? How to free memory from a dynamic variable variable?
6. What such "dereference"?
7. Which situations lead To emergence V dynamically distributed memory of "garbage"?
8. What is meant by a "linked list"? Definition. Graphical representation.
9. How classify related lists?
10. What are the basic operations on lists and list components that are typically implemented?
11. How is described list?
12. Concept stack, queues.
13. Typical operations, performed over stack.
14. Typical operations, performed over queue.

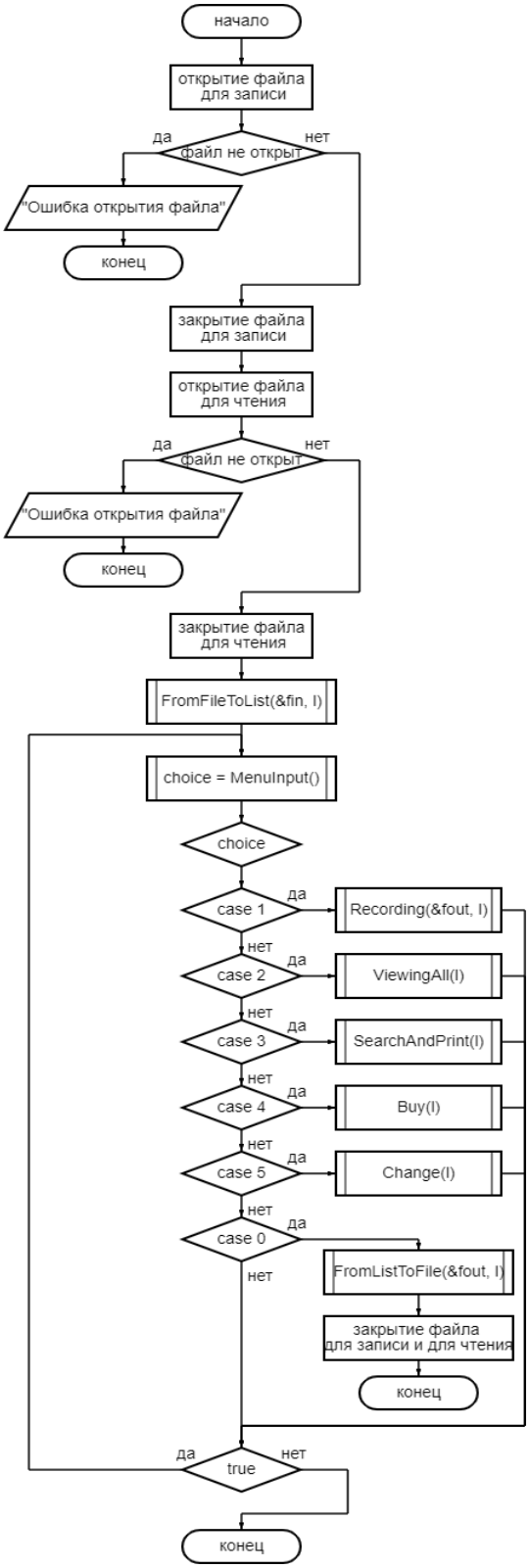
Example execution tasks II

Exercise II :

Information about office premises is provided. The structure is as follows: owner's last name, area, floor, address (district + actual address). Display data on offices located in the desired LOC district and whose area is greater than the min but less than the max. The LOC, min, and max are entered by the user using the keyboard. If one of the office premises is purchased, it will need to be deleted from the database.

Solution

1. Algorithm



2. Description used functions

`void set_color( int text , int background = 0)` – setting the desired color for the text
`void write_colored_text( string text , int color , int background = 0)` - displaying colored text on the screen

`void push_back( DynamicList & List , Node * note )` – adding a note element to the end of a List

`void ViewingAll( DynamicList list )` – displaying a dynamic list list in the form of a table

`void ViewingSingle( DynamicList list , Node * to_delete )` – output information about the requested element to_delete from a dynamic list list

`void Recording( ofstream * file , DynamicList & list )` – writing the fields of the list structure to the file

`void FromFileToList( ifstream * file , DynamicList & list )` – adding all entries from the file included. list list

`void FromListToFile( ofstream * file , DynamicList list )` – write data from the dynamic list list to the file file

`void DeleteNode( DynamicList & list , Node * to_delete )` – deleting the to_delete entry from the dynamic list

`void SearchAndPrint( DynamicList list )` – search for records in a dynamic list

`Node * Search( DynamicList list , int number )` - search for a record in a dynamic list list by the specified number number

`void Buy( DynamicList & list )` - "purchase" premises - Find and delete the specified entry from the dynamic list

`void PrintMenu()` - conclusion menu on the screen

`bool Check_correctly( int x )` - examination on the correctness of the entered command x (must be in the range 0-5)

`int MenuInput()` - organization input teams 0- 5

`void Change( DynamicList & list )` - change data in a dynamic list

3. Program

```
#include <iostream>
#include <iomanip>
#include <string>
```

```
#include <fstream>
#include <conio.h>
#include <windows.h>

using namespace std;

struct Location {
	char district[20] = ""; // district
	char address[20] = ""; // actually, address
};

structNode {
	// fields data
	charsurname[20] = ""; // surname owner
	double square = 0; // area
	int price = 0; // price
	Location loc; // location
	// link on next node Node*
	next = nullptr;
};

structDynamicList {
	// at creation dynamic sheet initially empty, those. no on What Not specifies
	Node* Head = nullptr;
};

// sets necessary color For text (very cool)
voidset_color(inttext, intbackground = 0) { // Installation flowers For text And his
	background HANDLE hStdOut = GetStdHandle(STD_OUTPUT_HANDLE);
	SetConsoleTextAttribute(hStdOut, (WORD)((background << 4) | text));
}

// outputs on screen colored text
void write_colored_text(string text, int color, int background = 0) {
	// change the text color
	set_color(color, background);
	// we are deducing this text
	cout<<text;
	// we are installing back color text By default (white)
	set_color(15, background);
}

// adds V element V end list
void push_back(DynamicList& Leaf, Node* notes) {
	Node* NewNode = new Node{};
	// we copy structure
	strcpy_s(NewNode->surname, 20, notes-
	>surname); NewNode->square = note->square;
	NewNode->price = note- >price;
	strcpy_s(NewNode->loc.district, 20, note- >loc.district);
	strcpy_s(NewNode->loc.address, 20, note- >loc.address);

	// If our din. list empty, That new element will first if
	(List.Head == nullptr)
		List.Head = NewNode;
	else {
		Node* current = List.Head;
		// we are looking for last element, those. That, which indicates
		to NULL while (current->next != nullptr)
			current = current->next;
		current->next = NewNode;
	}
```

```
}

// outputs our All dynamic list void
ViewingAll(DynamicList list) {
	system("cls");

	if (list.Head == nullptr) {
		write_colored_text("\nList empty!\n\n", 4);
	}
	else {
		cout<< "                  Table \"INFORMATION ABOUT OFFICE PREMISES " <<endl;
		cout<<                                                                                    "
_____________________________________________________________________________________________
_____________" <<endl;
		cout<< "           |      |                     |              |            |
Location                  |" <<endl;
		cout<< "          |  No.      Surname     |  Square    |   Price  |     District
                          |
|      Address     |" <<endl;
		cout<<                                                                                    "
| ____| _______________| ___________| _________| ___________________| ___________________|"
<<endl;

		// counter For numbering elements
		int counter = 1;
		while (list.Head != nullptr) {
			cout << "          | "
				<< left << setw(3) << counter << "|  "
				<< left << setw(15) << list.Head->surname << "|  "
				<< left << setw(10) << list.Head->square << "|  "
				<< left << setw(8) << list.Head->price << "|  "
				<< left << setw(16) << list.Head->loc.district << "|  "
				<< left << setw(16) << list.Head->loc.address << "|\n";

			counter++;
			list.Head = list.Head->next;
		}
		cout                                          <<                                          "
|____|_______________|___________|_________|___________________|___________________|"   <<
endl;
	}
}

void ViewingSingle(DynamicList list, Node* to_delete) {
	cout << "          | "
		<< left << setw(15) << to_delete->surname << "|  "
		<< left << setw(10) << to_delete->square << "|  "
		<< left << setw(8) << to_delete->price << "|  "
		<< left << setw(16) << to_delete->loc.district << "|  "
		<< left << setw(16) << to_delete->loc.address << "|\n";
}

// writes down fields structures V file
void Recording(ofstream* file, DynamicList& list) {
	system("cls");
	// we're writing it down list V file, What would them Can was to use
	later file->open("primer.txt", ios::out | ios::app);

	cout<< "Start entering data\n[Esc] - End entering data\nUse this after you have finished
entering data into the structure.\n\n";
	Nodehome{};
```

```
	// we delete symbol new lines from thread
	cin.ignore();
	do {
		cout<< "Surname owner premises: ";
		gets_s(home.surname);

		cout << "Площадь помещения: ";
		cin >> home.square;

		cout << "Цена: ";
		cin >> home.price;

		cin.ignore(1024, '\n');
		cout << "Район: ";
		gets_s(home.loc.district);
		cout << "Адрес: ";
		gets_s(home.loc.address);

		file->write((char*)&home, sizeof(Node));

		write_colored_text("Data successfully recorded! [Esc] - exit.\n", 2);
		// we add structure V dynamic list push_back(list,
		&home);

	} while (_getch() != 27); // 27 - code buttons Esc

	file->close();
	system("cls");
}

// adds all records from the file to a dynamic void list
FromFileToList(ifstream* file, DynamicList& list) {
	file->open("example.txt", ios::in | ios::app);

	// If file Not empty, we take from there
	data if (file->peek() != EOF) {
		Node home{};
		while (file->read((char*)&home, sizeof(Node))) {
			push_back(list, &home);
		}
	}

	file->close();
}

// writes down from dynamic list All records V void file
FromListToFile(ofstream* file, DynamicList list) {
	// For beginning we clean file
	file->open("primer.txt", ios::trunc);
	file->close();

	file->open("primer.txt", ios::out | ofstream::app);
	Node* current = list.Head;
	while (current != nullptr) {
		file->write((char*)current, sizeof(Node));
		current = current->next;
	}

	file->close();
}

// deletes recording from dynamic list
```

```
void DeleteNode(DynamicList& list, Node* to_delete) {
	// if the component is the first (that is, the head points to it), then it is enough to
just move the reference to the first element forward
	if (to_delete == list.Head)
		list.Head = to_delete->next;
	else {
		Node* current = list.Head;
		while (current->next != to_delete)
			current = current->next;
		current->next = to_delete->next;
	}
	delete to_delete;
}

void SearchAndPrint(DynamicList list) {
	system("CLS");
	cout<< "\nIt begins search data..\n\n";

	cout<< "Enter minimum square: ";
	doublesqr_min;
	cin>>sqr_min;

	cout<< "Enter maximum square: "; double
	sqr_max;
	cin >> sqr_max;

	cout << "Enter district: ";
	char distr[20] = "";
	cin.ignore(1024, '\n');
	gets_s(distr);


	int counter = 0;
	boolflag = false; // If will Although would one suitable recording, Then let's
change the flag's state
	while (list.Head != nullptr) {
		if (!strcmp(list.Head->loc.district, distr) && (sqr_min < list.Head->square) &&
(sqr_max > list.Head->square)) {
			if (!flag) {
				cout                                    <<                                                "
_____" << endl;
				cout << "          |                    |             |           |
Местоположение            |" << endl;
				cout << "           |      Last name    |  Area       |    Price   |
District   |      Address     |" << endl;
				cout<<                                                                                "
| _________________| ___________| _________| ___________________| ___________________|" <<endl;
				flag = true; // we change state flag, because found Although
				would one
record
ing
			}
			ViewingSingle(list, list.Head);
		}
		list.Head = list.Head->next;
	}
	if (flag) {
		cout                                          <<                                              "
|_________________|____________|_________|___________________|___________________|" << endl;
	}
	else
		write_colored_text("\nТаких записей нет!\n\n", 4);
	system("pause");
	system("CLS");
```

```
}

Node* Search(DynamicList list, int number) {
	int counter = 1;
	while (list.Head != nullptr) {
		if (number == counter)
			return list.Head;
		counter++;
		list.Head = list.Head->next;
	}
	return list.Head;
}

// покупка
void Buy(DynamicList& list) {
	if (list.Head == nullptr) {
		system("CLS");
		write_colored_text("\nСписок пуст!\n\n", 4);
		system("pause");
		system("CLS");
		return;
	}

	ViewingAll(list);

	write_colored_text("Room under what number You do you want purchase?\n", 14);

	int number = 1;
	cin >> number;

	Node* to_buy = Search(list, number);

	if (to_buy == nullptr) {
		system("CLS");
		write_colored_text("\nRecords With like this number No! \n", 4);
		system("pause");
		system("CLS");
		return;
	}

	cout                                        <<                                                                  "
_____" << endl;
	cout << "         |                 |            |          |           Location
|" << endl;
	cout << "           |      Last name    |  Area       |   Price   |        District    |
Address     |" << endl;
	cout                                        <<                                                                  "
| _______________| ___________| _________| _________________| _________________|" << endl;
	ViewingSingle(list, to_buy);
	cout                                        <<                                                                  "
| _______________| ___________| _________| _________________| _________________|" << endl;

	write_colored_text("\nYou we are sure What do you want buy This premises? (+ Yes; - No)
	\n",
14);

	char choose;
	do {
		choose = _getch();
	} while ((choose != '+') && (choose != '-'));

	if (choose == '+') {
		system("CLS");
```

```
			write_colored_text("\nWrite successfully removed!\n\n", 2);
			DeleteNode(list, to_buy);
			system("pause");
			system("CLS");
		}

}

// prints a color menu
void PrintMenu() {
		write_colored_text("\n                    MENU\n", 2);
		write_colored_text(" • [1] Add a record to a file \n", 14);
		write_colored_text(" • [2] View data \n", 14);
		write_colored_text(" • [3] Search By area And district \n",
		14); write_colored_text(" • [4] Purchase of premises \n", 14);
		write_colored_text(" • [5] Change of information \n", 14);
		write_colored_text(" • [0] Exit \n", 14);
		write_colored_text("Select an item: ", 15);
}

// checks on right introduced team (What would was included V interval 0-5)
bool Check_correctly(int x) {
		switch (x) {
		case 1:
		case 2:
		case 3:
		case 4:
		case 5:
		case 0:
			return true;
		default:
			return false;
		}
}

// organizes the introduction of a
command (numbers 0-5) int MenuInput()
{
		PrintMenu();

		int m;
		do {
			cin>>m;
			// examination on happening, If introduced
			NOT number if (cin.fail()) {
				cin.clear();
				cin.ignore(32767, '\n');

				system("cls");
				PrintMenu();
				write_colored_text("\nYou introduced Not number! Repeat attempt: \n",
				4); continue;
			}
			// If introduced number, But Not correct, we write
			about in this if (!Check_correctly(m)) {
				system("cls");
				PrintMenu();
				write_colored_text("\nEntered unacceptable number. Repeat attempt: \n",
4);
				continue;
			}
			returnm;
```

```
	} while (true);
	return m;
}

void Change(DynamicList& list) {
	if (list.Head == nullptr) {
		system("CLS");
		write_colored_text("\nList empty!\n\n", 4);
		system("pause");
		system("CLS");
		return;
	}

	ViewingAll(list);

	write_colored_text("Enter number premises For changes data O him:  \n", 14); int

	number = 1;
	cin >> number;

	Node* to_change = Search(list, number);

	if (to_change == nullptr) {
		system("CLS");
		write_colored_text("\nЗаписи с таким номером нет! \n", 4);
		system("pause");
		system("CLS");
		return;
	}
	cout<< "Surname owner premises: ";
	gets_s(to_change->surname);

	cout << "Square premises: ";
	cin >> to_change->square;

	cout << "Price: ";
	cin >> to_change- >price;

	cin.ignore(1024, '\n');
	cout << "District: ";
	gets_s(to_change->loc.district);
	cout << "Address: ";
	gets_s(to_change->loc.address);
	system("CLS");
}

// home function
int main() {
	// we do setting For work With Russian language
	setlocale(LC_ALL, "Rus");
	SetConsoleCP(1251);
	SetConsoleOutputCP(1251);

	ofstream fout("primer.txt", ios::out | ios::app);
	if (!fout.is_open()) {
		write_colored_text("Ошибка открытия файла!\n", 4);
		exit(0);
	}
	fout.close();

	ifstream fin("primer.txt", ios::in | ios::app);
	if (!fin.is_open()) {
```

```
		write_colored_text("Ошибка открытия файла!\n", 4);
		exit(0);
	}
	fin.close();


	DynamicList l;
	FromFileToList(&fin, l);

	short choice;
	do {
		choice = MenuInput();
		switch (choice) {
		case 1:
			Recording(&fout, l);
			break;
		case 2:
			ViewingAll(l);
			system("pause");
			system("CLS");
			break;
		case 3:
			SearchAndPrint(l);
			break;
		case 4:
			Buy(l);
			break;
		case 5:
			Change(l);
			break;
		case 0:
			FromListToFile(&fout, l);
			fout.close();
			fin.close();
			return EXIT_SUCCESS;
		}
	} while (true);

	system("pause");
	return 0;
}
```

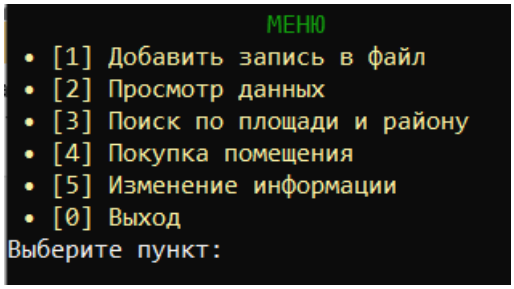
4. Testing programs

Task: Information about office premises is given. The structure is as follows: Owner's last name, area, floor, address (district + address). Display data on offices located in the desired LOC district and whose area is greater than the min but less than the max. The LOC, min, and max are entered by the user using the keyboard. When one of the office spaces is purchased, its entry is removed from the list.

Entrance data:

Surname	Square	Price	Location	
			District	Address
Ivanov	35	18500	Leninist	Vavilov 23
Petrova	20	11000	Shpakovsky	Pirogova 45
Kurochkin	30	17,000	Seaside	Lenin 678
Volkova	20	9300	Leninist	Chekhov 1
Lysenko	40	20,000	October	Mira 9B

External menu view :



1. Input data V structure. (Pressed team 1)

```
Начинайте ввод данных
[Esc] - Завершение ввода данных
Используйте это после завершения внесения данных в структуру.

Фамилия владельца помещения: Иванов
Площадь помещения: 35
Цена: 18500
Район: Ленинский
Адрес: Вавилова 23
Данные успешно записаны! [Esc] - выход.
Фамилия владельца помещения: Петрова
Площадь помещения: 20
Цена: 11000
Район: Шпаковский
Адрес: Пирогова 45
Данные успешно записаны! [Esc] - выход.
Фамилия владельца помещения: Курочкин
Площадь помещения: 30
Цена: 17000
Район: Приморский
Адрес: Ленина 678
Данные успешно записаны! [Esc] - выход.
Фамилия владельца помещения: Волкова
Площадь помещения: 20
Цена: 9300
Район: Ленинский
Адрес: Чехова 1
Данные успешно записаны! [Esc] - выход.
Фамилия владельца помещения: Лысенко
Площадь помещения: 40
Цена: 20000
Район: Октябрьский
Адрес: Мира 9Б
Данные успешно записаны! [Esc] - выход.
```

2. View data (pressed team 2).

Таблица "ИНФОРМАЦИЯ О ОФИСНЫХ ПОМЕЩЕНИЯХ"

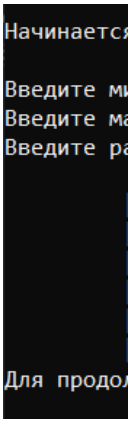
№	Фамилия	Площадь	Цена	Местоположение	
				Район	Адрес
1	Иванов	35	18500	Ленинский	Вавилова 23
2	Петрова	20	11000	Шпаковский	Пирогова 45
3	Курочкин	30	17000	Приморский	Ленина 678
4	Волкова	20	9300	Ленинский	Чехова 1
5	Лысенко	40	20000	Октябрьский	Мира 9Б

3. Search By area And district (pressed team 3).

Table tests:

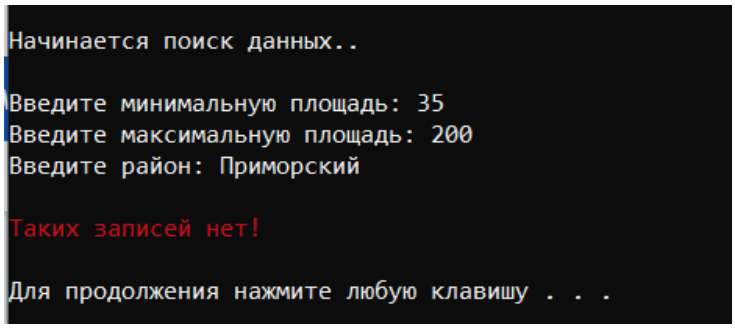
№ теста	min площадь	max площадь	Район	Ожидаемый результат
1	15	49	Ленинский	Вывод информации о помещениях владельцев: Иванов и Волкова
2	35	200	Приморский	Сообщение об отсутствии искомых записей
3	19.9	20.1	Шпаковский	Вывод информации о помещении владельца: Петрова
4	1	300000	Октябрьский	Вывод информации о помещении владельца: Лысенко
5	34	56	Промышленный	Сообщение об отсутствии искомых записей

1 th os

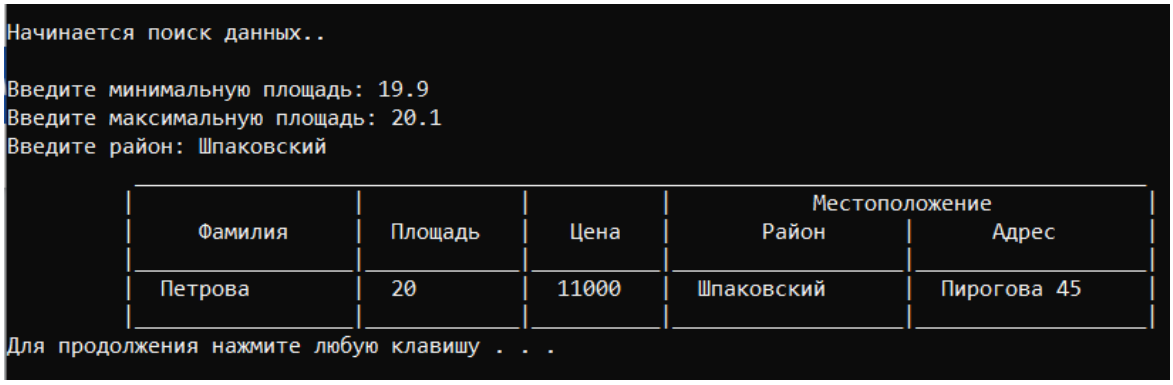


2 those

Art.



3 test.



4 test.

```
Начинается поиск данных..

Введите минимальную площадь: 1
Введите максимальную площадь: 300000
Введите район: Октябрьский
```

Фамилия	Площадь	Цена	Местоположение	
			Район	Адрес
Лысенко	40	20000	Октябрьский	Мира 9Б

```
Для продолжения нажмите любую клавишу . . .
```

5 test.

```
Начинается поиск данных..

Введите минимальную площадь: 34
Введите максимальную площадь:
56
Введите район: Промышленный

Таких записей нет!

Для продолжения нажмите любую клавишу . . .
```

4. Purchase premises (pressed team 4).

```
                    Таблица "ИНФОРМАЦИЯ О ОФИСНЫХ ПОМЕЩЕНИЯХ"
 ____ ____________ ____________ ____________ ____________ ____________
|    |            |            |            |  Местоположение         |
| №  |  Фамилия   |  Площадь   |   Цена     |   Район    |   Адрес    |
|____|____________|____________|____________|____________|____________|
| 1  | Иванов     | 35         | 18500      | Ленинский  | Вавилова 23|
| 2  | Петрова    | 20         | 11000      | Шпаковский | Пирогова 45|
| 3  | Курочкин   | 30         | 17000      | Приморский | Ленина 678 |
| 4  | Волкова    | 20         | 9300       | Ленинский  | Чехова 1   |
| 5  | Лысенко    | 40         | 20000      | Октябрьский| Мира 9Б    |
|____|____________|____________|____________|____________|____________|
Помещение под каким номером вы хотите приобрести?
2
 ____________ ____________ ____________ ____________ ____________
|            |            |            |  Местоположение         |
|  Фамилия   |  Площадь   |   Цена     |   Район    |   Адрес    |
|____________|____________|____________|____________|____________|
| Петрова    | 20         | 11000      | Шпаковский | Пирогова 45|
|____________|____________|____________|____________|____________|

Вы уверены, что хотите купить это помещение? (+ да; - нет)
```

At pressing "+" appears message:

```
Запись успешно удалена!

Для продолжения нажмите любую клавишу . . .
```

Now the deleted entry will not be displayed. at viewing tables (team 2). Alternative option:

```
                    Таблица "ИНФОРМАЦИЯ О ОФИСНЫХ ПОМЕЩЕНИЯХ"
 ____ ____________ ____________ ____________ ____________ ____________
|    |            |            |            |  Местоположение         |
| №  |  Фамилия   |  Площадь   |   Цена     |   Район    |   Адрес    |
|____|____________|____________|____________|____________|____________|
| 1  | Иванов     | 35         | 18500      | Ленинский  | Вавилова 23|
| 2  | Курочкин   | 30         | 17000      | Приморский | Ленина 678 |
| 3  | Волкова    | 20         | 9300       | Ленинский  | Чехова 1   |
| 4  | Лысенко    | 40         | 20000      | Октябрьский| Мира 9Б    |
|____|____________|____________|____________|____________|____________|
Помещение под каким номером вы хотите приобрести?
1
 ____________ ____________ ____________ ____________ ____________
|            |            |            |  Местоположение         |
|  Фамилия   |  Площадь   |   Цена     |   Район    |   Адрес    |
|____________|____________|____________|____________|____________|
| Иванов     | 35         | 18500      | Ленинский  | Вавилова 23|
|____________|____________|____________|____________|____________|

Вы уверены, что хотите купить это помещение? (+ да; - нет)
```

At pressing "–" will return menu, removal Not will happen And recording will remain V table.

5. Change information (pressed team 5).

```
                    Таблица "ИНФОРМАЦИЯ О ОФИСНЫХ ПОМЕЩЕНИЯХ"
┌──────┬──────────────┬──────────┬──────┬──────────────┬──────────────┐
│      │              │          │      │ Местоположение              │
│  №   │   Фамилия    │ Площадь  │ Цена │    Район     │    Адрес     │
├──────┼──────────────┼──────────┼──────┼──────────────┼──────────────┤
│  1   │ Иванов       │ 35       │ 18500│ Ленинский    │ Вавилова 23  │
│  2   │ Петрова      │ 20       │ 11000│ Шпаковский   │ Пирогова 45  │
│  3   │ Курочкин     │ 30       │ 17000│ Приморский   │ Ленина 678   │
│  4   │ Волкова      │ 20       │ 9300 │ Ленинский    │ Чехова 1     │
│  5   │ Лысенко      │ 40       │ 20000│ Октябрьский  │ Мира 9Б      │
└──────┴──────────────┴──────────┴──────┴──────────────┴──────────────┘
Введите номер помещения для изменения данных о нём:
4
Фамилия владельца помещения: Волков
Площадь помещения: 45
Цена: 29000
Район: Промышленный
Адрес: Ермолова 150
```

After changing the information in the table , the updated information will be displayed when viewing (command 2) :

6. Exit from programs (pressed command 0).

```
                МЕНЮ
 • [1] Добавить запись в файл
 • [2] Просмотр данных
 • [3] Поиск по площади и району
 • [4] Покупка помещения
 • [5] Изменение информации
 • [0] Выход
Выберите пункт: 0

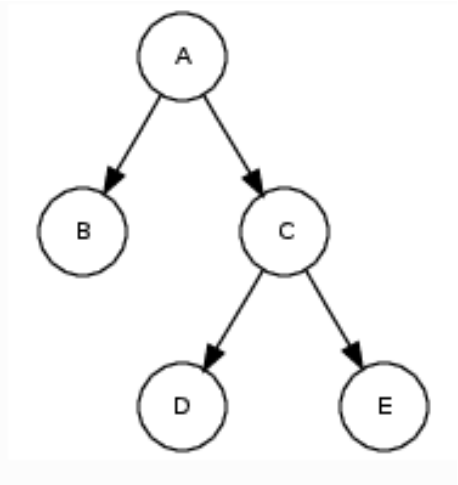
C:\Users\ANNA\source\repos\lab16example\Debug\lab16example.ex
Чтобы автоматически закрывать консоль при остановке отладки,
томатически закрыть консоль при остановке отладки".
Нажмите любую клавишу, чтобы закрыть это окно…
```

17. Trees

17.1 Concept tree

A tree is one of the most fundamental data structures, widely used in programming. It stores data hierarchically rather than linearly .

A tree is a hierarchical data structure consisting of nodes arranged at levels and connected by branches.

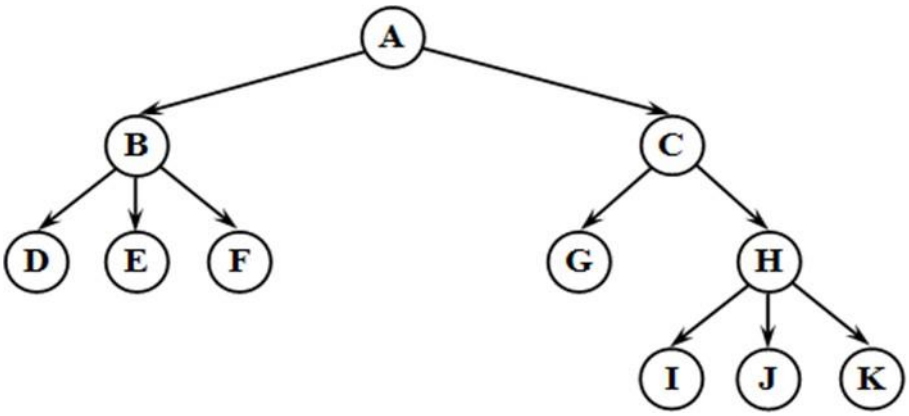


Trees combine the advantages of the other two structures: ordered array And linked list.

If in a list access is carried out on average in O(N), then in a binary tree it is O(log2(N))

- A - корневой узел (нулевой уровень).
- B,C - потомки первого уровня.
- D,E - потомки второго уровня.
- B,D,E - листы.
- **Глубина дерева** - 2.

17.2 Classification trees



Main classifications trees.

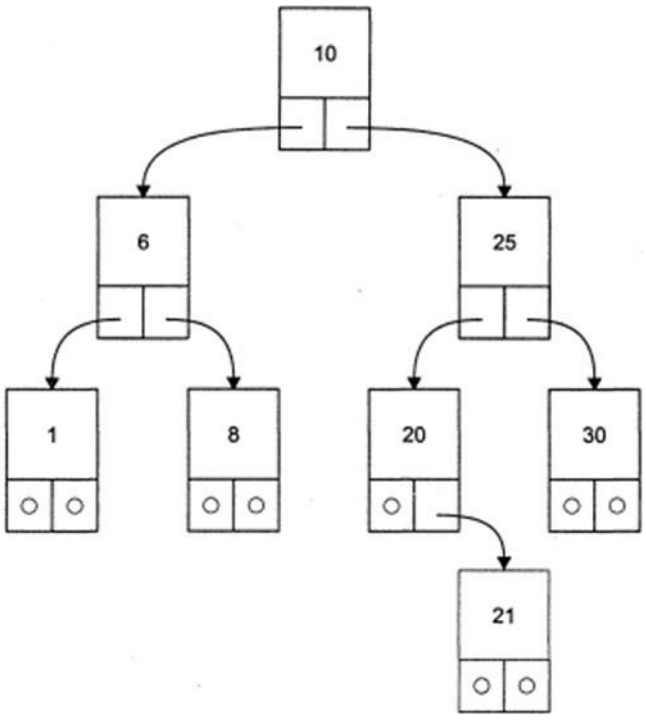
- By maximum quantity descendants at one node:
  - binary (2 descendant);
  - ternary (3 descendant);
  - M-ary (M- descendants);
- By character data:
  - With ordered data;

– With disordered data;

- по структуре:
  - симметричные;
  - сбалансированные;
  - несбалланси́рованные;
  - полные;
  - вырожденные;

- по виду:
  - бинарные с упорядоченными данными (поиска);
  - B-деревья;
  - Красно-чёрные (с балансировкой);

17.3 Binary tree



A binary tree is an unbalanced tree in which each node has at most two **children** . Typically, the first is called the parent node, and the children are called the left and right **descendants** .

A binary tree is not an ordered oriented tree.

17.4 Job With binary tree

To work with a binary tree, three variables are used: a pointer to the left child, a pointer to the right child, and the node value.

Structure node (example):

```
struct BinaryTree {
int value;
BinaryTree* left;
BinaryTree* right;
};
```

Filling tree

```
if (key < leaf->value) { //проверка значения
    if (leaf->left != NULL) //если узел не пустой
    insertFromChild(leaf->left, key); //отправляем в рекурсию
else { //если пустой
    leaf->left = new BinaryTree; //заполняем его поданным значением
    leaf->left->value = key;
    leaf->left->left = NULL;
    leaf->left->right = NULL;
    }
}
```

Bypasses

Bypass - This a procedure in which each node is processed exactly once in a uniform manner. Tree traversal consists of partitioning the tree into a root, left, and right subtrees, and applying the appropriate processing procedure to each subtree until the partitioning process yields an empty tree.

Bypass – is a procedure in which each node is processed exactly once in a uniform manner. Tree traversal consists of splitting the tree into a root, left, and right subtrees, and applying the appropriate processing procedure to each subtree until the splitting process yields an empty tree.

Descending (direct) bypass is performed according to the formula ***"root-left-right" (RLR)*** .

- process root knot;
- go around left subtree V descending in order;
- go around right subtree V descending in order.

Laboratory Job No. 17
Study features processing data at using a binary tree

TARGET WORKS:

– Improvement skills developments programs V environment programming
– Improvement skills V programming With using pointers
– Study process formations elements the simplest dynamic structure data - binary tree
– Study operations at work With binary trees

EXECUTION WORKS: V in accordance With option make up And implement programs.
Time to complete work: 4 hours

Exercise I

According to the option, write and debug a program using a dynamic data structure: Binary Trees.

1. Develop a program that constructs a binary tree T. Describe a logical procedure or function that determines whether some element E is included in the tree T.
2. Develop a program that builds a binary tree T. Describe a procedure or function that determines the number occurrences of some element E in the tree T.
3. Develop a program that constructs a binary tree T. Describe a procedure or function that calculates the sum of the elements of a non-empty tree T.
4. Develop a program that constructs a binary tree T. Describe a procedure or function that determines the value of the largest element of a non-empty tree T.
5. Develop a program that constructs a binary tree T. Describe a procedure or function that determines the maximum depth of a non-empty tree T, i.e., the number of branches in the longest path from the root to the leaves.
6. Develop a procedure that constructs a binary tree T. Describe a procedure or function that tests trees T1 and T2 for equality.
7. Develop a program that constructs a binary tree T. Describe a procedure or function that determines whether the tree T contains at least two identical elements.
8. Develop a program that constructs a binary tree T. Describe a procedure or function that determines the arithmetic mean of all elements of the tree T.
9. Develop a program that constructs a binary tree T. Describe a procedure that replaces all negative elements in the tree T with their absolute values. values.
10. Develop a program that constructs a binary tree T. Describe a procedure that swaps the maximum and minimum elements of a non-empty tree T, all of whose elements are distinct.

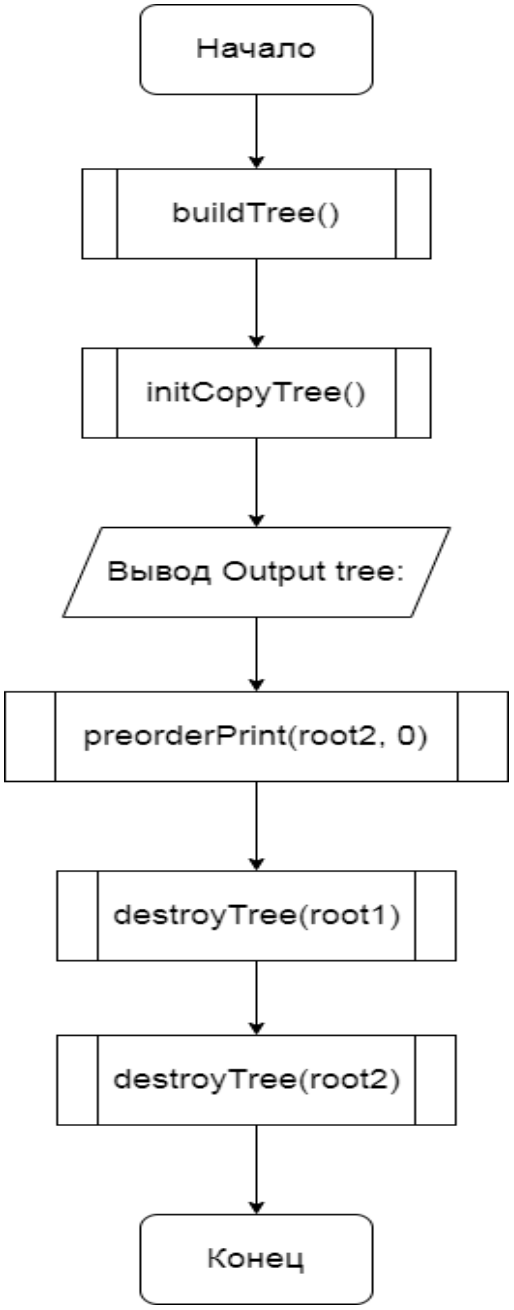
11. Develop a program that constructs a binary tree T. Describe a procedure that prints elements from all leaves of tree T.
12. Develop a program that constructs a binary tree T. Describe a procedure or function that determines the length of the path from the root to the nearest vertex with element E; if E is not included in the tree, take -1 as the answer.
13. Develop a procedure that constructs a binary tree T. Describe the procedure copy(T,T1), which constructs T1 – a copy of the tree T.
14. Develop a program that constructs a binary tree T. Describe a procedure or function that assigns to the parameter E an element from the leftmost leaf of the non-empty tree T.
15. Develop a program that constructs a perfectly balanced binary tree T. Describe a logical procedure or function that determines whether some element E is included in the tree T.
16. Develop a program that builds a binary tree T. Describe a procedure or function that determines the number occurrences of some element E in the tree T.
17. Develop a program that constructs a perfectly balanced binary tree T. Describe a procedure or function that calculates the sum of the elements of a non-empty tree T.
18. Develop a program that constructs a binary tree T. Describe a procedure or function that determines the value of the largest element of a non-empty tree T.
19. Develop a program that constructs a binary tree T. Describe a procedure or function that determines the maximum depth of a non-empty tree T, i.e., the number of branches in the longest path from the root to the leaves.
20. Develop a procedure that constructs a binary tree T. Describe a procedure or function that tests trees T1 and T2 for equality.
21. Develop a program that constructs a binary tree T. Describe a procedure or function that determines whether the tree T contains at least two identical elements.
22. Develop a program that constructs a binary tree T. Describe a procedure or function that determines the arithmetic mean of all elements of the tree T.
23. Develop a program that constructs a binary tree T. Describe a procedure that replaces all negative elements in the tree T with their absolute values. values.
24. Develop a program that constructs a binary tree T. Describe a procedure that swaps the maximum and minimum elements of a non-empty tree T, all of whose elements are distinct.
25. Develop a program that constructs a binary tree T. Describe a procedure that prints elements from all leaves of tree T.

Control questions

1. What such tree?
2. Classification trees.
3. Which tree they call binary?
4. What such knot, descendant, ancestor, leaves, root?
5. Description structures node.
6. What such bypass? Types bypasses
7. Which dynamic structures data to you are known?
8. Which main advantage tree before list?

Example execution tasks I

13. Develop a procedure that builds a binary tree T. Describe the procedure copy(T,T1), which builds T1 – a copy of the tree T.



Execution work

1. Mathematical model

Input data: n – the number of elements to insert into a binary tree (of type int), req – arbitrary meaning key node tree (type int) *Weekend data:* n elements binary tree and the output of tree leaves

Intermediate quantities: counters i, lvl

2. Algorithm

Let's watch block diagram (rice.

Left) buildTree() – building a tree

initCopyTree() – initializing the copy

preorderPrint() - formation conclusion V in the form of a tree

destroyTree() - removal nodes trees from allocated memory

3. Program

```
#include <iostream>

usingnamespace std ;

struct TreeNode {
int data ;
    TreeNode * left ;
    TreeNode * right ;
} ;

TreeNode *root1 =NULL, *root2 =NULL;

void buildTree();
void initCopyTree();
void insertFromChild(TreeNode*, int);
void insertFromRoot(int);
void destroyTree(TreeNode*);
```

```
void preorderPrint(TreeNode*, int);
void copyTree(TreeNode*, TreeNode*);

int main(){
    setlocale(LC_ALL, "rus");

    buildTree();
    initCopyTree();

cout<<"Output tree: \n";
    preorderPrint(root2, 0);


    destroyTree(root2);
    destroyTree(root1);
return0;
}

void insertFromChild(TreeNode *leaf, int key){
if(key < leaf->data){
if(leaf->left !=NULL)
            insertFromChild(leaf->left, key);
else{
            leaf->left =new TreeNode;
            leaf->left->data = key;
            leaf->left->left =NULL;
            leaf->left->right =NULL;
}
}
else{
if(leaf->right !=NULL)
            insertFromChild(leaf->right, key);
else{
            leaf->right =new TreeNode;
            leaf->right->data = key;
            leaf->right->left =NULL;
            leaf->right->right =NULL;
}
}
}

void insertFromRoot(int key){
if(root1 !=NULL){
        insertFromChild(root1, key);
}
else{
        root1 =new TreeNode;
        root1->data = key;
        root1->left =NULL;
        root1->right =NULL;
}
}

void buildTree(){
int n;
cout<<"Enter count of elements to insert in bintree:\n";
cin>> n;
cout<<"Input elements:\n";
for(int i =0; i < n; i++){
int req;
```

```
cin>> req;
        insertFromRoot(req);
}
}

void destroyTree(TreeNode *leaf){
if(leaf !=NULL){
        destroyTree(leaf->left);
        destroyTree(leaf->right);
delete leaf;
}
}

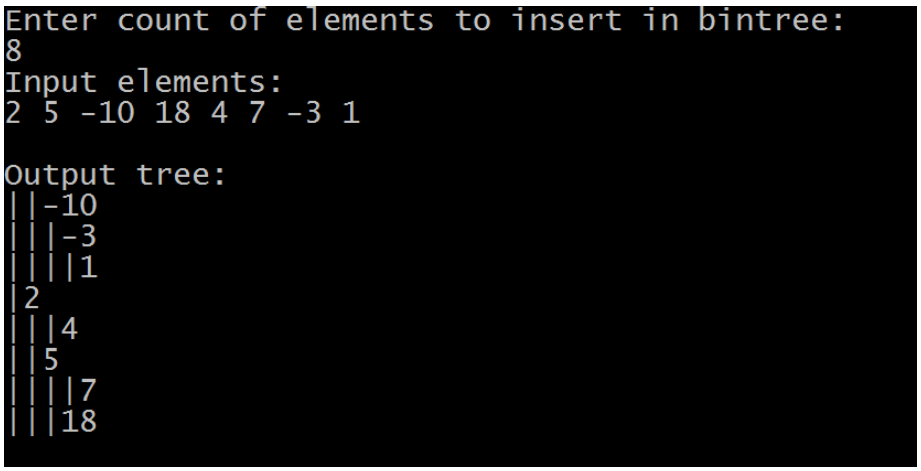
void preorderPrint(TreeNode *root, int lvl){
if(root ==NULL){
return;
}
    preorderPrint(root->left, ++lvl);
for(int i =0; i < lvl;++i)
cout<<"|";
cout<< root->data <<'\n';
    lvl--;
    preorderPrint(root->right, ++lvl);
}

void initCopyTree(){
if(root1 !=NULL&& root2 ==NULL){
        root2 =new TreeNode;
        root2->data = root1->data;
        root2->left =NULL;
        root2->right =NULL;
        copyTree(root1, root2);
}
}

void copyTree(TreeNode *rootFrom, TreeNode *rootTo){
if(rootFrom->left !=NULL){
        rootTo->left =new TreeNode;
        rootTo->left->data = rootFrom->left->data;
        rootTo->left->left =NULL;
        rootTo->left->right =NULL;
        copyTree(rootFrom->left, rootTo->left);
}
else{
        rootTo->left =NULL;
}

if(rootFrom->right !=NULL){
        rootTo->right =new TreeNode;
        rootTo->right->data = rootFrom->right->data;
        rootTo->right->left =NULL;
        rootTo->right->right =NULL;
        copyTree(rootFrom->right, rootTo->right);
}
else{
        rootTo->right =NULL;
}
}
```

4. Result work programs



18. Vector

Lab 10 examined dynamic arrays. Only the necessary memory was allocated for their storage. Lab 16 demonstrated that working with pointers and dynamic memory allocation requires great care and precision. To make life easier for programmers, the STL standard library, which contains many very useful structures and classes, is available. STL appeared quite a while ago, proved very convenient, and quickly became a standard part of C++. STL is the most efficient library for C++ currently available.

This library was used to study strings (string is an integral part of the STL). Let's take a closer look at the library.

Today, there are a great many implementations of the STL standard library that follow the standard but also offer their own extensions, which is a plus on the one hand, but not very good on the other, since it is not always possible to reuse the code with another compiler.

Let's look at the most frequently used features of the STL library. STL actually contains a large number of classes, but first, let's look at the most popular ones that are most likely to appear in other people's code. These classes are more than sufficient to solve 99% of real-world problems.

vector - collection elements type T, saved V array, increasing By measure necessity. For Togo, to begin usage this collections, turn on #include
<vector>.

list - collection elements type T, saved, How bidirectional connected list.
For Togo, to begin usage this collections, turn on #include <list>.

A map is a collection that stores pairs of values pair<const Key, T>. This collection is designed to quickly look up a value T by key. const Key. Anything can be used as a key, such as a string or an int, but it's important to remember that the key's main feature is the ability to apply a comparison operation to it. Fast value lookup by key is achieved because pairs are stored in sorted order. This collection also has a corresponding drawback—insertion speed. the new pair is inversely proportional to the number of elements stored in the collection, Because simply appending a new value to the end of the collection won't work. Another important thing to remember when using this collection is that the key must be unique. To start using this collection, include #include <map>.

A set is a collection of unique const Key values—each of which is also a key—that is, simply put, a sorted collection designed for quickly finding a specific value. The requirements for a key are the same as for a map key. Naturally, using it for this purpose doesn't make sense if you want to store simple data types in it; at a minimum, you need to define your own class storing key-value pairs and defining a comparison operation by key. This collection is very convenient if you want to avoid storing the same value twice. To start using this collection, include #include <set>.

A multimap is a modified map that does not require a unique key. - That There is, If You you will produce search By key, That to you will return Not one meaning, A kit

values stored with this key. To start using this collection, include #include <map>.

Multiset - The same applies to this collection. It doesn't require unique keys, which means it's possible to store duplicate values. However, it's possible to quickly find values by key if you define a custom class. Since all values in map and set are stored sorted, we can very quickly find the required value by key in these collections, but inserting a new element into T will be somewhat more expensive than, say, in a vector. To start using this collection, include #include <set>.

Let's consider more details `vector` . He named V honor mathematical term "vector" And means numbered subsequence objects.

18.1 Concept vectors. Format descriptions

Vector Can introduce, How one-dimensional dynamic array.

Format descriptions:

```
#include<vector>
…
vector<base type> vector_name(number_of_vector_elements);
```

Where

`vector` – official word;
`base type` – type vector elements ;
`vector_name` – correct With points vision C++ identifier;
`number_of_vector_elements` - constant or variable integer type.

For example,

```
#include <vector>
int n;
cin >> n;
vector <int> a(n);                    // 1
//reading:
for (int i = 0; i < n; i++)           // 2
    cin >> a[i];                      // 3
                                      // 4
// вывод:
for (int i = n - 1; i >= 0; i--)
    cout << a[i] << " ";
```

Let's take a closer look at the new code. Line `1` creates a new vector. First, write the word vector, then the type of the vector elements in angle brackets (for example, `int` or `double`), then the vector name and, in parentheses, the number of elements in it. You can't read an entire vector in C++, so we'll read each number and store it in the vector element with the corresponding number (lines 2-3).

Any processing vectors, So same, How And array, is being carried out element by element. For appeals To element vectors, need to indicate number V square in brackets

after its name. Any arithmetic expression is allowed inside the parentheses, and an individual vector element can be treated as a regular variable. If you specify a number in parentheses that exceeds the vector's size (or a negative number), the program will compile but will run incorrectly or even crash. This is because such an access falls into a memory area that is not related to our vector.

So, a vector is almost the same as an array; its elements can be accessed just like array elements. However, vectors have some advantages. You don't need to specifically manage their size, as a vector knows its size. Changing the size is as simple as removing or adding elements at runtime.

To use a vector in a program, you need to include the vector header file and specify the std namespace to use.

Conclusion:

- Vectors are part STL And relate To consistent containers.
- Sequential containers - This ordered collections, V in which each element occupies a specific position.

According to standard C++, any container must contain methods `begin()` , `end()` , `size()` , `max_size()` , `empty()` , and `swap()` .

18.2 Main methods Vectors

`size()` - returns quantity elements V vector.
`capacity()` – returns number elements, which vector Maybe contain without allocation of additional space.
`emplace(pos, value)` - inserts the element `value` at the position pointed to by the iterator `pos` .
`push_back(element)` – add an element to the end `of the vector` , and `pop_back(element)` – delete last element `vector-` a
`insert(***)` - three options (method overload) for insertion into any area in vector, first parameter - position inserts, given iterators, the rest indicate on container, or quantity And container, or couple iterators indicating from which position to which position from another container to take data. `erase` **(** iterator or iterator from, And before **)** - deletes element or sequence of elements from a `vector`
`begin()` - returns iterator, pointing on start collections.
`end()` – returns iterator, pointing on end collections. At this it points not to the very last element, but to an imaginary element after the last one. `at(index)` – method access, To elements collections, V difference from operator [], checks for collection bounds and throws an exception if so.

`clear()` – deletes all elements of a collection. If it contains class objects, it calls their destructors. If it contains pointers to objects, you'll have a memory leak (`memory leaks=`), since `delete` No one will call for you.

IN addition To functions direct access To elements, elements Vectors can be obtained using iterators.

18.3 Iterators

Iterator - this is an object, which is capable of sorting through elements container class without the user needing to know the implementation of a particular container class.

Iterators are usually used in pairs, one of which is needed for indicates the current iteration, and the second serves to indicate the end of the container. Iterators are created using standard methods such as `begin()` and `end()`. Function `begin()` returns pointer on first element, and `end()` - to an imaginary non-existent element following the last one.

Vector uses the most feature-rich iterator type, the RandomAccessIterator, which allows traversing the container in any order and modifying the vector's contents during traversal. However, modifying the vector may invalidate the iterator.

Example counting amounts elements vectors at help iterators:

```
vector<int> the_vector;
    vector<int>::iterator the_iterator;

    for (int i=0; i < 10; i++)
        the_vector.push_back(i);
    int total = 0;
    the_iterator = the_vector.begin();
    while (the_iterator != the_vector.end()) {
      total += *the_iterator; /* Please pay attention attention, What
access To an element can be accessed by dereferencing the iterator */
++the_iterator;
    }
    cout << "Total=" << total << endl;
```

Vectors are iterable objects. It's possible to loop through all the values of a vector using an iterator, for example:

```
#include <iostream>
#include <vector>

int main()
{
    std::vector<int> numbers;      // empty vector
    numbers.push_back(5);
```

```
    numbers.push_back(3);
    numbers.push_back(10);
    for(int n : numbers)   /* n - iterator, running through iterable Object
numbers */
        cout << n << "\t";      // 5    3   10

    std::cout << std::endl;
    return 0;
}
```

или

```
#include <vector>
#include <iostream>
int main() {
     std::vector<int> array { 0, 1, 2 };
     array.resize(7); // изменяем длину array на 7
     std::cout << "The length is: " << array.size() << '\n’;
     for (auto const &element: array)
          std::cout << element << ' ';
     return 0;
}
```

Результат:

```
The length is: 7
0 1 2 0 0 0 0
```

Лабораторная работа № 18
Usage vectors at decision tasks

TARGET WORKS:

- Improvement skills developments programs V environment MSVStudio programming
- Improvement skills V programming With using vectors
- Study process formations vectors
- Study operations With elements vectors

EXECUTION WORKS: V in accordance With option make up And Implement programs.
Time to complete the work: 4 hours

Exercise I

Using the program obtained during laboratory work 7 in Task II, implement the ability to save and process data using a vector.

Exercise II

1. Develop a program corresponding to the option using the existing descriptions, debug her And re-decide With using vectors.
2. Prepare kit tests, confirming right operation of the program.
3. Design report, turning on V him production tasks codes programs and the results of their work.

1. Given description:

```
typedefloat * Vector [100];
Vector x;
```

Considering, What All elements vectors x excellent from `NULL` , describe

- function `max(x)` For location more from numbers, on which refer to the elements of the vector `x` ;
- function `void Inp(x)` - formations vectors *x* ;
- function `void Out(x)` - conclusion numbers, on which refer elements vectors `x` .

2. Given description:

```
typedefloat * Vector [100];
Vector x;
```

Considering, What All elements vectors x excellent from `NULL` , describe

- function negl(x), the value of which is the first element of the vector `x` , referring on negative numbers, or `NULL` , If there are none;
- function `void Inp(x)` - formations vectors *x* ;
- function `void Out(x)` - conclusion numbers, on which refer elements of vector *x* .

3. Given description:

```
typedefloat * Vector [100];
Vector x;
```

Considering, What All elements vectors x excellent from `NULL`, describe

- logical function `same(x)` value which is true `(TRUE)` if in vector `x` there are at least two links pointing to the same numbers, otherwise – `false` ;
- function `void Inp(x)` - formations vector `x` ;
- function `void Out(x)` - conclusion numbers, on which refer elements of vector `x` .

4. Given description:

```
typedefloat * Vector [100];
Vector x;
```

Considering, What All elements vectors x excellent from `NULL` , describe

- function `void unique(x)`, which V vector x All elements, referring to equal numbers, replaces with the first of these elements;
- function `void Inp(x)` - formations vectors *x* ;
- function `void Out(x)` - conclusion numbers, on which refer elements of vector `x` .

5. Given description:

```
typedefloat * Vector [100];
Vector x;
```

Considering, What All elements vectors x excellent from `NULL` , describe

- function `min(x)` For location the smallest from numbers, on which refer to the elements of the vector `x` ;
- function `void Inp(x)` - formations vectors `x` ;
- function `void Out(x)` - conclusion numbers, on which refer elements of vector `x`.

6. Given description:

```
typedefloat * Vector [100];
Vector x;
```

Considering, What All elements vectors `x` are distinct from `NULL` , describe
- function `posit(x)` , meaning which is first from elements vectors `x` , referring on positive numbers, or `NULL` , If such No;
- function `void Inp(x)` - formations vectors `x` ;
- function `void Out(x)` - conclusion numbers, on which refer elements of vector `x` .

7. Given description:

```
typedefloat * Vector [100];
Vector x;
```

Considering, What All elements vectors `x` excellent from `NULL` , describe
- logical function `same(x)` meaning which is true `(TRUE)` if in vector `x` there are no links pointing to the same numbers, otherwise – `false` ;
- procedure `Inp(x)` - formations vectors `x` ;
- procedure `Out(x)` - conclusion numbers, on which refer elements vectors `x` .

8. Given description:

```
typedefloat * Vector [100];
Vector x;
```

Considering, What All elements vectors `x` excellent from `NULL` , describe
- procedure `unique(x)` , which V vector `x` All elements, referring by equal numbers, replaces with the first of these elements;
- procedure `Inp(x)` - formations vectors `x` ;
- procedure `Out(x)` - conclusion numbers, on which refer elements vectors `x` .

9. Given description:

```
typedefloat * Vector [100];
Vector x;
```

Considering, What All elements vectors x excellent from `NULL` , describe

- procedure `del(x)` , which V vector x All links indicating replaces duplicate elements with the `NULL value` ;
- procedure `Inp(x)` - formations vectors *x* ;
- procedure `Out(x)` - conclusion numbers, on which refer elements vectors `x` excellent from `NULL` .

10. Given description:

```
typedefloat * Vector [100];
Vector x;
```

Considering, What All elements vectors `x` excellent from `NULL` , describe

- function `NOL(x)` value which is first from elements vectors `x` , referring to numbers equal to 0, or `NULL` if there are none;
- procedure `Inp(x)` - formations vectors `x` ;
- procedure `Out(x)` - conclusion numbers, on which refer elements vectors `x` .

Exercise III

IN in accordance With option write And debug program, using a data structure: a) *an array of string references;* b) *vector of strings;*

One of the possible representations of a “long text” is to divide it into sections (strings) of equal length and creating an array of references to these strings:

```
constant d = …, n = …; //length lines And maximum number string typedefchar
string [d];
line * Text[n];
```

n lines in the text , then the last elements of the array are `NULL` ; at the beginning of the array of links `are NULL` should not be. If the text operation specifies the number of a missing line, i.e., the array element with this number is `NULL` , then such an operation is not performed).

1. Using given performance text, describe
   - function `int KolStrok(T)` for counting numbers lines V text `T` ;
   - `the void Inp(T)` function , which reads a sequence of characters from the input file up to the first * (- the end-of-text symbol) and forms the text `T from them` (the last line, if necessary, can be supplemented with spaces);
   - function `void out(T)` , printing line by line text `T` .

2. Using this text representation, describe the logical function `Elem(T, i, j, With),` checking, There is whether in text T line with number i, and, if present, assigning the j-th character of this string to parameter c.

3. Using given performance text, describe `void` function

- function `void Change(T, i, j)` , changing in some places `i` -yu And `j` - yu lines of text `T` ;
- `void Inp(T)` function , which reads from the input file a sequence of characters up to the first * (- the end-of-text symbol) and forms from them text `T` (last line, If necessary, fill in with spaces);
- function `void out(T)` , printing line by line text `T` .

4. Using given performance text, describe
   - function `voidSwap(T, i, j)` , changing in some places `i` -yu And `j` -yu lines of text `T` ;
   - `void Inp(T)` function , which reads from the input file a sequence of characters up to the first * (- the end-of-text symbol) and forms from them text `T` (last line, If necessary, fill in with spaces);
   - function `void out(T)` , printing line by line text `T` .

5. Using given performance text, describe
   - function `void Copy(T, i, j )`, replacing `i-` yu line text `T` on copy `j` - and lines;
   - function `void Inp(T),` reading from input file a sequence of letters up to the first * (- the end-of-text symbol) and forming the text `T` from them (the last line, if necessary, should be supplemented with spaces);
   - function `void out(T),` printing line by line text `T` .

6. Using given performance text, describe
   - function `void Add(T, i, j),` adding after `i` -th lines of text `T` copy `j-` th row;
   - `void` function `Inp(T),` which reads from the input file a sequence of characters up to the first * (- the end-of-text symbol) and forms from them text `T` (last line, If necessary, fill in with spaces);
   - function `void out(T),` printing line by line text `T` .

7. Using given performance text, describe
   - function `void delete(T, i),` removing i-th line from text `T` ;
   - `void` function `Inp(T),` which reads from the input file a sequence of characters up to the first * (- the end-of-text symbol) and forms from them text `T` (last line, If necessary, fill in with spaces);
   - function `void out(T),` printing line by line text `T` .

8. Using given performance text, describe

- `void` function `find(T, c, f, i, j)`, which assigns `f` the value `true` if the letter *c* is contained in the text `T`, and if `f = true`, which assigns to the parameters `i` and `j` the "coordinates" of the first occurrence of this letter, `i` *is* the line number, and `j` – position number in this line;
- `void` function `Inp(T)`, which reads from the input file a sequence of characters up to the first * (- the end-of-text symbol) and forms from them text `T` (last line, If necessary, fill in with spaces);
- function `void out(T)`, printing line by line text `T`.

9. Using given performance text, describe

- function `find(T, c)`, determining, How many once litera `with` enters V text `T`;
- `void Inp(T)` function, which reads from the input file a sequence of characters up to the first * (- the end-of-text symbol) and forms from them text `T` (last line, If necessary, fill in with spaces);
- `void` function `out(T)`, printing line by line text `T`.

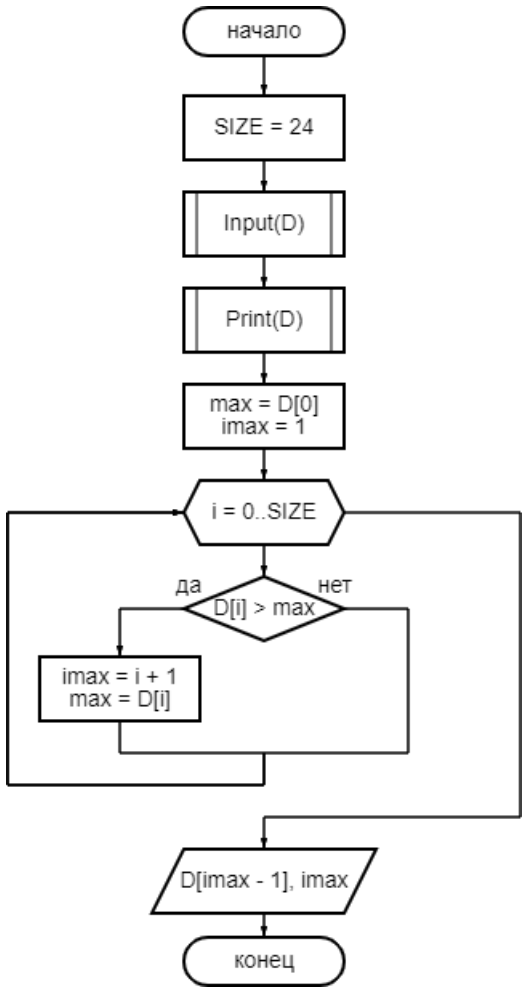
Control questions

1. What such STL?
2. Which varieties collections to you are known?
3. What such vector V C++?
4. What format descriptions vectors?
5. What are peculiarities processing vectors?
6. How to carry out access To element vectors?
7. What such sequential containers?
8. What such iterator?
9. What such iterable object?
10. How organize bypass vectors at help iterator?

Example execution tasks I And II laboratory work 17

Exercise :
Array D contains 24 atmospheric pressure readings for each hour of the day. Determine which atmospheric pressure reading was the highest and at what time it was recorded.

Solution:
The atmospheric pressure values are the elements of the 24-element vector D, and the time values are the element indices. Solving the problem boils down to finding the maximum element in the array and determining its index.

1. Algorithm	2. Description functions used .
	`void Input( vector < int >& arr )` – function for inputting array elements arr `void Print( vector < int >& arr )` - function conclusion elements array arr to the screen

3. Program code

```
#include <iostream>
#include <vector>

usingnamespace std;
constint SIZE = 24;

void Input(vector<int>&arr);
void Print(vector<int>&arr);
int main() {
```

```
	setlocale(LC_ALL, "Rus");

	// we create our vector from 24 elements. By default They are initialized zeros
	vector < int > D(SIZE);

	cout << " Enter array from 24 elements: \n" ;
	Input(D);
	cout << " Array values pressure: \n" ;
	Print(D);

	int max = D[0], imax = 1;
	for (int i = 0; i < SIZE; i++)
		if (D[i]> max) {
			imax = i + 1;
			max = D[i];
	}

	cout << " Maximum meaning pressure: " << D [ imax - 1 ]<< " V " << IMAX << " h.\n" ;
	system( "pause" );
	return 0;
}

void Input(vector<int>&arr) {
	for (int i = 0; i < SIZE; i++) {
		cout << i + 1 <<"-йэлемент: ";
		cin >>arr[i];
	}
}

void Print(vector<int>&arr) {
	// we are deducing on screen, using iterators
	for ( auto i = arr .begin(); i != arr .end();
		i ++ ) cout <<* i << " " ;
	cout << endl;
}
```

4. Result work programs.

```
 Введите массив из 24-х элементов:
1-й элемент: 737
2-й элемент: 736
3-й элемент: 736
4-й элемент: 738
5-й элемент: 736
6-й элемент: 737
7-й элемент: 741
8-й элемент: 743
9-й элемент: 741
10-й элемент: 743
11-й элемент: 746
12-й элемент: 749
13-й элемент: 752
14-й элемент: 752
15-й элемент: 750
16-й элемент: 747
17-й элемент: 746
18-й элемент: 747
19-й элемент: 746
20-й элемент: 745
21-й элемент: 745
22-й элемент: 743
23-й элемент: 747
24-й элемент: 750
 Массив значений давления:
737 736 736 738 736 737 741 743 741 743 746 749 752 752 750 747 746 747 746 745 745 743 747 750
 Максимальное значение давления: 752 в 13 ч.
Для продолжения нажмите любую клавишу . . .
```

Exercise II

Given description:

```
typedefloat * Vector [100];
Vector x;
```

Considering, What All elements vectors `x` excellent from `NULL` , describe

- procedure `Middle(x)` , defining average arithmetic numbers, referenced by the elements of the vector `x` ;
- procedure `Inp(x)` - formations vectors `x` ;
- procedure `Out(x)` - conclusion numbers, on which refer elements vectors `x` .

Solution:

```
#include <fstream>
#include <iostream>
#include <cstdlib>
#include <cstdio>
#include <iomanip>
#include <windows.h>

using namespace std;
void new1();
ifstream fin;
ofstream fout;

typedef float* Vector[100];

void newX();
void print();
void del(Vector, int);
int printArr(Vector);

float Middle(Vector, int);

int main() {
	SetConsoleCP(1251);
	SetConsoleOutputCP(1251);

	Vector x;
	int N = 0;
	short n;
	do {
		system("cls");
		cout << "---------Menu -------- " << endl;
		cout << "[1] - Создать файл чисел " << endl;
		cout << "[2] - Finish writing V file new number " <<
		endl; cout << "[3] - Show numbers from file" << endl;
```

```
		cout << "[4] - Download numbers from file V array" << endl;
		cout << "[5] - Process array: Define average arithmetic" << endl; cout << "[6] - Clear
		dynamic memory" << endl;
		cout << "[7] - Выход" << endl;
		cout << "Выбор: ";
		cin >> n;

		switch (n) {
			case 1: new1(); break;
			case 2: newX(); break;
			case 3:
				system("cls");
				print();
				cout << endl;
				system("pause");
				break;
			case 4:
				system("cls");
				N = printArr(x);
				cout << endl;
				system("pause");
				break;
			case 5:
				if (N == 0) cout << "For calculation average arithmetic
material numbers required execute paragraph [4]";
				else cout << "Average arithmetic numbers equals: " << Middle(x, N); cout
				<< endl;
				system("pause");
				break;
			Case 6:
				if (N == 0) cout << "For execution cleaning memory it is required
at first use, created array, having completed paragraph [4]"; else
				{
					del(x, N);
					cout << "Памятьчиста)";
				}
				cout << endl;
				system("pause");
				break;
		}
	} while (n != 7);
	system("PAUSE");
	return 0;
}


float Middle(Vector X, int n) {
	float s = 0;

	for (int i = 0; i < n; i++)
		s += *X[i];
	s /= n;
```

```
	return s;
}

void new1() {
	fout.open("data.txt");
	char ch = 'd';
	if (fout) {	//If file exists, That
		cout << "file exists -> attempt records V file will lead To loss data.
Let's continue (d/n)";
		cin >> ch;
		if (ch == 'n') return;
	}
	int i = 0;
	do {
		cout << i << " -> ";
		float fX;
		cin >> fX;
		fout << fX << endl;
		cout << "\n продолжим? (d/n)";
		cin>>ch;
		i++;
	} while (ch!='n');
	fout.close();
}
void newX() {
	ofstream app("data.txt", ofstream::app);
	cout << "введите вещ. число: ";
	float fX;
	cin >> fX;
	app << fX << endl;
	app.close();
}
void del(Vector X, int n) {
	for (int i = n - 1; i >= 0; i--)
		delete X[i];
}
void print() {
	fin.open("data.txt");
	if (!fin) {	//If file Not exists, That
		cout << "file Not created -> necessary produce recording data V file"; return;
	}
	int i = 0;
	cout << "Вещественные числа в файле:\n";
	while (!fin.eof())
		if (!fin.eof()) {
			float fX;
			fin >> fX;
			if (!fin.eof())
				cout << i << " -> " << fX << endl;
			i++;
```

```
        }
        fin.close();
}
int printArr(Vector X) {
        fin.open("data.txt");
        if (!fin) {       //If file Not exists, That
                cout << "file Not created -> necessary produce recording data V file"; return 0;
        }

        int i = - 1;
        cout << "Material numbers we are unloading from file.\n";
        while (!fin.eof())  {
                        i++;
                        float fX;
                        fin >> fX;
                        if (!fin.eof()) {
                                X[i] = new float;
                                *X[i] = fX;
                                cout << i << " -> " << *X[i] << endl;
                        }

        }

        fin.close();
        return i;
}
```

19. Sets

19.1 Sets How computing structures: definition, description and limitations

A set is a finite, unindexed collection of similar elements that are logically related to one another. The nature of these relationships is implied by the programmer and is not controlled by the language syntax. A set is an iterable object of the set class, a container that can store unique elements (that is, each element can only appear in the set once). To use the set container, you must include the header file.
<set>.

Format descriptions has view:

```
#include<set>
…
set<base type > set_name;
```

Here **set_name -** correct identifier;
set - reserved word (set);
<basic type> - base type elements sets. For example, a set of elements of type int is declared as follows:

```
set<int>SetInt;
```

A set is a data structure equivalent to sets in mathematics. A set consists of distinct elements of a given type and supports operations such as adding an element to the set, removing an element from the set, and checking whether an element belongs to the set.

In the STL library, the set container is implemented using a balanced binary search tree (red-black tree), so sets in the STL are stored as an ordered structure, which allows you to iterate over the elements of the set in ascending order of their values.

19.2 Operations over sets

Set is container, So What has All standard For function container :

- Job With set elements :

`S.swap(S2)` - changes content containers in places,
`S.insert(a)` - insert element a,
Instead of insert , you can use the emplace function , which checks whether the element already exists before adding it, and if not, adds it.
`S.erase(S2)` - deletes subsequence elements,
`S.clear()` - cleaning container set,
`S.count()` - quantity elements V container,

For checks accessories element set The `count` method is used . This method returns the number of occurrences of the passed parameter in the given container, but since all elements in the set unique, then `count` For type `set` Always returns 0 or 1. That There is for verification accessories values `x` set `S` Can use the following code:

```
if (S.count(x)) { ...
```

`S.find(a)` - find element a in a container,

This method returns *iterator* on the element, and if the element is not found, it returns iterator `end()` (i.e., on a dummy element following the last element of the set. Using this method, you can check whether an element belongs to a set like this:

```
if (S.find(x) != S.end()) { ...
```

`S.lower_bound(a)` - first element, Not less how a,
`S.upper_bound(a)` - first element, more how a,
`S.equal_range(a)` - pair elements, first - lower border elements with such same values, What And a, second - top border elements With such the same values as a.

- Job With memory:

`S.size()` - container size ,
`S.max_size()` - maximum size container.

- Job With container:

`S.begin()` - pointer on beginning of the container,
`S.end()` - pointer on end of the container,
`S.rbegin()` - reversible pointer on end of the container,
`S.rend()` - reversible pointer on start container.

Unification sets:

Unification sets; result contains elements first sets, supplemented with missing elements from the second set

If us given two sets And us necessary unite their V third, then we use set.insert():

M, N - sets, which necessary unite, O - resulting set:

```
O.insert(M.begin(), M.end()); //insert in set ABOUT set M from start to finish
O.insert(N.begin(), N.end()); //insert in set ABOUT set N from start to finish
```

Intersection sets:

We use set_intersection(), Where we'll find intersection sets set1and set2, which will be written to the resulting set_out set. We loop through the data set from beginning to end and insert identical elements into the result, starting from the beginning:

```
set_intersection(set1.begin(), set1.end(), set2.begin(), set2.end(),
inserter(set_out, set_out.begin()));
```

Addition sets: is being implemented similarly unification, So How in a set, elements are not repeated

Difference sets:

Located difference sets a And b (from first subtracted second), c - result of the difference:

```
set_difference(a.begin(),a.end(), b.begin(),b.end(), inserter(c, c.begin()));
```

Example 1:

```
#include <string> // let's connect lines string
#include <iostream> // include cout

using namespace std; // to Not write std::

int main()
{
set <string> S; // инициализация set

S.insert("January");
S.insert("February");
S.insert("March");
S.insert("April");
cout<<S.size()<<'\n'; // будет выведено 4
cout<<S.max_size()<<'\n';
cout<<*(S.begin())<<'\n'; // April
cout<<*(S.rbegin())<<'\n'; // March
cout<<S.count("January")<<'\n'; // 1
cout<<*S.lower_bound("January")<<'\n'; // January
cout<<*S.upper_bound("January")<<'\n'; // March
cout<<*(S.equal_range("January")).first<<'\n'; // January
cout<<*(S.equal_range("January")).second<<'\n'; // March
cout<<*S.find("January")<<'\n'; // January
S.erase("January");
cout<<S.count("January")<<'\n'; // 0
S.clear();
cout<<S.size()<<'\n'; // 0
cin.get(); // we are expecting pressing user keys

return 0; // we issue 0 - rule good manners
}
```

Example 2: Dana non-empty subsequence symbols And set, consisting from the signs of arithmetic operations and numbers from 0 to 9. It is required to be displayed on the screen

symbols, encountered V this sequences And belonging to a given set.

```
      #include <iostream>
      #include <set>
      #include <string.h>
      #include <algorithm>

      using namespace std;

      int main() {
          cout<<"Enter subsequence characters:\n";
      string str_vvoda;
          getline(cin, str_input);
          set <char> set_str, set_const = {'+','-
                                ','*','/','0','1','2','3','4','5','6','7','8','9'};
          for (char i : str_input)
              set_str.insert(i);
          set <char> set_out;
          set_intersection(set_const.begin(), set_const.end(),set_str.begin(),
set_str.end(), inserter(set_out, set_out.begin()));
      cout<<"Symbols, encountered V this sequences And belonging to a given set: ";
      for (char i : set_out)
              cout << i << ' ';
          return 0;
      }
```

19.3 Iterators

With iterators of the `set container` You can perform increment (which means moving to the next element) and decrement (moving to the previous element). Iterators can be compared for equality and inequality. Iterator comparison operations using "<", "<=", ">", ">=" are not possible, and it is also not possible to use operations to add a number to an iterator.

Dereferencing iterator (application unary operator *) Returns the value of the element of the set pointed to by the iterator.

The set has `a begin() method ,` which returns an iterator to the first element of the set, and `the end () method` , which returns a dummy iterator to the element, next for last one element V set. Like this in this way, You can display all the elements of a set like this:

```
   set <int> S;

   set <int>::iterator it;

   for (it = S.begin(); it != S.end(); ++it)
       cout << *it << " "
```

Thanks to to that, What sets are stored V orderly view, All The elements will be displayed in ascending order of values.

IN standard C++11 allowed enumeration everyone element sets using `range-based` cycle:

```
for (auto elem: S)
    cout << elem << " ";
```

Elements Also will be withdrawn V in order increase.

For conclusion elements V in order decrease Can use `reverse_iterator` similarly vectors:

```
for (auto it = S.rbegin(); it != S.rend(); ++it)
    cout << *it << " ";
```

Functions deletions elements can accept iterator V quality parameter. In this case, the element pointed to by the iterator is removed. For example, to remove the smallest element:

```
S.erase(S.begin());
```

But to remove the last (largest) element in `a set`, you can't use `a reverse_iterator`, you need to take a regular iterator pointing to end(), decrement it, and remove it:

```
auto it = S.begin();
-- it;
S.erase(it);
```

Laboratory Job No. 19
Usage sets at decision tasks

TARGET WORKS:

- Improvement skills developments programs V environment MS Visual Studio programming
- Improvement skills V programming With using sets
- Study process formations sets
- Study operations With elements sets

EXECUTION WORKS: V in accordance With option make up And Implement programs. Time to complete the work: 4 hours

Exercise I

1. You are given 100 integers ranging from 1 to 50. Determine how many of them are Fibonacci numbers and how many numbers begin with 1 or 2.

2. A non-empty sequence of words made of lowercase Russian letters is given; between adjacent words there is a comma, and after the last word there is a period. Print in alphabetical order of voiceless consonants (k, p, s, t, f, kh, ts, ch, sh, shch), which are included in every odd word and are not included in at least one even word.

3. Given a text of numbers and lowercase Latin letters followed by a period. Determine, what kind letters - vowels (a, e, i, O, And) or consonants - more V this text.

4. A non-empty sequence of words made of lowercase Russian letters is given; between adjacent words there is a comma, and after the last word there is a period. Print in In alphabetical order, all consonants that are not part of any word.

5. The text is given from lowercase Latin letters, followed by followed by a period. Display the first occurrences of letters in the text, preserving their original mutual order.

6. A non-empty sequence of words made of lowercase Russian letters is given; between adjacent words there is a comma, and after the last word there is a period. Print in alphabetical order of all voiced consonants (b, c, g, d, zh, z, l, m, n, r) that are included in more than one word.

7. The text is given from lowercase Latin letters, followed by followed by a period. Display on the screen all letters that appear in the text at least twice.

8. A non-empty sequence of words made of lowercase Russian letters is given; between adjacent words there is a comma, and after the last word there is a period. Print in alphabetical order of all voiced consonants (b, v, g, d, zh, z, l, m, n, r) that are included in at least one word.

9. The text is given from lowercase Latin letters, followed by followed by a period. Display all the letters in the text on the screen once.

10. A non-empty sequence of words made of lowercase Russian letters is given; between adjacent words there is a comma, and after the last word there is a period. Print in alphabetical order of all vowel letters that do not appear in more than one word.

11. Dan text, for which follows dot. IN alphabetical in order withdraw on screen

(one by one once) all lowercase Russians vowels letters (A, e, and, oh, y, y, uh, me), included in this text.

12. IN increasing in order type All whole numbers from range from 1 to 255, representable as n2 + m2, n, m > 0.

13. A non-empty sequence of words made of lowercase Russian letters is given; between adjacent words there is a comma, and after the last word there is a period. Print in alphabetical order of all consonants that are included in only one word.

14. Withdraw on screen All words given offers, which consist from those the same letters as the first word of the sentence.

15. Given a non-empty sequence of words consisting of lowercase Russian letters and digits, with a space between adjacent words and a period after the last word. Form sets from the lowercase letters of each word and print all the lowercase vowels of each word.

16. In descending order, print all integers from 1 to 255 that are representable as n2 + 2k2 but not as 7ij + j + 3, where n, k, i, j ≥ 0.

17. A non-empty sequence of words made of lowercase Russian letters is given; between adjacent words there is a comma, after the last word there is a period. Print in alphabetically, the vowels that appear in each word.

18. Given a non-empty sequence of words made up of lowercase Russian letters and numbers, with a space between each word and a period after the last word. Print all the voiceless consonants (k, p, s, t, f, kh, ts, ch, sh, shch) that are not part of any word.

19. List all the words in a given sentence that consist of the same letters as the first word of the sentence.

20. A non-empty sequence of words made of lowercase Russian letters is given; between adjacent words there is a comma, and after the last word there is a period. Print in in alphabetical order all the voiced consonants (b, v, g, d, zh, z, l, m, n, r) that are included in every odd word and are not included in any even word.

21. A non-empty sequence of words consisting of lowercase Russian letters and numbers is given. There is a space between adjacent words, and a period follows the last word. Determine, which letters: vowels or consonants are there more in each word.

22. Given a non-empty sequence of words made up of lowercase Russian letters and digits, with a space between each word and a period after the last word. Print all the digits for each word in ascending order.

23. Print in reverse alphabetical order all letters that appear in the text at least three times.

24. Given a non-empty sequence of words made up of lowercase Russian letters and numbers, with a space between each word and a period after the last word. Print all the voiceless consonants (k, p, s, t, f, kh, ts, ch, sh, shch) that appear in each word.

25. Given a non-empty sequence of words from lowercase Russian letters and numbers, between neighboring in words - space, for last one in a word - dot. Type All

deaf consonants letters, which are included V each odd word And voiced consonants that appear in every even word.

26. Given whole number n from 2 to 127. Using method sieves Eratosthenes", print in descending order all prime numbers from the range [n .. 2n].

Exercise II

Given arbitrary sets A , B , and C with C++ constraints, illustrate the truth of the following properties of set operations and relations between them (With lower index V formulas is used complement of a set to the corresponding base type T , i.e. $B_t = T - B$, where B : set of T).

1. $A \setminus B = A \cap B_T$;
2. $A \setminus (A \setminus B) = A \cap B$;
3. $B \cup (A \setminus B) = A \cup B$;
4. $B \cap (A \setminus B) = \varnothing$;
5. $A \setminus (B \cup C) = (A \setminus B) \cap (A \setminus C)$;
6. $A \setminus (B \cap C) = (A \setminus B) \cup (A \setminus C)$;
7. $A \cap B \subset A \cup B$;
8. $A = (A \setminus B) \cup (A \cap B)$;
9. $(A \setminus B) \cap (A \cap B) = \varnothing$;
10. $(A \setminus B) \cap B = \varnothing$;
11. $A \setminus B = A \setminus (A \cap B)$;
12. $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$;
13. $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$;
14. $(A \setminus B) \setminus C = (A \setminus C) \setminus B$;
15. $A \setminus (B \cup C) = (A \setminus B) \cap (A \setminus C)$;
16. $A \cup (B \setminus C) \supset (A \cup B) \setminus C$;
17. $(A \setminus B) \cup C \supset (A \cup C) \setminus B$;
18. $(A \setminus B)_T = A_T \cup (A \cap C)$;
19. $A \cup (B \cup C) = (A \cup B) \cup C$;
20. $A \cap (B \cap C) = (A \cap B) \cap C$;
21. $(A \cap B_T)_T = A_T \cup B$;
22. $(A \cup B = A \cap B_T;$
$)_T \quad T \quad \cup B_T;$
23. $(A \cap B = A$
$)_T \quad T$
24. $A \cap (B \setminus C) = (A \cap B) \setminus (A \cap C)$;
25. $A \setminus (B \cup C) = (A \setminus B) \setminus C$.

Tasks For independent work

In options 1-14, a non-empty sequence of characters is given. The task is to construct and display a set whose elements are the characters occurring in the sequence:

1. Numbers from '0' to '9', letters from 'A' to 'F' And from 'X' to 'Z'.
2. Letters from 'G' to 'N' And numbers from '0' up to '9'.
3. Signs punctuation.
4. Letters from 'A' to 'Z' And numbers from '0' up to '5'.
5. Letters from 'T' to 'X' And signs punctuation.
6. Numbers from '5' to '9' And signs arithmetic operations.
7. Signs arithmetic operations And signs punctuation.
8. Signs punctuation And letters from 'e' to 'n'.
9. Letters from 'f' to 'k' And signs punctuation.
10. All characters of the text in alphabetical order that appear in the given sequence more than twice.
11. All symbols given sequences, incoming V her By alone once.
12. Numbers And signs arithmetic operations.

13. All characters of the text in alphabetical order, appearing in the given sequence at least twice.
14. Signs arithmetic operations And binary relationships.

Control questions

1. Give it to me definition sets.
2. How is described set V program, written on C++?
3. What it's called basic type For sets?
4. Which type let's say V quality basic For sets?
5. What it's called power sets? What such empty set?
6. How is being carried out access To elements sets?
7. Which operations acceptable over sets?
8. Which methods premises elements in set You do you know?
9. Can whether withdraw set X entirely?
10. How is being carried out conclusion elements sets?
11. Maybe whether set contain some identical elements?
12. Maybe whether set contain elements different types?

Example execution tasks I laboratory work

Task : Given a non-empty sequence of symbols and a non-empty set, you must construct and display in standard format a set whose elements are the objects of the given set that occur in the sequence.

```
#include <iostream>
#include <set>
#include <string>
#include <algorithm>
#include <windows.h>
#include <conio.h>

int main() {
using namespace std;
setlocale(LC_ALL, "Rus");
SetConsoleCP(1251);
SetConsoleOutputCP(1251);

string subsequence_1;
cout << "Enter the sequence of characters: ";
getline(cin, subsequence_1);

set<char> set_1;
for (int i = 0; i < (int)subsequence_1.size(); i++) {
	set_1.insert(subsequence_1[i]);
}

cout << "Enter nonemptyset(Esc or Enter - exit): "; set<char> set_2;
while (true) {
	char symbol;
	symbol = _getch();
	if (symbol == 13 || symbol == 27) // 13 - button code Enter, 27 - code
		Esc break;
```

```
		cout << symbol << " ";
		set_2.insert(symbol);
	}

	set<char> result;
	// the intersection of set_1 and set_2 is placed in the third set result
	set_intersection(set_1.begin(),	set_1.end(),	set_2.begin(),	set_2.end(),
	inserter(result, result.begin()));

	cout << "\n\nIntersection of sets:
	"; for (auto now : result)
		cout << now << " ";

	return 0;
}
```

20. Modules

20.1 Modular programming

Modular programming - This organization programs as a collection of small independent units called modules, whose structure and behavior are governed by specific rules. Using modular programming simplifies program testing and error detection. Hardware-dependent subtasks can be strictly separated from other subtasks, improving the portability of the resulting programs.

Module – a functionally complete fragment of a program. In many languages (But far Not Necessarily) is being drawn up V in the form of a separate file With original code or named continuous part of the program. Some languages provide for unification modules in bags.

20.2 Modular programming means languages C And C++

The module logically consists of two files: a source file and a header file.

The source code file (module.cpp) includes function definitions, as well as definitions of global variables and constants (if any); first line usually connects heading file the same module:

#include "module.h"

The header file (module.h) includes function prototypes, constant definitions, and global variable declarations—but only for those module elements that other modules should know about.

If any module uses this one, you need to include its header file (in double quotes - this indicates that the header file is not a system one, but your own): #include "module.h"

The main module of a program usually contains only the main function and does not have a header file (since the main function is not used in other modules)

Some modules include only a header file—for example, one containing definitions of global constants.

Recording type extern int Var;

Which indicates, What V one from files With original code on At the top level, a global variable is defined: int Var;

A declared variable can be used even if it is not defined in the file. There is no definition of it in the source code.

If several modules use the same constants, they usually are defined in the header file; constant declarations are not used

The concept of modular programming provides a deeper understanding of the steps involved in creating an executable program.

Assembly (build) executable the program consists of from 3 main stages:

1) on stage **preprocessing (preprocessing)**

directives preprocessor (For example, #include) are replaced contents

the header file specified in them, as a result, the source code file is supplemented with prototypes of the functions specified there and declarations of global variables - due to this, the file can call the specified functions (the prototype in front) and use global variables (declaration ahead).

The preprocessor can create temporary files on disk, which, however, they are removed after the assembly is completed

2) At the **compilation stage** , tables of functions and global variables defined in each source file are created, and all defined functions are translated into machine language (compiler errors may be detected during translation into machine language). The output of the compiler is module.obj files (object files) for each module used in the program.

3) During the linking phase, all used (called) functions and global variables are bound to the table in which they are defined; if the definitions are not found in any table, a linking error occurs (unresolved external symbol...). The result of linking is the program.exe file (for MVS, the name is the same as the project name) – an executable file.

What at assembly is happening With library functions?

Function prototypes are located in header files (iostream, math.h, stdio.h, string.h, etc.)

Object files with function definitions already translated into machine language are pre-assembled into libraries (files with the .lib or .dll extension). Definitions from static libraries (.lib extension) are added to the program's executable file at the linking stage, while definitions from dynamic libraries (.dll extension) are not added to the executable file; instead, the link to the corresponding function indicates that it is in a dynamic library, and when it is called, the library is accessed. Therefore, static libraries are needed only at the linking stage, while dynamic libraries are also needed at the program execution stage.

Laboratory Job No. 20
Usage vectors at decision tasks

TARGET WORKS:

- Improvement skills developments programs V environment MS Visual C++ programming
- Improvement skills V programming With using modular approach

EXECUTION WORKS: V in accordance With option make up And implement programs.
Time to complete work: 2 hours

Exercise I

With a prototype, called functions don't have to be in the same file as the calling function. That is, the function body declarations can be placed in another file, which must be included in the original file using the **`#include directive`** .

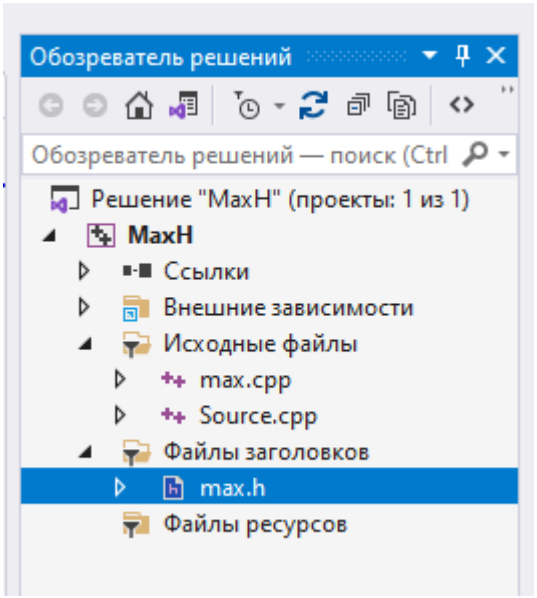
Create a program that contains a call to the function for calculating the maximum of two numbers:

Possible solution this tasks has view:

1) Header file **`max.h. His text:`**

```
#pragmaonce
int max( int a , int b );
```

This file (**`max.h`**) is added V folder Header files :



2) **`File Source.cpp contains original code project:`**

```
/* inclusion heading file max.h, containing the prototype
   of the function max */

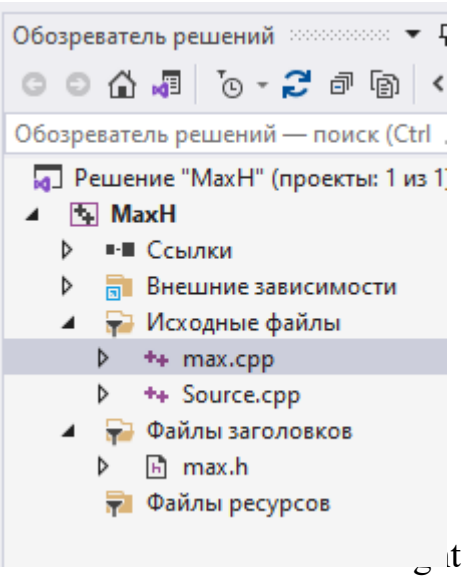
#include <iostream>
```

```
#include "max.h"

using namespace std;

int main()
{
     int x, y, z;
     cout<< "\n one by one enter X And at \n";
     cin>>x>>y;
     z = max(x, y);        /* call functions
     */ cout << "z=" << z;
     return 0;
}
```

3) Description functions **max** contained in a separate file - file implementation: **max.cpp** , located in the directory from which the source program is launched , and has the following form:




```
int max (int a, int b)
{
    int c;          //
    c=(a>b)?a:b;    //   body
    functions return c;     //
}
```

A function call is an exp t parts operator assignments:

```
z = max(x,y);
```

during the execution of which the values of the arguments **X** and **at** are substituted instead of parameters **A** and **b,** respectively (passing parameters to a function by value). After executing the function body, the return value is passed to the function call site and assigned to the variable **z** . The function description is in the **max.cpp file** , so To include a file in a program, you must specify the preprocessor directive **#include "max.h" in the program text** .

The function description may be in one file with the main program. In this case, the directive **#include "max.h"** is not specified, and the function description is placed instead.

Exercise II

In accordance with your version of Lab 16, Task II , formulate the functions implementing the basic algorithms for working with dynamic data structures as a module. Organize the program to call the module.

Control questions

1. IN how consists of principle modular programming?
2. What such module?
3. From what kind parts logically consists of module?
4. What includes heading file (module.h) /file With source code (module.cpp)?
5. IN what section programs indicated connection modules?
6. IN how consists of difference spheres actions variables V module And functions?
7. From what kind sections consists of module And What is described V each from sections?
8. How is being organized module on language C/C++?
9. How is happening assembly programs?
10. What at assembly is happening With library functions?

21. Classes in C++

Classes and objects in C++ are the core concepts of object-oriented programming (OOP). Object-oriented programming is an extension of structured programming, in which the core concepts are classes and objects. The difference between the C++ programming language and C is that C does not have classes, and therefore the C language does not support OOP, unlike C++.

To understand why classes are really needed, let's draw an analogy with an everyday object, for example, a bicycle. A bicycle is an object that was built according to blueprints. These blueprints play the role of classes in OOP. Thus, classes are descriptions, diagrams, and blueprints from which objects are created. Now it's clear that to create an object in OOP, you first need to create blueprints, or classes. Classes have their own functions, which are called class methods. A bicycle moves by rotating the pedals; if we consider a bicycle from an OOP perspective, the pedaling mechanism is a class method. Each bicycle has its own color, weight, and various components—all these are properties. Moreover, each created object may have different properties. With one class, you can create There's an unlimited number of objects (bicycles), each of which will have the same set of methods. There's no need to worry about the internal implementation of the pedaling mechanism, wheels, or braking system, as all of this will already be defined in the class. Having clarified the purpose of the class, let's give it a proper definition.

Classes in C++ are an abstraction describing methods and properties that do not yet exist. objects. Objects – a concrete representation of an abstraction, with its own properties and methods. Objects created based on a single class are called instances of that class. These objects may have different behavior and properties, but they will still be objects of the same class. In OOP, there are three main principles for constructing classes:

7. **Encapsulation** - This is a property that allows a class to combine both data and methods that work with it, while hiding implementation details from the user.
8. **Inheritance** - This property, allowing create new descendant class on basis already existing, while all the characteristics of the parent class are assigned to the child class.
9. **Polymorphism** - property classes, allowing use objects classes With the same interface without information about the type and internal structure of the object.

We'll examine each class construction property in detail as needed, but for now, just remember these three. Now, back to classes, let's first look at the class declaration structure.

```
1     // announcement classes V C++
2     class/*name class*/
3     {
4       private:
5       /* list properties And methods For use inside class */
6       public:
7       /* list methods available to others functions And objects programs */
```

```
8       protected:
9       /*list funds, available at inheritance*/
10    };
```

Announcement class it begins With reserved key words `class` , after whom is written Name class. IN curly in brackets, lines 3 - 10 is announced body class, moreover after closing brackets It is necessary to put a semicolon, line 10. In the class body, three access specification labels are declared, lines 4, 6, 8, After each label, you must put a colon. In line 4 access specifier label declared `private` . All methods and properties of the class declared after specifier access `private` will be available only inside class. In line 6 the access specifier `public is declared` , all methods and properties of the class declared after the access specifier `public` will be accessible to other functions and objects in the program. For now, let's stop there, the access specifier `protected` disassemble Now Not we will, Just remember, What he is. When declaring a class, you don't have to declare three access specifiers, and you don't have to declare them in this order. However, it's best to decide on the order of access specifier declarations and try to stick to it. Let's develop a program declaring a simple class containing a single function that prints a message.

```
1. // classes.cpp: defines point entrance For console applications.

2. #include stdafx.h
3. #include <iostream>
4. usingnamespacestd;
5. // start announcements class
6. classCppStudio // Name class
7. {
8. public: // specifier access
9. voidmessage() // function (method class) output message on screen
10.     {
      a. cout << "website: cppstudio.com\ntheme: Classes and Objects in C +
         +\n";
11.     }
                                    12.     }; // end of class declaration CppStudio

                                             13.     intmain(intargc, char* argv[])
                                                                          14.     {
                                          15.     CppStudio objMessage; // object declaration
                                 16.     objMessage.message(); // class function call message
                                                                  17.     system("pause");
                                                                         18.     return0;
                                                                               19.     }
```

IN lines 7–14 A class named `CppStudio is defined` . Class names are usually capitalized, and subsequent words in the name must also begin with a capital letter. This combination of letters is called camel case, as the alternating uppercase and lowercase letters resemble the silhouette of a camel. The class body declares the `public access specifier` , which allows other functions to call class methods declared after `public` . exactly That's why V main functions, V line 19 We were able to call

function `message` (). In class `CppStudio` Only one function is declared, which has no parameters and displays a message on the screen, line 12. Class methods are the same functions, only they are declared inside the class, so everything that relates to functions is also relevant for class methods. Class declarations are performed in the same way as function declarations, that is, a class can be declared in a separate file or V the main thing file, Later We'll see How This is being done. In line 18 the variable is declared `objMessage` type `CppStudio` , so here's the variable `objMessage` - This object class `CppStudio` . Like this in this way, A class is a complex data type. Once an object of the class is declared, its methods can be used. There is only one method – a function `message` (). To do this, we turn to To method `objMessage` object through point, as shown in line 19, As a result, the program will display a text message (see Figure 6.1).



Drawing 6.1 - Classes in C++

21.1 set - functions And get – class functions

Every object has certain properties or attributes that characterize it throughout its lifetime. An object's attributes are stored in variables declared within the class to which the object belongs. Moreover, variable declarations must be made with an access specifier. `private` . Such variables are called data elements. Since data elements are declared `private` , only class methods can access them; external access to data elements is prohibited. Therefore, it is customary to declare V special methods in classes - the so-called `set` And `get` functions, using which Can manipulate elements data. `set` - functions initialize data elements , `get` functions allow you to view values elements data. We'll work on it. `CppStudio` class so that it can store the date in the dd.mm.yy format. To change and view the date, we implement `set` and `get` functions.

```
1. // classes.cpp: defines point entrance For console applications.

2. #include stdafx.h
3. #include <iostream>
4. usingnamespacestd;

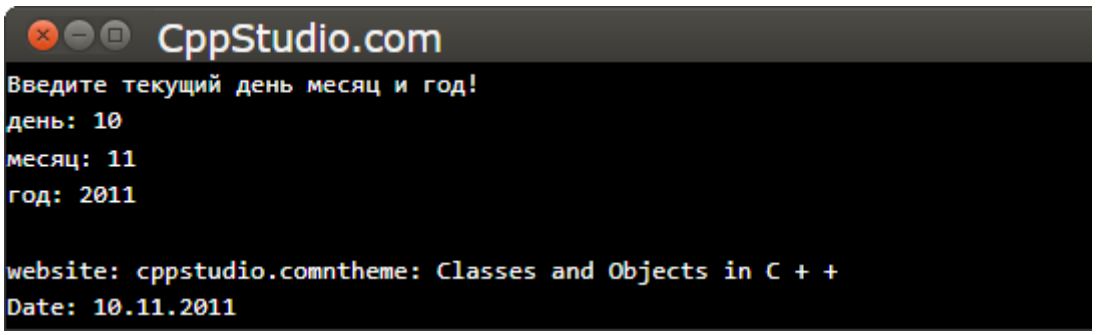
5. classCppStudio // class name
6. {
7. private: // access specifier private
8. intday, // day
9.         month, // month
10.        year; // year
11. public: // access specifier public
```

```
12. voidmessage() // function (method class) output message on screen
13. {
14.         cout << "\nwebsite: cppstudio.comntheme: Classes and Objects in C +
            +\n";
15. }
16. voidsetDate(intdate_day, intdate_month, intdate_year) //
    Set the date to dd.mm.yy
17. {
18.       day   = date_day; // initialization day
19.       month = date_month; // initialization month
20.       year  = date_year; // initialization year
21. }
22. voidgetDate() // display current date
23. {
24.       cout << "Date: "<< day << "."<< month << "."<< year << endl;
25. }
26. }; // конецобъявленияклассаCppStudio

27. intmain(intargc, char* argv[])
28. {
29. setlocale(LC_ALL, "rus"); // locale installation
30. intday, month, year;
31. cout << "Enter current day month And year!\n";
32. cout << "день: ";      cin >> day;
33. cout << "месяц: ";     cin >> month;
34. cout << "год: ";  cin >> year;
35. CppStudio objCppstudio; // object declaration
36. objCppstudio.message(); // class function call message
37. objCppstudio.setDate(day, month, year); // date initialization
38. objCppstudio.getDate(); // display date
39. system("pause");
40. return0;
41. }
```

IN definition class appeared new access specifier `private` , line 9. This access specifier restricts access to variables, which announced after him And to beginning access specifier `public` , lines 9 – 12. Thus, the variables `day` , `month` , `year` can only be accessed by class methods. Functions that do not belong class members cannot access these variables. Data members or class methods declared after the `private access specifier` but before the next access specifier are called private data members. and private class methods. Following the principle of least privilege and the principle of good programming, it is advisable to declare data elements after the `private access specifier` , and class methods after `the public access specifier` . Then, to manipulate data elements, special functions - `get` And `set` . IN Class `CppStudio` We added two method `setDate` () And `getDate` (), in detail let's consider each method. Method `setDate` () defined with 18 By 23 lines. How already previously mentioned, `set` – functions initialize elements data, therefore the method `setDate` () performs exactly such function. That There is

method `setDate` () initializes `day` variables , `month` , `year` . To view, values V closed elements data `the getDate` () function is declared , which returns values from `day` variables , `month` , `year` V as a date. This completes the class definition. In `main` (), as always, we create an object of the class and call its methods through the object (lines 39–41). If elements data were declared after the specifier `public` We could access them just like class methods. The program's output is shown in Figure 6.2.



Drawing 6.2 - Classes in C++

21.2 Constructors

`the CppStudio` class data members that could store date information were declared. When an object of the class was created, We first called `set` – a function to set the current date (thereby initializing the data elements), and then – called `get` - function and saw the corresponding date on the screen. If we had first called `get` – function, instead of a date we would see some numbers – garbage. So, when creating objects, you can immediately initialize the class's data elements, which performs this constructor function. Constructor – a special function that performs the initialization of data elements, and the constructor name must match the class name. An important difference between a constructor and other functions is that it does not return any values, including `void` . In any class should be designer, even If obvious way the designer is not declared (as in the previous class), the compiler provides a default constructor without parameters. Let's modify the `CppStudio class` by adding a constructor to it.

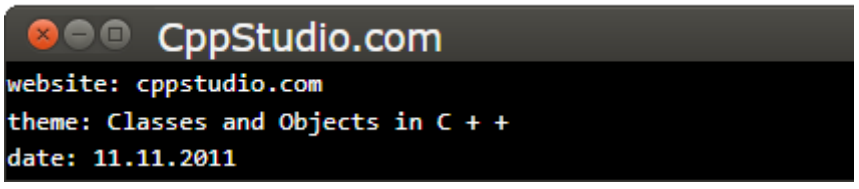
```
1. // classes.cpp: defines point entrance For console applications.

2. #include stdafx.h
3. #include <iostream>
4. usingnamespacestd;
5. classCppStudio // class name
6. {
7. private: // access specifier private
8.       intday, // day
9.       month, // month
10.   year; // year
11.   public: // access specifier public
12.   CppStudio(intdate_day, intdate_month, intdate_year ) //
   class constructor
13.   {
```

```
14.      setDate(date_day, date_month, date_year); //
         call the datesetting function
15.   }
16.   voidmessage() // function (method class) output message on screen
17.   {
18.      cout << "\nwebsite: cppstudio.com\ntheme: Classes and Objects in C +
         +\n";
19.   }
20.   voidsetDate(intdate_day, intdate_month, intdate_year) //
   установкадатывформатедд.мм.гг
21.   {
22.      day   = date_day; // initialization day
23.      month = date_month; // initialization month
24.      year  = date_year; // initialization year
25.   }
26.   voidgetDate() // display current date
27.   {
28.      cout << "date: "<< day << "."<< month << "."<< year << endl;
29.   }
30.   }; // end of class declaration CppStudio

31.   intmain(intargc, char* argv[])
32.   {
33.CppStudio objCppstudio(11,11,2011); // object declaration and
   data member initialization
34.   objCppstudio.message(); // call functions message
35.   objCppstudio.getDate(); // display date
36.   system("pause");
37.   return0;
38.   }
```

The constructor is declared in lines 13 – 16. The constructor has three parameters, through which He receives information O date, V body the constructor is called `set` is a function for setting the date. It was possible to implement the initial initialization of class data elements without `set` - functions, but since this function was provided, it would be more correct to use this function, line 15. IN line 35 We declare a class object, passing three arguments in parentheses after the object name. This is how the initialization of data elements is performed using a constructor (see Figure 6.3).



Drawing 6.3 - Classes V C++

21.3 Announcement class V separate file

Until now, the class declaration was done in the file with the main function and everything worked. Let's say we need to write some program, for this we need take advantage of class `CppStudio` - developed previously class. To use this class, you need to include the file in which it is declared. As we already mentioned, including files is done using preprocessor `#include directives` . But even, If We we can

connect file with class, will appear new the problem is like this as in file with If the class already has a main() function, the compiler will return an error when building the project. The error message is: "Multiple `main` () functions found in the project" – functions." This is why a class must be declared in a separate file so it can be reused. Previously, we declared functions in a separate file; placing a class in a separate file is similar. To do this, three steps are required:

1. add V project heading file `*.h` ;
2. V title file announce user Class, V ours case - `CppStudio` ;
3. connect heading file To program, V ours case - `#include "CppStudio.h"` .

Depending on the development environment, the methods for adding files to a project may differ, but the essence of the task remains the same. In MVS2010, the header file Can add, having called contextual menu(click) right button mice) V "Solution Explorer" by selecting "Create new item". In the dialog box that appears, window we choose necessary us type file – This *.h And we fill in field "File name". As before, choose a meaningful name, usually the same as the class name. Now a new header file has been added to our project – `CppStudio` .h .

In the newly created In the header file, we declare the class and, if necessary, include additional header files. This is what it should look like:

```
1. // header file CppStudio.h

2. #include <iostream>
3. usingnamespacestd;

4. // class announcement
5. classCppStudio // class name
6. {
7. private: // access specifier private
8. intday, // day
9.         month, // month
10.        year; // year
11. public: // access specifier public
12. CppStudio(intdate_day, intdate_month, intdate_year ) // class constructor
13. {
14.        setDate(date_day, date_month, date_year); //
           call the datesetting function
15. }
16. voidmessage() // function (method class) output message on screen
17. {
18.        cout << "nwebsite: cppstudio.comntheme: Classes and Objects in C +
           +n";
19. }
20. voidsetDate(intdate_day, intdate_month, intdate_year) //
    Set the date to dd.mm.yy
21. {
22.        day   = date_day; // initialization day
```

```
23.         month = date_month; // initialization month
24.         year  = date_year; // initialization year
25. }
26. voidgetDate() // display current date
27. {
28.         cout << "date: "<< day << "."<< month << "."<< year << endl;
29. }
30. }; // конец объявления класса CppStudio
```

In order for the main function to see and use the class we've created, we need to include the class definition in the executable file, along with the main() function. This is done like this:

```
1. // classes.cpp: defines point entrance For console applications.

2. #include stdafx.h
3. // connecting the class CppStudio
4. #include "CppStudio.h"

5. intmain(intargc, char*argv[])
6. {
7. CppStudio objCppstudio(11,11,2011); // announcement object And
   initialization of data elements
8. objCppstudio.message(); // call functions message
9. objCppstudio.getDate(); // display date
10. system("pause");
11. return0;
12. }
```

IN line 5 connected definition class `CppStudio` , only After this, you can create objects of the class, use its methods, etc. The result of the program is exactly the same as before, it has changed only the structure of the project.

21.4 Department interface from implementation

Class interface – a construct that defines the methods and properties provided by a class. Class implementation – this is the way a class implements its functionality. Previously, we didn't separate a class's interface from its implementation, meaning the methods were implemented within the class. Separating a class's interface from its implementation is done to hide how the class operates. Separating the interface from its implementation is accomplished in five steps:

1. add V project heading file `*.h` ;
2. define interface class V title file
3. add V project performing file `*.cpp` ;
4. V executable file execute implementation class;
5. connect heading file To program.

We already know how to add a header file to a project; an executable file is added in the same way; these files are usually given the same names. The class interface should look like this:

```
1. // heading file class CppStudio.h
2. // interface class

3. // announcement class
4. classCppStudio // Name class
5. {
6. private: // access specifier private
7. intday, // day
      a. month, // month
      b. year; // year
8. public: // access specifier public
9. CppStudio(int, int, int); // constructor class
10. voidmessage(); // function (method class) output message on screen
11. voidsetDate(int, int, int); // installation dates V format dd.mm.yy
12. voidgetDate(); // display current date
13. }; // end announcements class CppStudio
```

The class interface contains the declared variables and prototypes of the class methods. Now let's look at the contents of the class method implementation file.

```
1. // file implementation class CppStudio.cpp

2. #include <iostream>
3. usingnamespacestd;
4. // let's connect interface class To file his implementation
5. #include "CppStudio.h"

6. CppStudio::CppStudio(intdate_day, intdate_month, intdate_year ) //
   class constructor
7. {
8. setDate(date_day, date_month, date_year); // call the datesetting function
9. }

10. voidCppStudio::message() // function (method class) output message on the
    screen
11. {
12. cout << "nwebsite: cppstudio.comntheme: Classes and Objects in C + +n";
13. }

14. voidCppStudio::setDate(intdate_day, intdate_month, intdate_year) //
    Set the date to dd.mm.yy
15. {
16. day   = date_day; // initialization day
17. month = date_month; // initialization month
18. year  = date_year; // initialization year
19. }

20. voidCppStudio::getDate() // display current date
21. {
22. cout << "date: "<< day << "."<< month << "."<< year << endl;
    }
```

To link a class interface and its implementation, it is necessary in the implementation file connect heading file With definition class, string 6 (highlighted line). After this, you can declare class methods. Class methods are declared in the same way as functions, except that you must precede the method name with the class name and the unary scope resolution operator "::".

```
1. // syntax announcements methods class outside bodies class
2. /*returned type data*//*name class*/::/*name method*/(/* method
   parameters*/)
3. {
4. // operators
   }
```

Since class methods are declared outside the class body, it's necessary to associate the method implementation with the class, and to do this, we use the binary scope resolution operator. The binary scope resolution operator binds an externally declared method to the class whose name matches the name in the method declaration. This is why the class name and the scope resolution operator must be added to the class method declaration.

So, the class interface is defined, the class methods are declared, all that remains is to include the header file in the executable file with `main` () function and the program is ready.

```
1. // classes.cpp: defines point entrance For console applications.

2. #include stdafx.h
3. // connecting the class CppStudio
4. #include "CppStudio.h"

5. intmain(intargc, char*argv[])
6. {
7. CppStudio objCppstudio(11,11,2011); // announcement object And
   initialization of data elements
8. objCppstudio.message(); // function call message
9. objCppstudio.getDate(); // display date
10.   system("pause");
11.   return0;
12.   }
```

IN line 5 We include the class header file, after which we can create objects of this class.

Some tips on how to shape the interface of the future class and what needs or doesn't need to be done for this.

Laboratory Job No. 21
Usage objects at decision tasks

TARGET WORKS:

– Improvement skills developments programs V MS Visual C++

programming environment

– Improvement skills V programming With using

modular approach

EXECUTION OF WORK: in accordance with the option, draw up and implement programs. Time for completion work: 4 hours

Exercise

Option 1

Describe a class implementing a stack. Write a program using this class to model a T-shaped railway marshalling yard. The program must split a train consisting of two types of cars into two directions (a train of one type of car is formed for each direction). Provide the ability to create a train from a file and from the keyboard.

Option 2

Describe a class that implements a binary tree that has the ability to add new elements, delete existing ones, search for an element by key, and also sequentially access all elements.

Write a program that uses this class to represent an English-Russian dictionary. Provide the ability to generate the dictionary from a file and from the keyboard.

Option 3

Build a class system to describe flat geometric figures: circle, square, rectangle. Provide methods for creating objects, moving them on the plane, resizing them, and rotating them by a given angle.

Write program, demonstrating work With these classes.

Option 4

Create a class definition for a class containing information about an organization's postal address. Provide the ability to separately modify the address's components, create, and destroy objects of this class. Use dynamic fields in the object.

Write program, demonstrating work With this class.

Option 5

Create a class definition for representing complex numbers. Provide operations for addition, subtraction, and multiplication of complex numbers.

Write program, demonstrating work With this class.

Option 6

Create a class description for vector objects defined by the coordinates of their endpoints in three-dimensional space. Provide operations for adding and subtracting vectors to produce a new vector (sum or difference), calculating the scalar product of two vectors, the length of a vector, and the cosine of the angle between vectors.

Write program, demonstrating work With this class.

Option 7

Create a description of a class of rectangles with sides parallel to the coordinate axes. Provide the ability to move the rectangles on a plane, change their sizes, and construct the smallest rectangle containing two given rectangles and a rectangle that is the common part (intersection) of the two rectangles.

Write program, demonstrating work With this class.

Option 8

Compose description class For definitions one-dimensional arrays integers (vectors). Provide the following options:

- appeal To separate element array With control exit beyond array;
- exercise arbitrary borders indices at creation object;
- execution operations element-by-element additions And subtraction of arrays with the same index boundaries;
- Multiplication And division everyone elements array on scalar;

- conclusion on screen element array By given index And the entire array.

Write program, demonstrating work With this class.

Option 9

Compose description class For definitions one-dimensional arrays fixed-length strings. Provide the following options:

- appeal To separate lines array By indexes; array out-of-bounds control;
- execution operations element-by-element clutch two arrays with the formation of a new array;
- mergers two arrays With exception recurring elements;
- conclusion on screen element array By given index And the entire array.

  Write program, demonstrating work With this class.

Option 10

Create a description of a class of polynomials in one variable, defined by the degree of the polynomial and an array of coefficients. Provide methods that implement :

- calculation values polynomial For given argument;
- operation additions, subtractions And multiplication polynomials with the receipt of a new polynomial object;
- conclusion on screen descriptions polynomial.

Write program, demonstrating work With this class.

Option 11

Write a description of a class of one-dimensional arrays of strings, where each string is specified by its length and a pointer to the memory allocated for it. Provide the following capabilities:

- appeal To separate lines array By indices;
- control exit for limits arrays;

- execution operations element-by-element clutch two arrays with the formation of a new array;
- merger two arrays With exception recurring elements;
- displaying an array element and the entire array. Write

program, demonstrating work With this class.

Option 12

Create a description of a class that provides the representation of a matrix of arbitrary size with the ability to change the number of rows and columns, as well as display a submatrix of any size and the entire matrix on the screen.

Write program, demonstrating work With this class.

Option 13

Write Class For work with lines, allowing keep V them as long, integer numbers and perform addition and subtraction operations on them.

Write program, demonstrating work With this class.

Option 14

Describe the class "home library". Provide the possibility working with an arbitrary number of books, searching for a book by any characteristic (for example, by author or by year of publication), adding books to the library, removing books from it, sorting books by different fields.

Written by» program, demonstrating work With this class.

Option 15

Describe the "Notebook" class. Provide the ability to work with an arbitrary number of entries, search for entries by any attribute (for example, last name, date of birth, or phone number), add and delete entries, and sort by various fields. Write a program demonstrating how to work with this class.

Option 16

Describe Class student group". Provide for opportunity work

with a variable number of students, searching for a student by some characteristic (for example, by last name, date of birth, or phone number), adding and deleting records, sorting by different fields. Write a program demonstrating how to work with this class.

Option 17

Describe a class implementing the "real matrix" data type and how to work with it. The class should implement the following operations on matrices:

- addition, subtraction, multiplication, division (multiplication And division by both another matrix and a number);
- calculation reverse And transposed matrices;
- methods, implementing check type matrices (square, diagonal, zero, unit).

Write program, demonstrating work With this class.

Option 18

Describe Class, implementing type data "material matrix" And working with them. The class must implement the following operations on matrices:

- calculation determinant And norms;
- methods, implementing check type matrices (symmetrical, upper triangular, lower triangular);
- comparison on equality- inequality;
- construction to the degree;

Write program, demonstrating work With this class.

Option 19

Describe Class, implementing stack. Write program, using this class to find a passage through a maze.

The maze is represented as a matrix of squares. Each square is either open or closed. Entry into a closed square is prohibited. If a square is open, entry is possible from the side, but not from the corner. Each square is determined his coordinates V matrix. After searches passage

program prints found path V view coordinates squares.

Option 20

Describe the class "subject index". Each index component contains word And numbers pages, on which This word occurs. The permissible number of page numbers per word is from one to ten. The following options should be provided:

- formation pointer With keyboards And from file;
- conclusion pointer;
- conclusion numbers pages For given words;
- removal element from pointer.

Write program, demonstrating work With this class.

Control questions

1. Concept object-oriented programming.
2. Extensions C++ relatively C.
3. What such Class? What are his Features? How? are called specific values of the data type "class"?
4. What V OOP understand under message? How is being implemented message mechanism ?
5. What understand under event-driven model"?
6. What such interface And What is interface class?
7. What understand under fields And methods class?
8. What are differences between classes And structures?
9. Give definitions encapsulation, inheritance And polymorphism
10. Describe general structure announcements class
11. Appeal To components class.
12. Initialization objects.
13. Inheritance properties And access modifiers .
14. Role specifiers access private, public And protected
15. What such set And get functions?
16. What such constructor And For what is it used?
17. What such destructor And For what is it used?
18. How announce Class V separate file And use his V further?
19. At creation instance class at help operator new, Where it is created, on the stack or in dynamic memory.
20. Maybe whether one method class call another?
21. If the members of a class are private and no friend class or function is declared for it, who can access its members ?

Application 1

Order execution laboratory works

1. Familiarize yourself with the theoretical material relevant to the topic of the laboratory work and prepare answers to the proposed questions control questions.
2. From the list of problems, select the ones that match your option. For each problem, develop a mathematical model, algorithm, and solution program.
3. Implement program on a computer.
4. To defend your laboratory work, you must know the answers to the control questions and provide a written report containing:

   A) Name laboratory works; b)

   version number;

   V) solution each tasks, inclusive V myself:

   - condition tasks;
   - mathematical model;
   - algorithm;
   - listing programs With mark teacher O her execution;
   - original kit data And corresponding to him result;
   - analysis results work programs.

Appendix 2 Basic functions

Function	Description	Example
abs(x)	returns module or absolute meaning from **x**	abs(-3.0) = 3.0 abs(5.0) = 5.0
fabs(x)	absolute magnitude (numbers With floating point)	
sqrt(x)	root square from the whole **x,** moreover **x** Not negatively	sqrt(9) = 3
sqrtf(x)	root square fractional **x,** moreover **x** Not negatively	sqrt(9.0) = 3.0
pow(a, b)	construction **A** V degree **b**	pow(2,3) = 8
ceil(x)	rounding **x** to the smallest whole, But Not less how **x**	ceil(2.3) = 3.0 ceil(-2.3) = - 2.0
floor(x)	rounding **x** to nearest smaller whole numbers	floor(12.4) = 12 floor(-2.9) = - 3
fmod(a, b)	calculation remainder departments completely For numbers With floating point (a/b)	fmod(4.4, 7.5) fmod(7.5, 4.4) =
modf(x,p)	extracts the integer and fractional parts (including sign) from a floating-point number	
exp(a)	calculation exhibitors **e** [a]	exp(0)=1
frexp(x)	breaks number **x** With floating point on mantissa and the exponent.	
ldexp(x)	multiplication numbers With floating point on kiss degree two	
acos(x)	arccosine of **x**	
asin(x)	arcsine of **x** , Where **-1.0 <x< 1.0**	asin(1)=1.5708
atan(x)	arctangent **x**	
atan2(a, b)	arctangent With two parameters	
cos(x)	cosine **x** (**x** is being asked V radians)	
sin(x)	sine **x** (**x** is being asked V radians)	
sinh(x)	hyperbolic sinus **x**	
tan(x)	tangent **x**	
tanh(x)	hyperbolic tangent **x**	
log(x)	natural logarithm **x** (base is exponent)	log(1.0)=0.0
log10(x)	decimal logarithm **x**	Log10(10)=1

The table shows the basic mathematical functions, prototypes which are contained in the header file `<math.h>` .

math.h - heading file standard libraries language programming C , designed for performing simple mathematical operations. Most functions rely on floating-point numbers. C++ also implements these functions for compatibility; they are all contained in the cmath header file.

All these functions accept doubles unless otherwise specified. To work with float and long double types, use functions with the suffixes f and l, respectively. All functions that accept or return an angle work with radians.

LITERATURE

1. Goryushkin, A. P. Mathematical logic and theory of algorithms: textbook / A. P. Goryushkin. - Saratov: Vuzovskoe obrazovanie, 2022. - 499 p. - Text: electronic // Digital educational resource IPR SMART: [site]. - URL: https:/ /www.iprbookshop.ru/117296.html, unlimited copies.
2. Afanasyev, S. G. Mathematical logic: a tutorial / S. G. Afanasyev. - Moscow: Ouch Pi Ar Media, 2021. — 82 c. — ISBN 978-5-4497-0963-9. — Text:
   Electronic // Digital educational resource IPR SMART: [website]. — URL: https:/ /www.iprbookshop.ru/103656.html, unlimited copies.
3. Gamova, A. N. Mathematical logic and theory of algorithms: a textbook for students of the Faculty of Mechanics and Mathematics and the Faculty of Computer Science and information technology / A. N. Gamova. - Saratov: Publishing House Saratov University, 2020. — 91 p. — Text: electronic // Digital educational resource IPR SMART: [site]. — URL: https:/ /www.iprbookshop.ru/106266.html, unlimited copies.
4. Khomenko, T. V. Discrete Mathematics. Selected Methods of Set Theory and Mathematical Logic. Laboratory Workshop / T. V. Khomenko. - Astrakhan: Astrakhan state architectural and construction university, EBS ASV, 2020. — 111 p. — Text: electronic // Digital educational resource IPR SMART: [site]. — URL: https:/ /www.iprbookshop.ru/100830.html, unlimited copies
5. Machikina, E. P. Mathematical logic and theory of algorithms: a teaching aid / E. P. Machikina. - Novosibirsk: Siberian State University of Telecommunications and Informatics, 2020. — 86 p. — Text: Electronic // Digital educational resource IPR SMART: [website]. — URL: https:/ /www.iprbookshop.ru/102154.html, unlimited copies