

MINISTRY OF SCIENCE AND HIGHER EDUCATION OF THE RUSSIAN FEDERATION
FEDERAL STATE AUTONOMOUS EDUCATIONAL INSTITUTION OF HIGHER
EDUCATION
"NORTH CAUCASUS FEDERAL UNIVERSITY"

SUGGESTIONS FOR TEACHING
for the performance of practical work
in the discipline "Advanced Language Programming C" for students of the direction of
training
11.03.02 Infocommunication Technologies and Communication Systems
"Intelligent Data Processing in Infocommunication Systems and Networks"

Stavropol 2026

Practicum on the discipline "Advanced Language Programming C" for students of the field of study

11.03.02 Infocommunication Technologies and Communication Systems

The following issues are considered: development of programs of linear, branching, cyclic and mixed structures; one-dimensional and two-dimensional arrays; structures and strings; functions, standard functions of the C++ language.

Each laboratory work indicates its purpose, sets out brief theoretical information, describes the stages of work and task options, and also offers control questions that allow the student to make a self-assessment.

CONTENTS

Practical Lesson No 1. Linear Structure Programs in Language C++.	9
Practical Lesson No 2. Conditional Operator in C++	14
Practical lesson No 3. Switch in C++	17
Practical lesson No 4. Loops. Loop with parameter	19
Practical lesson No 5. Cycle with a precondition	24
Practical lesson No 6. Cycle with postcondition	26
Practical Lesson No 7. One-dimensional arrays	27
Practical Lesson No 8. Multidimensional Arrays	30
Practical lesson No 9. Symbolic data	33
Practical Lesson No 10. Character Strings	41
Practical lesson No 11. Structures in the C++ language	44
Practical Lesson No 12. Associations. Bitfields	48
Practical Lesson No 13 Functions in the C++ Language	51
Practical Lesson No 14. Recursion. Recursive Functions	57

Practical Lesson No 1 Linear Structure

Programs in C++

1. Purpose of the work: Consolidation of knowledge about algorithms and programs of linear structure.

2. Basic information.

First, let's look at an example of a linear structure program in the language S.

Suppose you want to calculate the value of z using the formula

$z = 3(y + 3x) + (x+y)^2$, with x and y values entered using the keyboard.

A calculation program in C using format I/O is as follows:

```
lab2_1.c    calculations    using    the
formulas #include<stdio.h>
#include<conio.h>
#include<math.h>
main(){
    float x,y,z;
    printf(«Enter x,y\n»);
    scanf(«%f%f»,&x,&y);
    z = 3*(y+3*x)+(x+y)*(x+y);
    printf(“x=%8.2f, y=%8.2f, z=%10.4f”,x,y,z);
    getch();
    return 0;
}
```

Let's give a brief explanation. The first line of the program is a one-line comment explaining the purpose of the program. The comment begins with two slash characters in a row and is valid until the end of the line. The comment does not affect the compilation of the program and is intended for the programmer, including the author of the program, because programs are forgotten quite quickly.

Although a comment is optional, it is strongly recommended that you annotate the beginning of the program to distinguish one program from another.

The next three lines of the program are directives for including header files `stdio.h` (header file for I/O), `conio.h` (for console input-output, header file for working with console I/O), `math.h` (header file for working with mathematical functions).

The next line is a description of the main function (the main function of the program). A C (and C++) program consists of a number of functions, of which the main function must be present, it serves as an entry point into the program.

The parameters of any function, including the main function, are enclosed in parentheses. Even if the parameter list is empty (as in this case), parentheses are still used to distinguish the function from a simple variable. After the parameter list, the function body (the list of actions to be performed) is written in parentheses and enclosed in curly braces.

The string `float x,y,z;` contains a description of real variables of the `x`, `y`, `z` type (from `float` - floating, floating-point numbers, can contain both integer and fractional parts). Before using each variable, it must be described, i.e. its type must be specified. Float variables (real with single precision) take up 4 bytes in RAM and are used in non-critical calculations and educational programs. In critical calculations, real variables with double precision, which occupy 8 bytes, are used.

In addition to real variables, C and C++ use integer variables of the `int` type. The amount of RAM used for variables of this type depends on the implementation. In the Borland C++ version 3.1 environment, these variables occupy 2

bytes, in the Borland C++ Builder environment, int variables take up 4 bytes. Variables of other types will be considered as we explore.

The printf and scanf functions are contained in the stdio.h header file and are used for screen output and keyboard input, respectively. The "\n" (backslash n) control characters in the printf function are used to break a new line. Sequences of characters that begin with the backslash character are called escape sequences, so "\n" is one of the escape sequences. The "&" character (ampersand) in the scanf function indicates the address of the variable.

The printf("Enter x,y") operator; means displaying the text "Enter x,y", i.e. the prompt to enter x and y, the next line is scanf("%f%f",&x,&y); – typing x and y values and entering them into the addresses of these variables (the ampersand symbol is used for this).

The operator $z = 3 * (y + 3 * x) + (x + y) * (x + y)$; assigns the variable z the value $3(y + 3x) + (x + y)^2$ the asterisk denoting the multiplication sign.

The printf("x=%8.2f, y=%8.2f, z=%10.4f",x,y,z); is used to output the values x, y, z. In the format string enclosed in quotation marks, the formats of the display are indicated: %8.2f %8.2f %10.4f of the variables x, y, z respectively, the first 2 variables are displayed in 8 positions with two digits after the decimal point, the variable z in 10 positions with 4 digits after the decimal point, the formats are indicated after the "percentage" symbol.

The "f" symbol indicates a float number. The remaining characters in the format string are displayed unchanged. Therefore, the text "x=" will be displayed first, then the value x in the format "%8.2f", then the text "y=", then the value y in the format "%8.2f", then the text "z=" and finally the value of z in the format "%10.4f".

The getch function contained in the conio.h header file waits for any key to be pressed, and the user has the opportunity to view the results of the program. Without using this function, the program will terminate and the text will be displayed

program, and the user will not be able to see the results (the results screen will only appear for a moment).

All functions, except for functions of the void type, return a value of a certain type, which is done by the return a operator, where a is the return value. If the function does not have a type specified (as in this case), then by default the function is of the int type. A return value of 0 means that the program ran without errors.

A control example for the above program. Let $x=3.4$; $y=2.5$. In response to the "Enter x,y" invitation, enter 3.4_2.5 and press ENTER. (Here the underscore means a space). The separator of the integer and fractional parts is a period.

The result of the program is as follows:

$x = \underline{\hspace{1cm}} 3.40, \quad y = \underline{\hspace{1cm}} 2.50, \quad z = \underline{\hspace{1cm}} 72.9100$

Here, too, underscores mean spaces.

This result should be checked using a Windows calculator.

All statements, both those that perform certain actions and those that serve to describe program objects (variables, constants, types, etc.), must end with a semicolon.

The same C++ program with the use of streaming I/O looks like this:

lab2_2.cpp calculations using the

```
formulas #include<iostream.h>
```

```
#include<conio.h>
```

```
#include<math.h>
```

```
main() {
```

```
float x,y,z;
```

```
cout<<"Enter x,y\n";
```

```
cin>>x>>y;
```

$$z = 3 * (y + 3 * x) + (x + y) * (x + y);$$

```
cout<<"x= "<<x<<" y="<<y<<" z="<<z<<endl;
```

```

    getch();
    return 0;
}

```

The difference between the lab2_2.cpp program and the lab2_1.c program is that it uses the streaming input and output contained in the iostream.h file. The output stream cout is the display screen, the input stream cin is the keyboard. The tokens "<<" and ">>" indicate the direction of I/O: to the output stream cout from variables and to variables from the input stream cin. Streaming I/O is formatless, and the compiler determines the format by the type of variable.

The text constant "endl" (newline) means, like the escape sequence "\n", a line break. If a text constant is output that ends with a line break, "\n" can be used, but if variable values are output followed by a line break, endl should be used. The I/O of each I/O argument must be accompanied by the tokens "<<" or ">>".

3. Getting the job done

3.1. Type and compile both of the above versions of the program. Check the results using the Windows calculator.

3.2. To develop programs of linear structure for calculations using formulas in C and C++ according to the variants of tasks.

4. Variants of tasks

Calculate the value of the function given parameter values:

1. $x=2y+3t-z$ at $y=2$; $t=5/(1+y^2)$; $z=4$.
2. $x=3y^2/(4z-2t^2)$ at $t=0.5$; $z=6$; $y=t+2z$.
3. $x=4y^2/(4z-2t^3)$ at $t=1$; $z=3$; $y=2+t$.
4. $x=4y^3-z/t$ at $t=2$; $z=3$; $y=(t+z)$.
5. $x=6t^2-(z+1)/y^2$ at $y=2$; $z=4$; $t=(2+z)$.
6. $x=(8z^2+1)/(y+t^2)$ at $z=1$; $t=2$; $y=t+z$.

7. $x=6t^3-3z^2/(y+1)$ at $t=2$; $z=t+1$; $y=3$.

8. $x=8z/(t+3)-y^2$ at $t=1$; $z=t+2$; $y=4$.

9. $x=6y^2+3(z-t)^3$ at $t=2$; $z=t+1$; $y=3$.

10. $x=6t^2-(z+1)/(y+1)$ at $t=0.5$; $z=6$; $y=t+2z$.

5. Control questions

1. What is a linear structure algorithm? Linear structure program?
2. Which header file contains math functions?
3. How do I output a real number in a field with a given number of positions with a given number of decimal places after a period?

Practical Lesson No2
Conditional Operator in C++

1. Purpose of the work: Consolidation of knowledge about programs of branching structure, compilation of a program with a conditional operator and work with it.

2. Basic information

A branching structure algorithm is an algorithm in which a computational process is carried out either along one or the other branch, depending on the fulfillment of some condition. A branching structure program implements such an algorithm. A branching structure program has one or more conditional statements.

A conditional operator in the C language has the format:

if (condition) operator1; else operator2; (full form) or if
(condition) operator1; (abbreviated form).

The condition is enclosed in parentheses, when it is met, operator1 is executed, if it is not fulfilled, operator2 is executed (if the conditional operator is fully formed). For an incomplete form of a conditional operator, when the condition is met, operator1 is executed, otherwise operator1 is skipped and the operator following the conditional statement is executed.

Operator1 and operator2 can represent simple operators (a single operator), in which case they are not enclosed in parentheses. If operator1 and/or operator2 represent a compound operator (multiple operators), they must be enclosed in curly braces.

As an example, let's give a program for calculating a function by various mathematical expressions at different intervals.

```
razvetvl.c Branching Structure Program #include <stdio.h>
#include <conio.h>
main()
{float x,y;
```

```

printf("Type    x\n");
scanf("%f",&x);
if (x<=0) {y=x*x+1; goto m;}
else if (x<2) {y=x+1; goto m;}
else y=7-x*x;
m: printf(" x=%8.2f y=%8.2f\n",x,y);
getch();
return 0; }

```

3. Getting the job done

Type and compile the above program for calculating a given function, correct the identified errors. Enter several variants of the argument values (in different intervals), calculate the function manually and compare it with the results obtained by the program.

Compose a program of a branching structure using a conditional operator according to the variants of tasks, compile it, enter initial data, and check the result.

4. Variants of tasks

Calculate the value of a function given by different analytic expressions at different intervals:

Options 1, 6 $y =$

$$\left\{ \begin{array}{l} x^2 \text{ at } x \leq -3, \\ 1+x^3 \text{ at } -3 \leq x < 5, \\ x^2/(3+4x) \text{ at } x \geq 5. \end{array} \right.$$

Options 2, 7 $y =$

$$\left\{ \begin{array}{l} 3x + 4 \text{ at } x < 1, \\ x^2/(6+x) \text{ at } 1 \leq x < 4, \\ x^3 + 8 \text{ at } x \geq 5. \end{array} \right.$$

Options 3, 8 $y =$

$$\left\{ \begin{array}{l} (6 - x^2) / 3 \text{ at } x \leq 2, \\ x^2 / (3 + 4x^2) \text{ at } 2 < x \leq 4, \\ 6x^3 - 4x \text{ at } x > 4. \end{array} \right.$$

Options 4, 9. $y =$

$$\left\{ \begin{array}{l} 6x^2 + 3 / (4 - x) \text{ at } x < -2, x \\ x^2 / (3 + x^2) \text{ at } -2 \leq x < 3, \\ (2x^3 + 4) / (5x^2 - 2) \text{ at } x \geq 3. \end{array} \right.$$

Options 5, 10 $y =$

$$\left\{ \begin{array}{l} 4x^2 + 6 \text{ at } x < -1, \\ x^3 / (5 + x) \text{ at } -1 \leq x < 2, 4 \\ x^2 - 3 \text{ at } x \geq 2. \end{array} \right.$$

5. Control questions

1. Which algorithm is a branching structure algorithm?
2. According to the program of section 2, make expressions for the function on intervals.
3. What is a conditional operator? Long and abbreviated forms.
4. What is a compound operator? The format of its notation.
5. What is an unconditioned jump operator?
6. Look for unconditioned jump operators in the sample program.
7. Is it possible to do without unconditional transition operators?

Practical Lesson No3 The Switch in C++

1. Purpose of the work: Consolidation of knowledge about the switch, compilation of a program with a switch and work with it.

2. Basic information

In some cases, the branching of the program depending on the value of a certain variable should be carried out not by 2, but by more branches. For example, let's determine the season by the entered month number.

//switch.c Multiple Choice

```
#include<stdio.h>
#include<conio.h>

int n;

main() {
printf("Enter month number\n");
scanf("%d",&n);
printf("\nSeason: ");
switch(n){
case 1:case 2:case 12:printf("Winter\n"); break;
case 3:case 4:case 5:printf("Spring\n"); break;
case 6:case 7:case 8:printf("Лето\n"); break;
case 9:case 10:case 11:printf("Autumn\n");
break; default:printf("\nMonth number is
incorrect\n");} getch();
return 0;}
```

In the example program, when you enter a month number from 1 to 12, the corresponding time of year is printed on the screen, and if the month number exceeds 12, a message is displayed indicating that the month was entered incorrectly, which is done by using the reserved word of the default language. Each of the constants that enumerate the value of the switch is followed by a colon. The final break statement for each branch is used to interrupt

check loop and go to the end of the switch. If there is no break, the switch does not work correctly - there is a transition to the next branch.

3. Getting the job done

Type and compile the above program using the switch. Correct the identified errors. Enter several values of the month number in this program, make sure that the program works correctly.

Compose a program with a switch according to the task options, compile it, enter the initial data, check the result.

4. Variants of tasks

1. Enter the number of exams $N \leq 20$. Type the phrase "We have successfully passed N exams", agreeing the word "exam" with the number N.
2. Enter a number - the number of the month. Display the number of days in a month (a non-leap year).
3. Introduced number of years ($N \leq 25$). Type the phrase "I am N years", agreeing the word "years" with the number N.
4. Enter the number of books $N \leq 10$. Output the phrase "I took N books from the library" by agreeing the word "book" with the number N.
5. Enter the number of pencils $N \leq 10$. Output the phrase "I bought N pencils", agreeing the word "pencil" with the number N.
6. Introduced Number versions $N \leq 10$. Output phrase "The investigator checked N versions" by agreeing the word "version" with the number N.
7. Enter the number of programs $N \leq 10$. Print the phrase "I have developed N programs", agreeing the word "program" with the number N.
8. Introduced Whole Number $-9 \leq c \leq 9$. Output magnitude Numbers in verbal form, taking into account the sign.
9. Enter the number of vertices $N \leq 10$. Output the phrase "We have conquered N vertices" by agreeing the word "vertex" with the number N.

10. The number of groups $N \leq 10$ is introduced. Output the phrase "At the faculty there are N groups", agreeing the word "group" with the number N.

5. Control questions

1. What is multiple choice?
2. When is the switch used?
3. Why is the break operator used in the switch?
4. Why do I use the reserved word default in the radio button?
5. Block diagram of the switch.

Practical Lesson No 4 Cycles.

Cycle with parameter

1. Purpose of the work: Consolidation of knowledge about cyclic structure programs, compilation of a cycle program with a parameter and work with it.

2. Basic information

A cyclic structure algorithm is an algorithm in which the same part of the program is repeated many times. Such repeatable sections of the computational process are called loops.

A cyclic structure program contains one or more cycles. A distinction is made between deterministic cycles with a predetermined number of repetitions and iterative cycles in which the number of repetitions is known in advance.

To organize a cycle, you need to do the following:

- 1) set the initial value of the variable that changes in the loop before the loop;
- 2) Change the variable before each new loop repetition.
- 3) check the condition for repeating the cycle;
- 4) control the loop, i.e. go to its beginning if it is not finished, or exit it at the end. A variable that changes in a loop is called a loop parameter, and several parameters are possible in one loop.

In the C language, there are 3 types of loops: 1) a loop with a parameter or a loop of the **for type**, 2) a loop with a precondition or a loop of the **while type**, 3) a loop with a postcondition or a loop of the **do ... while** loop. In a for loop, the number of repetitions is known in advance, in cycles of the while and do ... while loop is not known in advance, the condition of the loop repetition is checked: in a while loop - before the cycle, in a do ... while loop - after it ends.

In for and while loops, the repeating part (loop body) consists of a single statement, and if you want to execute multiple statements in a loop, they are enclosed in curly braces to form a compound statement. In a do ... while loop body is placed between the reserved language words (tokens) do and while, curly braces are also required, and the loop body is conventionally indicated by three dots.

In all types of loops, the condition for the continuation of the loop is enclosed in parentheses. For loops, the loop header consists of three sections: initialization (assigning initial values), checking the repeat condition, and modification (changing parameters). The separator between the sections of the for loop header is a semicolon.

When you find the sum of a variable, for example, S, the value 0 is assigned, then the corresponding member of the given sequence is added to this variable in the loop. Let's consider an example of calculating the sum of squares of a natural series of numbers from 1 to n.

$$S = 1^2 + 2^2 + \dots + n^2.$$

The program has the following form:

sum_sq.c, the sum of squares of the natural series

```
#include<stdio.h>
#include<conio.h>
main()
{   int S,n,i;
    clrscr(); printf("Enter
n\n"); scanf("%d",&n);
    for(S=0,i=1; i<=n; i++)
        S+=i*i; printf("n=%d
S=%d\n",n,S);
```

```
    getch();  
    return 0; }
```

The program defines integer variables S, n, i, unlike other languages in C and C++ languages, case sensitive when determining the identifier (name) of the variable, so that S and s are different variables. The clrscr function cleans the screen, which is contained in the header file conio.h. After entering the variable n, a loop of type for follows. In the initialization section, the initial values are assigned to the variables S (in which the sum of squares is accumulated) and i - to the loop parameter, assignments are separated by a comma, the initialization section is separated by a semicolon. In the condition check section, the value of i is compared with n, if i is less than or equal to n, the loop is repeated, otherwise the loop is exited. The modification section consists of an increment operator (increment by 1) of the value of the loop parameter i. $i++$ is equivalent to $i=i+1$. Similarly, in the decrement operation, we have $i-- \Leftrightarrow i=i-1$.

A single $S+=i*i$ operator is executed in a loop; it is a compound assignment operator equivalent to the $S=S+i*i$ operator, so curly braces are not required for the loop body. Increment, decrement, and compound assignment operators make the C++ language concise, reduce the amount of source text (C++ is the most concise of the high-level languages).

When the loop ends, the result is printed (printf statement). The getch function contained in the conio.h header file waits for any key to be pressed, and the user can view the results of the program without having to view the user's screen using the [ALT]+[F5] combination. As mentioned above, when writing functions without clrscr and getch parameters, parentheses should be placed even with an empty parameter list.

Products are calculated in the same way, but before the cycle of a variable, for example, S, in which the value of the product is accumulated,

not 0, but 1 is assigned. A composite assignment operator is also possible here, which has the form $S*=i$ when calculating the factorial; which is equivalent to $S=S*i$. In general, the composite assignment operator is also used for other binary operations (operations with two operands).

3. Getting the job done

Type and compile the above programs using the FOR loop, correct the identified errors. Enter several variants of the argument values in these programs, calculate the result manually and compare it with the results obtained by the program.

Compile programs of cyclic structure according to the variants of tasks, compile them, enter initial data, check the result.

4. Variants of tasks

1. Enter the integer $N>0$. Calculate the sum of N members of the sequence

$$S = \frac{(1+1^2)}{(2+1)} + \frac{(1+2^2)}{(2+2)} + \dots + \frac{(1+N^2)}{(2+N)}.$$

2. Enter an integer $N>0$. Calculate Product

$$P = \frac{1}{(1+1^2)} * \frac{1}{(1+2^2)} * \dots * \frac{1}{(1+N^2)}.$$

3. Enter an integer $N>0$. Calculate Product

$$P = \frac{(1+1)}{(1+1^2)} * \frac{(1+2)}{(1+2^2)} * \dots * \frac{(1+N)}{(1+N^2)}.$$

4. Introduced Whole Number $N>0$. Calculate amount N
Sequence Members

$$S = \frac{(1+1)}{(2+1^2)} + \frac{(1+2)}{(2+2^2)} + \dots + \frac{(1+N)}{(2+N^2)}$$

5. Introduced Whole Number $N>0$. Calculate amount N
Sequence Members

$$S = \frac{(1+12)}{(2+13)} + \frac{(1+22)}{(2+23)} + \dots + \frac{(1+N^2)}{(2+N^3)}.$$

6. Enter an integer $N > 0$. Calculate Product

$$P = \frac{2}{(1+12)} * \frac{2}{(1+22)} * \dots * \frac{2}{(1+N^2)}.$$

7. Enter an integer $N > 0$. Calculate Product

$$P = \frac{(1+13)}{(1+12)} * \frac{(1+23)}{(1+22)} * \dots * \frac{(1+N^3)}{(1+N^2)}.$$

8. Enter the integer $N > 0$. Calculate the sum of N members of the sequence

$$S = \frac{(2+12)}{(1+13)} + \frac{(2+22)}{(1+23)} + \dots + \frac{(2+N^2)}{(1+N^3)}.$$

9. Enter an integer $N > 0$. Calculate Product

$$P = \frac{(3+13)}{(2+12)} * \frac{(3+23)}{(2+22)} * \dots * \frac{(3+N^3)}{(2+N^2)}.$$

10. Enter the integer $N > 0$. Calculate the sum of N members of the sequence

$$S = \frac{(3+12)}{(2+13)} + \frac{(3+22)}{(2+23)} + \dots + \frac{(3+N^2)}{(2+N^3)}.$$

5. Control questions

1. Which algorithm is a cyclic structure algorithm?
2. Types of cycles in the language of C.
3. What are the three sections written in parentheses for a for loop operator?

What is the separator between sections?

4. What are nested loops? Examples:

Practical lesson No 5 Cycle with a precondition

1. Purpose of the work: Consolidation of knowledge about the programs of the cycle with a precondition, drawing up a program of the cycle with a precondition and working with it.

2. Basic information

The number of loop repetitions is not always known in advance, in these cases cycles with a precondition (check before the cycle) or with a postcondition (check after the cycle) are used.

A program to calculate the factorial of n using a precondition loop uses the formula

$$n! = 1 * 2 * \dots * n$$

i.e. the factorial of n is the product of all integers from 1 to n.

fact.c factorial calculation

```
#include<stdio.h>
#include<conio.h>
main()
{ int F,n,i;
  clrscr(); printf("Enter
n\n"); scanf("%d",&n);
  F=1; i=1;
  while(i<n) {F*=i; i++;}
  printf("n=%d n!=%d\n",n,F);
  getch();
  return 0;}
```

3. Getting the job done

Type and compile the above program using a WHILE loop, correct the identified errors. Enter several variants of the argument values in this program, calculate the result manually and compare it with the result obtained by the program.

Compile a cyclic structure program using a WHILE loop according to the variants of the tasks of the laboratory work No5, compile it, enter the initial data, and check the result.

4. Control questions

1. Types of cycles in the language of C.
2. How is the loop continuation condition written in a while loop?
3. What are nested loops? Examples:
4. Can a cycle with a precondition never be executed? In what case will this happen?

Practical lesson No 6 Cycle with postcondition

1. Purpose of the work: Consolidation of knowledge about the programs of the cycle with a postcondition, compilation of a cycle program with a postcondition and work with it.

2. Basic information

In programs that use a postcondition loop, the loop repeat condition is checked after the loop, so the postcondition loop will be executed at least once.

The program for calculating the double factorial of n using a postcondition loop uses the formula

$$n!! = 1*3*5* \dots *n \text{ for odd } n \text{ or } n!! = 2*4*6* \dots *n \text{ for } n \text{ even.}$$

dfact.c Double Factorial Calculation

```
#include<stdio.h>
#include<conio.h>
main()
{int F,n,i;
  clrscr(); printf("Enter
n\n"); scanf("%d",&n);
  F=1;
  if(n%2==0) i=0; else i=-1;//do parity
  condition { i=i+2;
F*=i;}
  while(i<n) ;
  printf("n=%d n!!
=%d\n",n,F); getch();
  return 0; }
```

3. Getting the job done

Type and compile the above program using a loop of type DO... WHILE to fix the detected errors.

Enter several variants of the argument values in this program, calculate the result manually and compare it with the result obtained by the program.

Compile a cyclic structure program using a cycle of the DO... WHILE according to the variants of the tasks of the laboratory work No5, compile it, enter the initial data, and check the result.

4. Control questions

1. Which algorithm is a cyclic structure algorithm?
2. Types of cycles in the language of C.
3. How is the loop continuation condition written in a do ... while?
4. What are nested loops? Examples:
5. Can a cycle with a postcondition not be fulfilled even once? Why?

Practical Lesson No 7 One-dimensional arrays

1. Purpose of the work: Consolidation of knowledge about arrays, compilation of programs with one-dimensional arrays.

2. Basic information

Arrays are a structured data type with elements of the same type. Arrays in C are described in the same way as simple variables, but after the name of the array, the number of its elements is indicated in square brackets. For example, the array is the salaries of employees of the enterprise department, here the number of elements is equal to the number of employees, the array forms the maximum daily temperatures during the year, the number of array elements is the number of days in a year. The number of an array element is called its index, in C and C++ languages the index of an array element starts from zero, so the index of the last element of the array is 1 less than the number of elements in this array.

Here is an example of a program to find the largest element in an array and its index.

poiskmax.c searches for max. electricity in an array.

```
#include<stdio.h>
#include<conio.h>
#define n 10
int i,max,k,a[n];//array size n
main() {
max=a[0]; k=0;
for(i=0; i<n; i++)
scanf("%d",&a[i]); i=0;
for(i=0; i<n; i++)
{if(max<a[i])
{k=i;max=a[i];}}
printf("Maximum number in the %d array, its index %d\n",max,k);
```

```
getch();  
return 0;}
```

Here, the directive `#define n 10` defines the constant `n=10`, then we describe array `a` with the number of elements `n`. Indices of array elements change from 0 to 9. When specifying an element of an array, the operator uses the name of the array and the index value in square brackets. The first loop on `i` in the program is element-by-element input of array `a`. Then comes the search for the maximum element in the array, also in the loop. The variable `max` is compared with the elements of the array, and if the element of the array is greater than `max`, the variable `max` is assigned the value of the element of the array, and the variable `k` is assigned the index value of this element (1 less than its number in the array).

3. Getting the job done

Type and compile the above program, correct the detected errors. Enter the elements of the array, make sure that the maximum element is found correctly.

Compose a program using arrays according to the variants of tasks, compile it, enter the initial data, check the result.

4. Variants of tasks

1. Enter array `A` of 10 elements, find the largest element, and rearrange it with the first element. Output the converted array.

2. Enter array `A` of 10 elements, find the smallest element, and rearrange it with the last element. Output the converted array.

3. Enter Array `A` of 10 elements, To find product of positive elements and display it on the screen.

4. Enter Array `A` of 10 elements, To find product of negative elements and display it on the screen.

5. Enter array A of 10 elements, find the sum of positive elements and display it on the screen.
6. Enter array A of 10 elements, find the sum of negative elements and display it on the screen.
7. Enter array A from 10 elements, find amount elements, greater than 3 and less than 8 and display it on the screen.
8. Enter array A from 10 elements, find amount elements smaller modulo 5 and display it on the screen.
9. Enter array A of 10 elements, find the sum of elements greater than 2 and less than 5, and display it.
10. Enter array A from 10 elements, find amount elements, greater than 4 and less than 7 and display it on the screen.

5. Control questions

1. What is an array?
2. What is the index of an array element?
3. Within what limits do the indexes of array elements change?
4. What is the define directive for?
5. How do I type and output an array element?

Practical Lesson No 8

Multidimensional Arrays

1. Purpose of the work: Consolidation of knowledge about multidimensional arrays, compilation of programs with multidimensional arrays.

2. Basic information

An array can have not one, but more indices. The number of indices is called the dimension of the array, for example, an array with two indices is called a two-dimensional array. Such a two-dimensional array is, in particular, a matrix of a system n of linear algebraic equations with n unknowns. At the same time, the column of free terms of this system is a one-dimensional array.

For each of the indexes, the size of the array is set, for example, the description of a two-dimensional array of four rows and three columns looks like `float a[4][3]`. Such an array has $3 \times 4 = 12$ elements, the first index changes from 0 to 3, the second index from 0 to 2. Thus, the elements of the array are: `a[0][0]`, `a[0][1]`, `a[0][2]`, `a[1][0]`, ..., `a[3][2]`.

As a rule, when processing multidimensional arrays, nested loops are used, i.e. a loop by columns is located inside a loop by rows. In this way, array elements are entered and output, maximum elements in rows are searched, etc.

Here is an example of working with multidimensional arrays. In the example, a 2×3 matrix is introduced, the elements of the 2nd and 3rd columns are multiplied by 2, and the transformed matrix is displayed on the screen.

`mnog_mas.c` Multidimensional Arrays

```
#include<stdio.h>
```

```
#include<math.h>
```

```
main() {
```

```
int a[2][3],i,j;
```

```
for(i=0; i<2; i++)
```

```
{ for(j=0; j<3;
```

```
j++) {
```

```

printf("Enter a[%d][%d]=",i,j);
scanf("%d",&a[i][j]);} }
for(i=0; i<2; i++) for(j=1;
j<3; j++) a[i][j]*=2;
for(i=0; i<2; i++) { for(j=0;
j<3; j++)
printf("%6d",a[i][j]);
printf("\n");}
return 0;}

```

In this example, when entering, the loop by *i* is external, the loop by *j* is internal, nested in the loop by *i*. When entered, the loop parameter *i* changes from 0 to 2, the internal parameter *j* changes from 0 to 3. When multiplying, *i* changes from 0 to 2, and the parameter *j* from 1 to 3, because the first column does not need to be multiplied by 2.

3. Getting the job done

Type and compile the above program, correct the identified errors. Enter the elements of the array, make sure that it is processed correctly.

Compose a program using multidimensional arrays according to the variants of tasks, compile it, enter initial data, check the result.

4. Variants of tasks

1. Enter matrix A of 3x3 elements and vector B of 3 elements. Move the elements of the vector to the main diagonal of the matrix, and the elements of the main diagonal to the vector. Display the converted arrays on the screen.

2. Enter matrix A from 3x3 elements, find the smallest element and subtract the smallest element from the elements of all columns, except the first one. Output the transformed matrix.

3. Enter array B of 3x4 elements, find the products of the elements of each column and display them on the screen.

4. Enter matrix A from 3x3 elements, find the largest element and subtract the largest element from the elements of all rows, except for the third. Output the transformed matrix.

5. Enter an array B of 3x4 elements, find the sums of the elements of each row and display them on the screen.

6. Enter matrices A and B from 3x3 elements, find matrix C, the elements of which are equal to the sums of the corresponding elements of matrices A and B, and display it on the screen.

7. Enter matrices A and B from 2x3 elements, find matrix C, the elements of which are equal to the differences of the corresponding elements of matrices A and B, and display it on the screen.

8. Enter array A of 4 elements, place these elements on the main diagonal of matrix B 4x4, put the rest of the elements of matrix B equal to 1. Display matrix B on the screen.

9. Enter matrix A from 3x3 elements, find the smallest element and subtract the smallest element from the elements of all columns, except for the third. Output the transformed matrix.

10. Enter an array B of 3x3 elements, find the sums of the elements of each column and display them on the screen.

5. Control questions

1. What is a multidimensional array?
2. How many indexes can an MDX array member have?
3. Within what limits do the indexes of array elements change?
4. How to Enter Elements multidimensional array?
How much Need cycles?
5. Where are multidimensional arrays used?

Practical Lesson No 9

Symbolic Data

1. Purpose of the work: Consolidation of knowledge about symbolic data, compilation of programs using symbolic data.

2. Basic information

As you know, data in a computer are represented using combinations of binary digits: zero and one. How to represent symbols, for example, letters of the Russian or English alphabets?

To solve this problem, special tables called code tables have been developed. In such a table, it is determined which combination of ones and zeros is given a certain alphabet symbol.

In the United States, the ASCII (American Standard Code for Information Interchange) table was adopted as the code table. As the basic unit of representation of symbolic information, 1 byte, 8 binary bits was accepted. In this representation, $2^8 = 256$ different characters can be defined.

The ASCII table consists of two parts. The first part (characters with codes 0 – 127) contains numbers, Latin characters (uppercase and lowercase letters of the Latin alphabet), punctuation marks, control characters (with codes 0 – 31), for example, line breaks, and other characters.

The second part of the ASCII table (symbols with codes 128 – 255) contains pseudographic symbols (for graphing tables), some other symbols, for example, sum and integral symbols, letters with diacritics.

The first half of the ASCII code table is used as the international code table, the second half of the code table is used in different countries to represent the characters of the national alphabets, so the second half of the code table may be different in different countries.

In Russia, two encodings were used to represent Cyrillic characters (letters of the Russian alphabet): GOST encoding and alternative encoding. In most cases, an alternative encoding is used. Table 1 shows the international part of the ASCII code table.

Table 1. International part of the ASCII table.

Code	Symbol	Code	Symbol	Code	Symbol	Code	Symbol
32		56	8	80	P	104	h
33	!	57	9	81	Q	105	i
34	“	58	:	82	R	106	j
35	#	59	;	83	S	107	k
36	\$	60	<	84	T	108	l
37	%	61	=	85	U	109	m
38	&	62	>	86	V	110	n
39	„	63	?	87	W	111	o
40	(	64	@	88	X	112	p
41	)	65	A	89	Y	113	q
42	*	66	B	90	Z	114	r
43	+	67	C	91	[	115	s
44	,	68	D	92	\	116	t
45	-	69	E	93	]	117	u
46	.	70	F	94	^	118	v
47	/	71	G	95	_	119	w
48	0	72	H	96	`	120	x
49	1	73	I	97	a	121	y
50	2	74	J	98	b	122	z
51	3	75	K	99	c	123	{
52	4	76	L	100	d	124	

53	5	77	M	101	e	125	}
54	6	78	N	102	f	126	~
55	7	79	O	103	g	127	

Symbols with codes 128 – 255 in alternative encoding, including Cyrillic characters, are shown in Table 2.

Table 2. Alternate encoding symbols.

Code	Symbol	Code	Symbol	Code	Symbol	Code	Symbol
128	A	160	a	192	└	224	p
129	B	161	b	193	┐	225	c
130	In	162	in	194	┌	226	t
131	G	163	g	195	└┐	227	U
132	D	164	d	196	—	228	ph
133	E	165	e	197	├	229	x
134	F	166	f	198	└┐	230	c
135	Z	167	z	199	└┐	231	h
136	AND	168	and	200	└┐	232	sh
137	Y	169	th	201	└┐	233	sch
138	K	170	K	202	└┐	234	Kommersa nt
139	L	171	L	203	└┐	235	S
140	M	172	m	204	└┐	236	P
141	T	173	T	205	=	237	E
142	A	174	about	206	o	238	yu
143	P	175	P	207	└┐	239	I
144	P	176	▒	208	└┐	240	Ë
145	S	177	/	209	└┐	241	•
146	T	178	■	210	└┐	242	≥
147	U	179	˘	211	└┐	243	≤
148	F	180	└	212	└	244	∫
149	X	181	└	213	└	245	└
150	C	182	└	214	└	246	†
151	H	183	└	215	└	247	≈
152	Sh	184	└	216	└	248	˘
153	Sch	185	└	217	└	249	•
154	Kommersa nt	186	└	218	π	250	□
155	S	187	└	219	■	251	√
156	b	188	└	220	˘	252	№

157	E	189	Љ	221	Љ	253	▣
158	Yu	190	Њ	222	/	254	▪

159	I	191	Ѳ	223	□	255	
-----	---	-----	---	-----	---	-----	--

In the Windows operating system, there are no pseudographic symbols, because lines are drawn by other means. In this operating system, the Windows 1251 code table is adopted. Cyrillic characters have the codes 192 –255 (without the letters è), Ë has the code 168, è has the code 184:

Table 3. Cyrillic characters in Windows 1251 encoding.

Code	Symbol	Code	Symbol	Code	Symbol	Code	Symbol
168	Ë	207	P	224	a	241	c
184	•	208	P	225	b	242	t
192	A	209	S	226	in	243	U
193	B	210	T	227	g	244	ph
194	In	211	U	228	d	245	x
195	G	212	F	229	e	246	c
196	D	213	X	230	f	247	h
197	E	214	C	231	z	248	sh
198	F	215	H	232	and	249	sch
199	Z	216	Sh	233	th	250	Kommersa nt
200	AND	217	Sch	234	K	251	S
201	Y	218	Kommersa nt	235	L	252	P
202	K	219	S	236	m	253	E
203	L	220	b	237	T	254	yu
204	M	221	E	238	about	255	I
205	T	222	Yu	239	P		
206	A	223	I	240	p		

In an ASCII code table (and other code tables that have 1 byte per character), you can represent 256 different characters. This is enough for alphabetic alphabets,

but not enough for Asian languages

with hieroglyphic writing (the number of hieroglyphs is several thousand). Therefore, the Unicode encoding was introduced, in which each character is allocated not 1 byte, but 2 bytes, 16 binary bits. In the Unicode encoding, it is possible to represent $2^{16} = 65536$ different characters, which is enough for all languages, including those with hieroglyphic writing.

In C++, character data is of type `char` or `unsigned char` and takes up 1 byte of RAM. Encoding depends on the implementation of the compiler.

In the Borland C++ Builder programming environment, ASCII encoding is used in the console application mode, Russian letters are written in an alternative encoding (Table 2). In the windowed application mode, Windows 1251 encoding is used.

Character input in C++ Builder is done in Windows 1251 encoding, and output in a console application is in an alternate encoding. Therefore, there will be a mismatch between the input and output of character and string data in console application mode.

Latin characters (Latin letters) have the same codes in both the alternate encoding and Windows 1251 encoding, so Latin letters in console application mode are entered and displayed correctly. Russian letters, on the other hand, have different codes in these tables, so pseudographic symbols are displayed on the screen when there are Russian letters.

In Windows windowed application mode, both input and output are based on the Windows 1251 code table, so there is no mismatch between input and output data.

In the C language, a string is defined as an array of characters, and a character with the code 0 is placed at the end of the string. Such strings are called null-terminal strings, i.e. strings in which the attribute of the end of the string is a character with the code 0. This character is added by the compiler itself when the string is formed.

The following program discusses three strings, of which one is a constant that is specified in the program, and two that are entered strings. A constant string defined in the program requires transcoding from the Windows 1251 code table to an alternate encoding. For the strings that are entered, both input and output are performed in an alternate code table, and no transcoding is required.

lab10.cpp symbolic data

```
#include<iostream.h>
#include<stdio.h>
#include<conio.h>
char* cyr(char* S){
unsigned char* r=S;
while(*r){if (*r==168) *r=240;
else if(*r==184) *r=241; else
if (*r>=192) if(*r<=239) *r-=64;
else *r-=16; r++;}
return S;} //cyr
main(){
int i,k;
unsigned char j,s0[]="Constant string",s1[20],s2[20],s00[18]; cout<<cyr("s0
without conversion")<<endl;
cout<<s0<<endl;
strcpy(s00,s0);
cout<<cyr("s0 with transformation")<<endl;
cout<<cyr(s00)<<endl;
cout<<cyr("s0 codes without conversion")<<endl;
strcpy(s00,s0);
for(j=0; j<=strlen(s00); j++)
printf("%u ",s00[j]); printf("\n");
cout<<cyr("S0 codes with transformation")<<endl;
```

```

strcpy(s00,s0); cyr(s00);
for(j=0; j<=strlen(s0); j++)
printf("%0u ",s00[j]);
printf("\n");
cout<<cyr("Enter string")<<endl;
gets(s1);
cout<<cyr("Output entered string")<<endl;
cout<<s1<<endl;
cout<<cyr("Output codes of the entered string")<<endl; for(j=0;
j<=strlen(s1); j++)
printf("%0u ",s1[j]); printf("\n");
cout<<cyr("Enter string")<<endl;
gets(s2);
cout<<cyr("Output entered string")<<endl;
printf("%s\n",s2);
cout<<cyr("Output codes of the entered string")<<endl; for(j=0;
j<=strlen(s2); j++)
printf("%0u ",s2[j]);
printf("\n"); getch();
return 0;}

```

In the above program, the cyr (from Cyrillic) function is used to convert from the Windows 1251 table to an alternative encoding. The function parameter is a pointer to the string, after applying this function, the string is re-encoded.

From the comparison of Tables 2 and 3, the algorithm for recoding in the cyr function follows: the letters Ё and ё, which have codes 168 and 184 in the 1251 encoding, are replaced by codes 240 and 241, respectively, all the large letters of the Russian alphabet and the first half of the small letters (codes 192 – 239) are replaced by codes that are 64 smaller, the second half of the small letters (codes 240 – 255) are replaced by codes that are 16 smaller.

The string s0, defined in the program as a "Constant string", turns into a set of meaningless characters when directly output (without recoding). If the string s0 is copied to the string s00 using the standard strcpy function, the string s00 is re-encoded using the cyr function and s00 is displayed on the screen, the correct output will be obtained "Constant line".

Output the codes of the converted and unconverted string s0, make sure that the unconverted string has an encoding of 1251, and the converted string has an alternative encoding.

Now enter the lines s1 and s2 from the keyboard, for example, "My line1" and "My string2" using the gets function. This function enters the entire string, along with spaces. Output the strings s1 and s2 using the cout output operator and the .printf function. Next, output the character codes of these strings. Make sure that both input and output are in an alternative encoding, so there is no need to use the cyr function for conversion.

3. Getting the job done

Type and compile the above program, correct the detected errors. Pay main attention to finding out the type of code table with different line definitions. Enter several variants of strings and find out the resulting encoding.

Modify the above program with different constant and input strings. Describe the results obtained.

4. Control questions

1. How is character data represented in a computer?
2. What is a code table?
3. What types of code tables do you know?
4. Why is recoding necessary? In what cases is it necessary?
5. Principles of constructing the recoding function.

6. Why do I use the `cyr` function when displaying the "Output Entered String" message?
7. Why is the `cyr` function not used when displaying the entered string?
8. How do I output the character codes of the entered string?
9. Why is line `s0` copied to line `s00`?

Practical Lesson No 10

Character Strings

1. Purpose of the work: Consolidation of knowledge about strings, compilation of programs using strings.

2. Basic information

A string is a one-dimensional array of characters containing one character more than the required number of elements in a string. The fact is that in the C language a string can contain up to 32767 characters, the sign of the end of the string is a zero-code character included in the string.

A string as an array of characters (type `char name[n]`) ending with a null code character (zero-terminal string, ASCIIZ string, C-string) is inherited in C++ from C. In addition to C-strings, C++ has a string class, which will be discussed later.

Working with null-terminal strings is easier than working with strings of the string class, but it should be remembered that for null-terminal strings there is no out of range check (out of range message). Therefore, the programmer must take care not to go out of range.

The header file `string.h` in the C++ standard library contains functions for working with C-strings. At the same time, in the visual environments of Visual C++ and Borland C++ Builder, the header file `string.h` contains functions for working with the string class. To avoid confusion in these environments, the header file for working with C-strings is called `cstring.h`.

The header file `cstring.h` contains the following functions: `strcmp` compares two lines;
`strcpy` copies one string to another; `strlen` returns the length of the string;
`strstr` looks for a substring in a string.

In the example program below, you enter a string of no more than 80 characters (strlen = 81 including a null character) that contains words separated by spaces. You want to count the total number of words.

slova.cpp word count

```
#include<stdio.h>
```

```
#include<conio.h>
```

```
#include<cstring.h>
```

```
main(){ const len=81;
```

```
int i,j,k;
```

```
char stroka[len];
```

```
printf("Enter string\n");
```

```
gets(stroka);
```

```
k=1; word counter
```

```
j=strlen(stroka);
```

```
printf("%s\n",stroka);
```

```
for(i=0; i<j; i++) if( stroka[i]= "=" , &&
```

```
stroka[i]!=" " ,) k++ ; between apostrophes there is a space
```

```
printf("The number of the words %d\n", k);
```

```
getch();
```

```
return 0;}
```

Note that we used the gets function from the stdio.h header file to enter the string. This function enters the entire string, including whitespace characters (space, tabs), while the scanf function enters the string only up to the first whitespace. We make sure of this by replacing the program string gets(stroka); with the string scanf("%s",&stroka);.

A similar result will be obtained by applying the input cin stream and the >> input operation, the input will be before the first whitespace.

The program searches all characters to the end of the line. If the current character is a space, and the next character is not a space, then the number of words k increases by 1 (the current word has ended).

3. Getting the job done

Type and compile the above program, correct the detected errors. Enter a string with words separated by spaces, make sure that the program works correctly. Replace the string with the gets function with the string with the scanf function, make sure that in this case the input is made before the first whitespace character.

Develop and execute the program according to the task options.

4. Variants of tasks

1. Enter a string of words separated by spaces. Count the number of words that begin with the prefix entered from the keyboard.
2. Enter a string of words separated by spaces. Count the number of words that begin with a given character.
3. Enter a string of words separated by commas. Count the number of words.
4. Enter a string of words separated by spaces. Count the number of words that contain numbers.
5. Enter a string of words separated by spaces. Count the number of words containing the letters "a", "b", "c", "d".
6. Enter a string of words separated by spaces. Count the number of words containing 3 letters.
7. Enter a string of words separated by spaces. Count the number of words containing 2 or 3 letters.
8. Enter a string of words separated by spaces. Count the number of words containing the letters "d", "r", "t".
9. Enter a string of words separated by spaces. Count the number of words beginning with the letter "z".

10. Enter string of words, divided spaces. Count the number of words containing 4 letters.

5. Control questions

1. What is a null terminal string?
2. How do you describe a null-terminal string?
3. Why is it advisable to use the gets function instead of scanf when entering a string?
4. Which Bjrland C++ Builder IDE header file contains null-terminal string functions?
5. Which function determines whether a substring is contained in a string?
6. Which function compares strings?
7. Which function copies strings?

Language

1. Purpose of the work: Consolidation of knowledge about structures, compilation of programs using structures.

2. Basic information

Unlike arrays, which contain elements of the same type, structures can contain elements of the same or different types. As an example, we can give information about goods: the name of the product (text), the quantity of the product, its price. The name of the product belongs to the string data type, the quantity and price of the product - to the real type (float).

Since one structure can combine heterogeneous types of data, structures can be used to conveniently describe databases in a variety of subject areas. For example, the "Student" structure can contain the following information: No of the grade book, last name, first name, patronymic, course, group, scholarship, home address, grades by subjects.

The example below describes a tovar structure with the following fields:

naim[20] - an array of characters of 19 elements - name; kol, cena -
real numbers (quantity, price), t1, t2, t3 are variables of the structure
type Name, as string variable, entered In the %s
format, the quantity and price of the product are entered in the usual %f format for real
numbers.

To refer to a structure element, first write down the structure name, followed by a period, then the field name.

struct.c working with #include

<stdio.h>

```
struct tovar{char naim[20]; float kol,cena;} t1,t2,t3;
```

```
float r1,r2,r3,r;//the cost of goods 1, 2, 3 and the total cost of main()
```

$$\{$$

```
printf("Enter item name 1\n");
```

```

scanf("%s",&t1.naim);//structure name,period,field name-name
%s - format input of the character string printf
("Enter the quantity of goods 1, kg\n");
scanf("%f",&t1.kol);//structure name,dot,field name-quantity
printf("Enter item price 1, rub/kg\n");
scanf("%f",&t1.cena);//structure name,dot,field name-price
r1=t1.kol*t1.cena;
printf("The cost of the product %s is %8.2f rubles\n",t1.naim,r1);
/*The data for goods 2 and 3 are entered in the same way and the
total value r=r1+r2+r3 */ is calculated.
return 0;}

```

Structs, like arrays, can be worked with using pointers. A pointer to a struct is entered using a description:

```
structure_name *pointer_name;
```

For example, for the tovar structure, the description of the pointer is:

```
struct tovar{char naim[20]; float kol,cena;} t1, t2, *p;
```

Here, t1, t2 are structures of the tovar type, p is a pointer to a structure of this type. The pointer can be initialized with the address of a particular structure, for example:

```
p = &t1;
```

When working with structures using pointers, you enter the "arrow", a token consisting of a minus sign and a "greater" sign, i.e. "->". The arrow replaces the point when referring to the structure field and is the separator between the structure pointer name and the structure field name. For example, after the described initialization of the p pointer to refer to the naim field of the t1 structure, instead of t1.naim you should write p -> naim.

3. Getting the job done

Type and compile the above program, correct the identified errors. Enter the elements of the structure, make sure that the program works correctly.

Compose a program using structures according to the variants of tasks, compile it, enter initial data, check the result.

4. Variants of tasks

1. Enter 3 structures of the student type with the fields: last name, scholarship; and output the amount of student scholarships.
2. Enter 2 structures of the kniga type with fields: author, title, year of publication; and output the number of books published before 1998.
3. Enter 3 structures of the VUZ type with the fields: name of the university, number of students and display the total number of students.
4. Enter 2 structures of the computer type with fields: name, clock speed, cost and display the total cost of computers with a clock speed of at least 100 MHz.
5. Enter 3 structures of the school type with fields: city, school number, number of students admitted to universities; and display the total number of applicants.
6. Enter 2 sport structures with fields: sport, number of sections, number of athletes per section; and output the total number of athletes.
7. Enter 3 bus type structures with fields: bus brand, speed, capacity; and output the total capacity of buses with a speed of at least 80 km/h.
8. Enter 2 fabric structures with fields: factory name, year of foundation, number of employees; and output the total number of employees in the foundation factories up to 1990.
9. Enter 2 structures of the kafedra type with the fields: full name, position, year of birth; and display the number of employees with the year of birth before 1978.

10. Enter 3 structures of the computer type with the following fields: name, amount of RAM, amount of external memory and output the number of computers with more than 60 GB of external memory.

5. Control questions

1. What is structure?
2. What is the separator between the structure name and the fields?
3. How do I enter a structure?
4. Can an array of structures exist? Is it possible in your program to use an array of structs?
5. How do you determine a pointer to a structure?
6. What is the separator between the pointer name and the structure field name?

Practical Lesson No 12

Associations. Bitfields

1. Purpose of the work: Consolidation of knowledge about unions and bit fields, compilation of programs using unions and bit fields.

2. Basic information

To save space in RAM, C/C++ introduces a structured data type called union.

Example of a union entry:

```
union a {  
    int c;  
    char d;  
    double r;  
};
```

Next, variables of this type are declared:

```
union a t1, t2, t3;
```

For each of the instances of t1, t2, t3 of union a, there will be as much space in RAM as the largest field occupies, in this case, a field r of type double (8 bytes). To refer to a field, write t1.c = 5; t2.d = "R", etc., as for a structure. An analogue of a union in Pascal's language is entries with variants.

Here is an example of a program that uses conjunctions to decipher the format of a real number with double the accuracy.

```
lab11.cpp union  
#include<stdio.h>  
#include<conio.h>  
main(){  
    int i,j;  
    union {  
        unsigned char a[8];  
        double b;} t1,t2;
```

```

t1.b=1; t2.b=-1;
for(i=0; i<8; i++)
printf("%X ",t1.a[7-i]); printf("\n");
for(i=0; i<8; i++)
printf("%X ",t2.a[7-i]); printf("\n");
getch();
return 0; }

```

The program describes a union pattern containing an array of 8 unsigned bytes and a real number. You enter 2 variables of the union type t1 and t2. The real field of the union t1 is assigned the value 1, and the real field of the union t2 is assigned the value -1.

Next, both unions are output in hexadecimal byte-by-byte form, as two hexadecimal digits for each byte with a space between the bytes. Thus, the numbers 1 and -1 will have representations:

3F F0 00 00 00 00 00 00

BF F0 00 00 00 00 00 00.

Converting from a 16-digit representation to a binary representation is easy: each 16-digit is represented as 4 binary digits.

Bit fields represent the ability to access each individual bit (binary bit), which is typical for low-level languages. A two-byte machine word (16 binary bits) is described as a set of signed int or unsigned int bit fields. Example of a bit field description:

```

struct hk{ int a:5, b:3; unsigned c:4, d:4) r1,r2;

```

There are two signed fields a and b of five and three digits (the signed specifier is omitted) and two unsigned fields c and d of 4 bits each (the int specifier is omitted). After the pattern of bit fields, two variables of this type r1 and r2 are described.

By using bitfields, you can save memory if your program requires many variables of short length (a few bits). In addition, you can pack several small data.

3. Getting the job done

Type and compile the above program, correct the identified errors.

Compose a program using combinations according to the variants of tasks, compile it, enter initial data, check the result.

4. Variants of tasks

1. Compose a program to obtain a bitwise representation of integers 384 and -1187 using conjunctions.
2. Compose a program to obtain bitwise representations of integers 314 and -987 using conjunctions.
3. Compose a program to obtain bitwise representations of integers 137 and -1082 using conjunctions.
4. Compose a program to obtain bitwise representations of integers 312 and -874 using conjunctions.
5. Compose a program to obtain a bitwise representation of integers 104 and -176 using conjunctions.
6. Compose a program to obtain bitwise representations of integers 231 and -687 using conjunctions.
7. Draw up a program for obtaining integers 95 and -271 using conjunctions. bitwise Submissions
8. Make a program for obtaining integers 82 and -563 using conjunctions. bitwise Submissions
9. Draw up a program for obtaining integers 529 and -688 using conjunctions. bitwise Submissions
10. Compose a program to obtain a bitwise representation of integers 614 and -391 using conjunctions.

5. Control questions

1. What is unification?
2. What separator is placed between the name union and the name fields?
?
3. What is the difference between unification and structure?
4. What memory is allocated for merging?
5. What is a bitfield?
6. What are the possible types of bitfields?
7. When is it convenient to apply a bitfield?

Practical Lesson No 13

Functions in the C++ Language

1. The purpose of the work: consolidation of knowledge about functions, compilation of programs using functions, mastering the functions of data translation from one type to another.

2. Basic information

Quite often in a program it is necessary to repeat a certain sequence of operators in different parts of the program. In order to describe this sequence once, and to apply it repeatedly, subprograms are used in programming languages. In the Pascal language there are 2 types of subroutines: procedures and functions, in the C language - one type of subroutines - functions. If the meaning requires a procedure, a function that does not return a value (of the void type) is used.

A C program must contain a main function that serves as an entry point into the program, and other functions are optional.

The form of recording a function in the program is as follows:

<type> <name> (<list of formal parameters>) {<function body>}

Here <type> is the type of the value returned by the function, if the type is not specified, the default is the int type, <name> is the name of the function, followed by a list of formal parameters in parentheses, even if the list of parameters is empty, parentheses are still placed, the list of parameters in parentheses is followed by the body of the function in curly brackets.

For each formal parameter, a type is specified, and formal parameters are separated by commas. There must be at least one operator in the function body

return <expression>; or

return;

where the expression defines the value returned by the function, if the function is of type void (does not return a value), then the second form of return from the function is used, without specifying the <expression>.

The description of the function must precede its use. If for some reason this cannot be done, you should place its prototype before using the function - the function header containing the type of the value returned by the function, the name of the function, the types of formal parameters in parentheses, after the parentheses in the prototype, a semicolon is placed instead of the function body. If a prototype is specified, the function description containing the function body can be placed after the function is used.

In the Borland C++ Builder environment, there are a number of header files that contain standard functions that do not need to be described in the program. However, it is necessary to use the `#include<file name>` directive, which specifies the name of the header file containing this function. For example, I/O functions are contained in the `stdio.h` file, string functions are contained in the `cstring.h` file, and mathematical functions are contained in the `math.h` file.

Functions in C cannot be nested inside each other, each function is described separately, but in the body of one function there can be a reference to another function.

The standard functions contained in the header files are not described in the program. The most commonly used mathematical functions are those contained in the `math.h` header file. As a rule, the arguments and values returned by the functions are of the double type, other cases are especially noted. In the program, variables can also have another real type, for example, float, when a standard function is accessed, an implicit conversion of the float -> double and double types will be performed

-> float. It is preferable to use the double type, because the calculations are carried out with double precision and rounding errors are less affected.

For the input and output of variables of type double in the printf functions and scanf uses the %lf format.

Here is an example of a program for compiling a table of the function

$$y = x^2 - 3x + 4$$

when x is changed from 0 to 1 in 0.1 increments.

lab14_1.cpp functions

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
double y(double x){
return x*x-3*x+4;}
main(){int i; double x;
printf("=====\n");
printf("    x    |    y\n");
printf("=====\n");
for(i=0; i<=10;
i++){x=(double)0.1*i; printf("%7.1lf
                        | %9.4lf\n",x,y(x));}
printf("=====\n");
getch();
return 0;}
```

The header file inclusion directives are followed by a description of the function. Formal parameters are used in the description, in this case a single real parameter x. In the main function, multiple calls to the function y (calls to the function) are made. The actual parameters of the function are used in the access.

Functions with several parameters are programmed in a similar way. In this case, each of the formal parameters is described in the function description with its type. When referring to a function, the number and types of actual parameters must coincide with the number and types of formal parameters.

Note: It is possible to describe functions with default parameters. Default parameters are described last. In this case, the number of actual parameters may be less than the number of formal parameters, and some or all of the default parameters may be omitted.

Here is an example of a program for compiling a table of functions of two variables

$$z = 2x^2 + 5y$$

for a change in x from 0.5 to 1.5 in 0.5 increments and a change in y from 1 to 2 in increments of 0.5:

lab14_2.cpp function of two variables

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
double z(double x, double y){
return 2*x*x+5*y;}
main(){int i,j; double x,y;
printf("=====\n");
printf("  x |   y |   z\n");
printf("=====\n");
for(i=5; i<=15; i+=5){x=(double)0.1*i; for(j=10;
j<=20; j+=5){y=(double)0.1*j; printf("%5.11f |%5.11f
| %9.4lf\n",x,y,z(x,y));}
printf("=====\n");
}
getch();
return 0;}
```

In Result	Jobs	Programs	will be Withdrawn	Function	two
variables. In	this and	previous	programs	We Used	

Simplified Graphing Tables c assistance signs Equality
and vertical dashes.

3. Getting the job done

Type and compile the above programs, correct the identified errors.

Compose program c Application Functions one and two
variables, check the result.

Two tasks are required in the work: task 1 – calculating the value of the function $y=f(x)$ from the specified initial values of the argument x_n , the final values of the argument x_k , the step of the value of the argument dx ;

Task 2 – calculation of the function $z= g(x,y)$ of two variables with the initial values of x_n and y_n , the final values of x_k and y_k , the steps dx and dy . Functions, initial, final values and steps are given in the variants of tasks.

4. Variants of tasks.

Number variant	Task 1			
	$y=f(x)$	x_n	x_k	dx
1	$\sqrt{x}+3$	0	1	0,2
2	$x^2 + 1$	0	2	0,2
3	$2x + 7 - x^3$	1	2	0,1
4	$7x + 8 - x^2$	0	1	0,1
5	$(x+3)^2 - 5$	1	2	0,25
6	$x^2 + 4 - x$	0	2	0,2
7	$x/(x + 1)$	1	3	0,2
8	$x/(3+x)$	0	2	0,2
9	$1,5 \cdot x^2$	0	1	0,1
10	$3/(x+2)$	1	2	0,1

Number variant	Task 2						
	$z=g(x,y)$	x_n	x_k	dx	y_n	y_k	dy
1	$3x-y^2$	1	2	0,5	0	1,5	0,5
2	$4x+3y^2$	0	1,5	0,5	1	2	0,5
3	$2x^2-3y$	1	2	0,5	0	1,5	0,5
4	$5x-y^2$	0	2	0,5	1	2	0,5
5	$2x+3y$	1	2	0,5	0	1	0,5
6	$2x+y$	0	1	0,5	1	2	0,5
7	$2x^2-y$	0,5	1,5	0,5	1	3	1
8	$5x+y^2$	0	2	1	1	2	0,5
9	$3x/(y+2)$	0	1	0,5	1	3	1
10	$(5x+4)/(y+1)$	1	2	0,5	0	1	0,5

5. Control questions

1. What is a subroutine? Types of subroutines in C++.
2. What are the formal parameters of a function?
3. How is the function header written? Function body?
4. Actual parameters of the function.
5. What is a feature prototype?
6. Yes, you can Lee Contact K functions,
without preliminary Its description?
7. Do I need to describe standard features?
8. How do I write a representative function in a program?

Recursion. Recursive Functions

1. The purpose of the work: consolidation of knowledge about recursive functions, compilation of programs using recursive functions.

2. Basic information. It is possible for a function to refer to itself. Such an inversion is called recursion, and functions are called recursive. A classic example of recursion is the calculation of an integer factorial.

The factorial of an integer n is the product of the numbers from 1 to n :

$$n! = 1 * 2 * 3 \dots * n$$

From the expression of the factorial, it is possible to derive the recurrence relation $n! = n * (n - 1) !$

This relation serves as the basis for the recursive function of finding the factorial:

lab15.cpp factorial

```
#include<stdio.h>
```

```
#include<conio.h>
```

```
int fact(int x){
```

```
if (x==0) return 1;
```

```
else return x*fact(x-1);}
```

```
main(){int n;
```

```
printf("Enter n\n");
```

```
scanf("%d",&n);
```

```
printf("n=%d n!=%d\n",n,fact(n));
```

```
getch();
```

```
return 0;}
```

In the main program, an integer n is entered, then the fact-finding function is accessed. In this function, there is a non-recursive output if $(x==0)$ return 1; i.e., if $n = 0$, the factorial is 1, otherwise (i.e., at $n \neq 0$) the same fact-function is accessed, but with an argument less than 1.

If in this case the argument is not equal to zero, the same function is again accessed, but with an argument that is 1 less than that, and so on.

The mechanism of operation of a recursive function, for example, a factorial calculation function, is as follows. When such a function is accessed, as well as any other function, the parameters of this function and the return address from the function, i.e. the address of the operator following the function call, are put on the stack.

If the function is not recursive, no more calls are made to it, and when the end of the function is reached, the return or return operator
<expression> the main program is accessed at the return address stored on the stack.

If the function is recursive, then its body contains an appeal to itself. Thus, for a recursive function, new parameters and return addresses will be entered into the stack from the body of the same function with each new access. This will continue until a non-recursive exit is encountered.

In the case of factorial calculation from a recursive function, the parameter n is put on the stack first, then $n-1$, and so on, until the parameter becomes zero and a non-recursive output occurs. In this case, all the parameters stored in the stack will be multiplied by each other and the correct result will be obtained.

If there is no non-recursive output in a recursive function, the function will be invoked indefinitely, and all new parameters will be added to the stack until a stack overflow occurs. In this case, the compiler issues a specific "stack overflow" message. If such a message is issued, you should look for the absence of a non-recursive output in one of the recursive functions.

It should not be thought that a factorial can be calculated only with the help of a recursive function. It can be computed with equal success using a loop with a parameter in which the multiplier increases by one for each step. In general, for any recursive function, there is a method for calculating it without the help of recursion.

Why is recursion used in this case? The point lies in the exceptional simplicity of recursive algorithms, as can be seen in the example of factorial. In some cases, it can be very difficult to formulate an algorithm without the help of recursion. Therefore, the use of recursive algorithms is justified.

When calculating the factorial, the depth of recursion is equal to one, i.e. the function refers to itself with a parameter that is less than one. There are cases of a greater depth of recursion. For example, when calculating Fibonacci numbers, the depth of recursion is two, the function refers to itself with parameters less by one and two. Let's consider the calculation of Fibonacci numbers in more detail.

In the XIII century, Leonardo Fibonacci (son of Bonacci) from Pisa posed and solved the problem of rabbit reproduction. There is a pair of rabbits, which in a month brings offspring. In another month, both the old and new pair of rabbits bring offspring. The number of pairs of rabbits after n months is the n th Fibonacci number F_n .

It is easy to see that Fibonacci numbers are described by a recurrent formula:

$$F_1 = 1, F_2 = 1, F_n = F_{n-1} + F_{n-2}.$$

For example, $F_3 = F_2 + F_1 = 1 + 1 = 2$, $F_4 = F_3 + F_2 = 2 + 1 = 3$, etc.

The program for calculating Fibonacci numbers looks like this:

lab15_2.cpp Fibonacci numbers

```
#include<stdio.h>
```

```
#include<conio.h>
```

```
int Fib(int x){if(x==1|| x==2)
```

```

return 1; else return
Fib(x-1)+Fib(x-2);}
main(){int n;
printf("Enter n\n");
scanf("%d",&n);
printf("n=%d F(n)=%d\n",n, Fib(n));
getch();
return 0;}

```

The above program works correctly, but is it optimal? You can see that this program spends a lot of time on repeated calculations. For example, when calculating F_5 , F_4 and F_3 are calculated, when calculating F_4 , F_3 and F_2 are calculated, i.e. F_3 is recalculated, and so on.

Thus, it is impractical to use a recursive function for calculating Fibonacci numbers. A function with cyclic calculations is more optimal. The reason for this is that the depth of recursion in the recurrence formula for Fibonacci numbers is equal to two, i.e. greater than one.

In addition to direct recursion, as in the calculation of factorial, there is also indirect recursion. This case is observed if there are at least two functions in the program that refer to each other. In this case, you should use the prototype of one of the related functions.

When using a prototype, one of the related functions is described after it has been used. This is made possible by a prototype that specifies the type of value that the function returns and the types of all parameters. (The sum of parameter types and function types is called a function signature.)

3. Getting the job done

Type and compile the above programs, correct the identified errors.

Compose a program using a recursive function according to the task options.

4. Variants of tasks

1. The sequence of real numbers is given by the rules: $R_1 = 1,2$; $R_2 = 0,9$; $R_n = 0,9 \cdot R_{n-1} + 1,1 \cdot R_{n-2}$.

Find the fifth term of the sequence.

2. The sequence of real numbers is given by the rules: $R_1 = 1.25$; $R_2 = 0,8$; $R_n = 0,95 \cdot R_{n-1} + 1,0 \cdot R_{n-2}$.

Find the fourth term of the sequence.

3. The sequence of real numbers is given by the rules: $R_1 = 1,1$; $R_2 = 0,9$; $R_n = 0,86 \cdot R_{n-1} + 1,15 \cdot R_{n-2}$.

Find the sixth term of the sequence.

4. The sequence of real numbers is given by the rules: $R_1 = 1.05$; $R_2 = 0,9$; $R_n = 0,95 \cdot R_{n-1} + 1,05 \cdot R_{n-2}$.

Find the seventh term of the sequence.

5. The sequence of real numbers is given by the rules: $R_1 = 1.0$; $R_2 = 0,82$; $R_n = 0,9 \cdot R_{n-1} + 1,15 \cdot R_{n-2}$.

Find the fourth term of the sequence.

6. The sequence of real numbers is given by the rules: $R_1 = 1.12$; $R_2 = 0,89$; $R_n = 0,91 \cdot R_{n-1} + 1,05 \cdot R_{n-2}$.

Find the eighth term of the sequence.

7. The sequence of real numbers is given by the rules: $R_1 = 1.22$; $R_2 = 0,93$; $R_n = 0,94 \cdot R_{n-1} + 1,16 \cdot R_{n-2}$.

Find the fifth term of the sequence.

8. The sequence of real numbers is given by the rules: $R_1 = 1,3$; $R_2 = 0,91$; $R_n = 0,89 \cdot R_{n-1} + 1,17 \cdot R_{n-2}$.

Find the ninth term of the sequence.

9. The sequence of real numbers is specified by the rules:

$$R_1 = 1,18; R_2 = 0,83; R_n = 0,9 \cdot R_{n-1} + 1,08 \cdot R_{n-2} .$$

Find the sixth term of the sequence.

10. The sequence of real numbers is given by the rules: $R_1 = 1.08; R_2 = 0,92; R_n = 0,84 \cdot R_{n-1} + 1,19 \cdot R_{n-2} .$

Find the ninth term of the sequence.

5. Control questions

1. What is recursion? Is it a recursive function?
2. What is a non-recursive output?
3. K What Can Lead Absence non-recursive Exit
in a recursive function?
4. Is it possible to compute factorial without recursion?
5. What is a feature prototype?
6. What is a function signature?
7. Is it always advisable to use recursive functions?

List of recommended sources

List of basic literature:

1. Tyulpinova N. V. Algoritmizatsiya i programmirovaniye Elektronnyy resurs: Uchebnoe posobie [Algorithmization and programming Elektronnyy resurs: Uchebnoe posobie]. - Saratov : Vuzovskoe obrazovanie, 2019. - 200 p. - The book is in the premium version of EBS IPR BOOKS. - ISBN 978-5-4487-0470-3, unlimited copies
 2. Bederdinova O. I Osnovy algoritizatsii i strukturnogo programmirovaniye Elektronnyy resurs [Fundamentals of algorithmization and structural programming] / Bederdinova O. I. : textbook. - Arkhangelsk: NArFU, 2017. - 88 p. - ISBN 978-5-261-01227-6, unlimited copies
 3. Zabihullin, F. Z. Strukturnoe programmirovaniye na S++ Elektronnyy resurs [Structural programming in C++] / Zabihullin F. Z. : textbook. - Ufa : BSPU named after M. Akmulla, 2019. - 45 p. - ISBN 978-5-907176-11-9, unlimited copies
 4. Konova, E. A. Algoritmy i programmy [Algorithms and programs]. Language C++ Elektronnyy resurs / Konova E. A., Pollak G. A. : textbook for higher vocational education. - 5th ed., ster. - St. Petersburg: Lan, 2020. - 384 p. - APPROVED BY the UMO on education in the field of applied informatics as a textbook for students studying in the direction of "Applied Informatics". - ISBN 978-5-8114-5431-0, unlimited copies
- Beleva, L.F. Programming in the C++ language Elektronnyy resurs: uchebnoe posobie [Elektronnyy resurs: uchebnoe posobie]. - Saratov : IPR Media, 2018. - 81 p. - The book is in the basic version of the EBS IPRbooks. - ISBN 978-5-4486-0253-5, unlimited copies

List of additional literature:

1. Belotserkovskaya I. E. Algoritmizatsiya [Algorithmization]. Introduction to the C++ programming language / I.E. Belotserkovskaya; N.V. Galina; L.Yu. - 2nd ed., ispr. - Moscow: National Open University "INTUIT", 2016. - 197 p., unlimited copies
2. Sedgwick R. Algoritmy na S++ [Algorithms in C++]. - 2nd ed., ispr. - Moscow: National Open University "INTUIT", 2016. - 1773 p., unlimited copies
3. Lubasheva T.V., Zhelezko B.A. Fundamentals of Algorithmization and Programming: Textbook. - Minsk : RIPO, 2016. - 378 p.: ill. - <http://biblioclub.ru/>. - Bibliogr. in the book - ISBN 978-985-503-625-9, unlimited copies
4. Zotkin, S. P. Programming in the language of the high level of S/S++ Elektronnyy resurs / Zotkin S. P. : lecture notes. - 3rd ed. - Moscow: MISI – MGSU, 2018. - 140 p. - ISBN 978-5-7264-1810-0, copies unlimited
5. Kashirskaya, E. N. Procedural'noe programmirovaniye: Praktikum Elektronnyy resurs [Procedural programming: Practicum Elektronnyy resurs] / Kashirskaya E. N. - Moscow: RTU MIREA, 2020. - 75 p., unlimited copies

6. Bazhenova I.Y. Introduction to Programming Elektronnyi resurs : uchebnoe posobie [Introduction to Programming Elektronnyi resurs: uchebnoye posobie]. - Introduction to Programming, 2020-07-28. - Moscow, Saratov: Internet University of Information Technologies (INTUIT), University Education, 2017. - 327 p. - The book is in the basic version of the EBS IPRbooks. - ISBN 978-5-4487- 0073-6, unlimited copies

MINISTRY OF SCIENCE AND HIGHER EDUCATION OF THE RUSSIAN
FEDERATION
FEDERAL STATE AUTONOMOUS EDUCATIONAL INSTITUTION OF HIGHER
EDUCATION
"NORTH CAUCASUS FEDERAL UNIVERSITY"

SUGGESTIONS FOR TEACHING

on the organization and conduct of independent work
in the discipline "Advanced Language Programming C" for students of the direction of
training

11.03.02 Infocommunication Technologies and Communication Systems
"Intelligent Data Processing in Infocommunication Systems and Networks"

GENERAL CHARACTERISTICS OF THE DISCIPLINE

Purpose and objectives of the discipline

The goal of Disciplines "Advanced Language Programming C" is the formation of a future bachelor's degree from a set of competencies. The study of the discipline ensures the implementation of the requirements of the Federal State Educational Standard of Higher Education in the direction of 11.03.02 Infocommunication Technologies and Communication Systems

Objectives of the discipline: to study the basics of algorithmization and programming using the C++ programming language as an example.

The place of the discipline in the structure of the educational program

The discipline belongs to the compulsory part. Its development takes place in the 2nd semester.

List of planned learning outcomes in the discipline (module) correlated with the planned results of the development of the educational program

Code, formulation of competence	Indicator code, wording	Planned learning outcomes in the discipline (module), characterizing the stages of formation Competencies, Indicators
OPK-3. Able to apply methods of search, storage, processing, analysis and presentation in the required format of information from various sources and databases, observing the with These are the main Information security requirements	ID-3 OPK-3 solves data processing problems using computer technology.	solves the problem of data processing using computer technology, using the C++ language.
OPK-4. Capable understand Principles of work modern information technologies and use them to solve the problems of professional activities	ID-4 OPK-4 uses the capabilities of the computational Techniques and software for solving problems of control and algorithmization of information processing processes.	uses the capabilities of computing and software to create applications, uses the C++ language to solve information processing problems
OPK-5. Able to develop algorithms and Computer programs suitable for practical use	ID-2 OPK-5 reads the codes of software products written in the mastered programming languages and enters Required changes	reads codes, understands and makes required changes to the program code
	ID-3 OPK-5 develops original algorithms and computer	develops original Applications suitable for

Code, formulation of competence	Indicator code, wording	Planned learning outcomes in the discipline (module), characterizing the stages of competence formation, Indicators
	Programs suitable for practical application	practical application

2. CHARACTERISTICS OF INDEPENDENT WORK

All types of independent work in the discipline "Advanced Programming Language C/Advanced Language Programming C" are given in the table "Technological map of independent work". The basis of students' independent work in the discipline is the note-taking of lectures and recommended sources, preparation for computer testing of knowledge, preparation and preparation for the defense of reports on laboratory work, the performance of multi-level individual tasks in the course of laboratory work. For each type of independent work, certain forms of reporting are provided, which students present to the teacher during laboratory classes or during the hours of individual consultations.

Table 1 - Technological map of independent work

Code of the evaluated competence, indicator(s)	The stage of competence formation and (No Threads) (in Compliance of AI with the work program disciplines)	Means and Assessment Technologies	View control, certification (current/intermediate)	Type of control (oral, written or technical means)	Name of the valuation tool
ID-3, OPK-3, ID-4, OPK-4, ID-2, OPK-5, ID-3, OPK-5	Topic 1-16	Questions for Interview on practical work	Current	Oral	Questions for Interview on practical work
ID-3, OPK-3, ID-4, OPK-4, ID-2, OPK-5, ID-3, OPK-5	Topic 1-16	Questions for IYA Interview	Current	Oral	Questions for IYA Interview

For the successful development of the discipline, it is necessary to perform the types of independent work indicated in the table, using the recommended sources of information.

1. List of basic literature:

1. Tyulpinova N. V. Algoritmizatsiya i programmirovaniye Elektronnyy resurs: Uchebnoe posobie [Algorithmization and programming Elektronnyy resurs: Uchebnoe posobie]. - Saratov : Vuzovskoe obrazovanie, 2019. - 200 p. - The book is in the premium version of EBS IPR BOOKS. - ISBN 978-5-4487-0470-3, unlimited copies
2. Bederdinova O. I Osnovy algoritizatsii i strukturnogo programmirovaniye Elektronnyy resurs [Fundamentals of algorithmization and structural programming] / Bederdinova O. I. : textbook. - Arkhangelsk: NArFU, 2017. - 88 p. - ISBN 978-5-261-01227-6, unlimited copies
3. Zabihullin, F. Z. Strukturnoe programmirovaniye na S++ Elektronnyy resurs [Structural programming in C++] / Zabihullin F. Z. : textbook. - Ufa : BSPU named after M. Akmulla, 2019. - 45 p. - ISBN 978-5-907176-11-9, unlimited copies
4. Konova, E. A. Algoritmy i programmy [Algorithms and programs]. Language C++ Elektronnyy resurs / Konova E. A., Pollak G. A. : textbook for higher vocational education. - 5th ed., ster. - St. Petersburg: Lan, 2020. - 384 p. - APPROVED BY the UMO on education in the field of applied informatics as a textbook for students studying in the direction of "Applied Informatics". - ISBN 978-5-8114-5431-0, unlimited copies
- Beleva, L.F. Programming in the C++ language Elektronnyy resurs: uchebnoe posobie [Elektronnyy resurs: uchebnoe posobie]. - Saratov : IPR Media, 2018. - 81 p. - The book is in the basic version of the EBS IPRbooks. - ISBN 978-5-4486-0253-5, unlimited copies

2. List of additional literature:

1. Belotserkovskaya I. E. Algoritmizatsiya [Algorithmization]. Introduction to the C++ programming language / I.E. Belotserkovskaya; N.V. Galina; L.Yu. - 2nd ed., ispr. - Moscow: National Open University "INTUIT", 2016. - 197 p., unlimited copies
2. Sedgwick R. Algoritmy na S++ [Algorithms in C++]. - 2nd ed., ispr. - Moscow: National Open University "INTUIT", 2016. - 1773 p., Unlimited Instances
3. Lubasheva T.V., Zhelezko B.A. Fundamentals of Algorithmization and Programming: Textbook. - Minsk : RIPO, 2016. - 378 p.: ill. - <http://biblioclub.ru/>. - Bibliogr. in the book - ISBN 978-985-503-625-9, unlimited copies
4. Zotkin, S. P. Programming in the language of the high level of S/S++ Elektronnyy resurs / Zotkin S. P. : synopsis lektiv. - 3rd ed. - Moscow: MISI – MGSU, 2018. - 140 p. - ISBN 978-5-7264-1810-0, copies unlimited

5. Kashirskaya, E. N. *Procesednoe programmirovaniye: Praktikum* [Procedural programming: Practicum]
Electronic resource / Kashirskaya E. N. - Moscow : RTU MIREA, 2020. - 75 p., unlimited copies
6. Bazhenova I.Y. *Introduction to Programming Elektronnyi resurs : uchebnoye posobie* [Introduction to Programming Elektronnyi resurs: uchebnoye posobie]. - Introduction to Programming, 2020-07-28. - Moscow, Saratov: Internet University of Information Technologies (INTUIT), University Education, 2017. - 327 p. - The book is in the basic version of the EBS IPRbooks. - ISBN 978-5-4487- 0073-6, unlimited copies

3. METHODICAL RECOMMENDATIONS Software STUDY OF THEORETICAL MATERIAL

When studying the topics of lectures and the given literature on their own, students are recommended:

- on the day of the lesson, work out the educational material studied in the lesson according to the notes. At the same time, it is necessary to highlight the questions that should be paid more attention to in the further study of the current topic. After working through the educational material, it is necessary to answer the recommended control questions;
- In order to qualitatively consolidate the studied material in the future, it is necessary to work out the educational material in more detail using the recommended literature. Students are recommended to identify issues that require more detailed study with the help of the teacher;
- On the eve of the lecture, the student is recommended to consolidate the previously studied theoretical material, prepare questions that require clarification from the teacher.

Recommendations for the organization of work with literature

To master, consolidate and systematize knowledge, students are recommended to work independently with the recommended literature and lecture notes. When working independently with the recommended literature, students are recommended to take notes of the educational material set forth in the recommended sources.

Taking notes of sources is carried out during a detailed study of the topic, while students are recommended:

- supplement the lecture notes by expanding it on the most important issues considered in the lesson.

Control of this type of independent work is carried out in training sessions by checking the lecture notes and interviews by the teacher.

When taking notes of sources, students should proceed from the rational volume of the material being summarized. The synopsis should be concise and understandable when used after studying the current topic.

For the successful development of the discipline, it is necessary to independently study the presented topics in detail according to the recommended sources of information.

Work on taking notes of sources is evaluated satisfactorily if:

- the synopsis has a sufficient amount of detail on the topic under study, providing a given level of mastery of the theoretical material of the discipline;
- The synopsis is neatly formatted, the studied material is presented sequentially in accordance with the order of study of the discipline, contains generalizing conclusions on each topic.

4. METHODOICAL DIRECTIONS Software PERFORMANCE OF PRACTICAL WORK

When performing a workshop on the discipline (practical work) to achieve a basic level of competence formation, the student must be able to apply basic knowledge in the sections of the discipline, have the skills to independently solve educational problems of the basic level.

When performing a workshop on the discipline (practical work) to achieve an increased level of competence formation, the student must be able to apply increased knowledge in the sections of the discipline, possess the skills of independent solution of educational tasks of an advanced level.

Methodical instructions for preparing for laboratory work

Preparation for laboratory work is recommended to be carried out in the following sequence:

- to understand the topic and purpose of the upcoming laboratory work;
- study theoretical material in accordance with the topic of laboratory work (it is recommended to use recommended literature, lecture notes, textbook (practical work));
- get acquainted with the equipment and materials used in laboratory work (when using specialized equipment or software, it is necessary to study the procedure and rules for its use).

Methodical instructions for performing laboratory work

When performing laboratory work, students must strictly comply with the established safety requirements.

When performing practical work, students are recommended:

- to understand the purpose of the tasks performed and ways to solve them;
- to perform the tasks specified in the practical work in the sequence in which they are indicated in the practical workshop;
- use the help of a teacher when performing a practical task and studying theoretical material;

- draw up a report on practical work;
- answer control questions.

Methodical instructions for preparing for the defense of practical work

When preparing for the defense of laboratory work, students are recommended:

- prepare a report on practical work;
- prepare a justification for the conclusions made;
- consolidate knowledge of theoretical material on the topic of practical work (it is recommended to use control questions);
- know the procedure for making calculations (conducting research);
- to be able to show and explain the order of research during the use of specialized hardware or software.

The defense of practical work is carried out by interviewing the student with the teacher. During the interview, the student submits a report on practical work for verification. An interview with a student applying for an "excellent" grade is conducted on questions and practical tasks of an advanced level of study. An interview with a student who does not apply for an "excellent" grade is conducted on questions and practical tasks of the basic level of study.

5. QUESTIONS FOR INTERVIEWS FOR INDEPENDENT STUDY OF LITERATURE

Interview Questions

Questions (task, task) to check the level of learning

1. What are UML activity diagrams for?
2. What is the state of action and the state of activity?
3. What notations are there to indicate transitions and branches in activity diagrams?
4. Which algorithm is a branching structure algorithm?
5. What is the difference between a branching algorithm and a linear one?
6. What is a conditional statement? What are its forms?
7. What is a compound operator? What is the format for writing it?
8. What comparison operators are used in C?
9. What is called a simple condition? Give examples.
10. What is a composite condition? Give examples.
11. What Boolean operators are allowed when composing complex conditions?
12. Maybe Lee Cinematographer Branching
 Contain Inside yourself Other branches?
13. What is multiple choice?

14. When is the switch used?
15. Why is the break operator used in the switch?
16. Why Reserved in switch The word default? Used
17. Which UML diagrams show how a switch works?
18. Which algorithm is a cyclic structure algorithm?
19. Types of cycles in the C language.
20. What are the main parameters of the cycle?
21. What are increment and decrement operators, and in what forms are they used?
22. How to write the condition for the continuation of the loop in loops like while and do ... while?
23. The basic form of the cycle do ... while.
24. What are the three sections written in parentheses for a for loop operator? What is the separator between sections?
25. What are nested loops? Examples:
26. How does an endless cycle form and how to get out of it?
27. A scheme for casting a while loop to its equivalent for loop.
28. Name the operators that can change the natural order of calculations.
29. What is the break statement for?
30. Where is the continue operator used and what is it used for?
31. What is the return statement used for?
32. What type should an expression have?
33. How are one-dimensional arrays declared in C++?
34. What must to be dimension with Description static array in C++?
35. What is the range of array index changes in C++?
36. How is an array initialized in C++?
37. What is an array ID?
38. What is the \t control sequence used for?
39. How does a two-dimensional array appear in C++?
40. Where and how is a two-dimensional array stored?
41. Within what limits do indices of a two-dimensional array change?
42. What are some ways to describe a two-dimensional array?
43. What are the actions for each row when calculating the number of positive items?
44. In what location Programs should be Write down initialization operators for the quantities accumulated in the loop?
45. How can two-dimensional arrays be created in a dynamic area of memory?
46. Ways to create a dynamic array.
47. How do you access dynamic array elements?
48. What presents yourselfFunction in C++? What Have to for her

use?

49. Give an example of a function header.
50. What does the definition of a function include?
51. What is the difference between a function and other software objects?
52. How is a function called?
53. Describe Existing Methods Solutions problems of obtaining a flag from a subroutine of its abnormal termination.
54. What is the mechanism for passing parameters to a function?
55. Input data transfer methods.
56. What presents yourself Remedy C++, called default parameter values?
57. How does the function name be passed to a function?
58. How image is happening Transfer one-dimensional arrays into a function?
59. How do I pass strings to a function?
60. How image is happening Transfer two-dimensional arrays into a function?
61. How is the transfer of structures to a function carried out?
62. What Function is called recursive? Benefits and the disadvantages of recursion.
63. What is the point of using templates?
64. How do I write down a template parameter?
65. List the main properties of parameterized classes?
66. What is an association?
67. What is inheritance?
68. What is aggregation?
69. What is Addition?
70. Name and Describe Three Main principle in the use of classes.

1. Criteria for assessing competencies (in accordance with the results of mastering the discipline)

The grade "**excellent**" is given to the student if the student has given a complete, detailed answer to the questions posed, while the totality of conscious knowledge in the discipline is shown, the main provisions of the questions are demonstratively disclosed; the answer traces a clear structure, a logical sequence that reflects the essence of the concepts disclosed. Knowledge is demonstrated against the background of understanding it in the system of the discipline. The answer is presented in literary language using technical terminology. Shortcomings in the definition of concepts may be made, corrected by the student independently in the process of answering, demonstrated a high ability to apply the acquired knowledge in practice through solving a specific problem, freely coped with the task without difficulty, showing the possession of versatile techniques and skills for its implementation, did not make mistakes and inaccuracies.

The answer is clearly structured, logical, presented in literary language using technical terminology. 2-3 inaccuracies or minor mistakes may be corrected by the student with the help of the teacher, demonstrated the ability to apply the knowledge gained in practice through solving a specific problem, coped with the task, showing the possession of the necessary techniques and skills for its implementation, while making no more than one mistake.

The grade "**satisfactory**" is given to the student if the student gave incomplete and insufficiently detailed answers to the questions on the ticket. The logic and sequence of presentation have violations. Mistakes were made in the disclosure of concepts, the use of terms. The student is not able to independently distinguish essential and insignificant features and cause-and-effect relationships. There are no conclusions in the answer. The ability to reveal the meaning of generalized knowledge is not shown. Speech design requires corrections, correction, demonstrated a mediocre ability to apply the knowledge gained in practice through solving a specific problem, hardly coped with the task, while making no more than two mistakes.

The grade "**unsatisfactory**" is given to the student if the answer is scattered knowledge with significant errors on the question. There are fragmentarity, illogical presentation. The student is not aware of the connection of the discussed issues with other objects of the discipline. There are no conclusions, concretization and evidence of the presentation. The speech is illiterate, histological terminology is not used. Additional and clarifying questions of the teacher do not lead to correction of the student's answer, the answer to the questions is completely absent or the student refused to answer, did not demonstrate the ability to apply the knowledge gained in practice through solving a specific problem, did not cope with the task or made three or more serious mistakes in solving it.

2. Description of the grading scale

The maximum possible score for the entire current control is set at **55**. The current control measure is considered passed if the student received at least 60% of the maximum score established for this control. The rating score given to the student for the current control measure passed by the student within the time limits established by the schedule of control measures is determined as follows:

Level of completion of the control task	Rating score (% of the maximum score for the control task)
Excellent	100
Good	80
Satisfactory	60
Unsatisfactory	0

3. Methodological materials that determine the procedures for assessing knowledge, skills, and (or) work experience that characterize the stages of competence formation

The procedure for conducting this assessment includes:

- a separate interview on each of these topics;
- compilation of a synopsis.

The proposed tasks allow you to test the competencies **of OPK-1**

To prepare for each topic, it is necessary to study and take notes of the recommended sources and scientific literature and, on their basis, prepare for the issues for discussion specified in the methodological materials.

When checking the task, the following are evaluated

- confident mastery of the material;
- the ability to express one's thoughts clearly and logically;
- the ability to creatively approach the solution of the main issues of the topic;
- independent thinking,
- full compliance of the synopsis with the established requirements;
- independent oral speech, fully revealing the essence of the problems of the interview.

7. PRACTICAL READINESS INTERVIEW QUESTIONS

Questions for the defense of practical works

Topic: Formats for representing constants and data, types of operations, input-output in C++

1. Formats for representing constants and data, types of operations, input-output in C++

Topic: Study of Programming of Branching Structure Algorithms

1. If and switch statements in C and C++.

Topic: Study of programming algorithms of cyclic structure

1. Statements while, do ... while and for C and C++.

Topic: Study of Algorithm Programming Using Static Arrays

1. Definition and use of static arrays in the C and C++ programming languages.

Topic: Study of algorithm programming using dynamic arrays

1. The relationship between arrays and pointers in C and C++.
2. Allocation and release of dynamic memory in C and C++.
3. Accessing dynamic array elements.

Topic: Study of algorithm programming using multidimensional arrays

1. Definition and use of static and dynamic multidimensional arrays in C and C++.

Topic: Study of String Processing Tools in C++

1. Defining and Manipulating C++ Standard Library Strings.

Topic: Study of programming algorithms for processing ASCIIZ strings

1. Definition and use of ASCIIZ strings in C and C++.
2. Explore the standard library tools for processing ASCIIZ strings.

Topic: Exploring the possibilities of code reuse in structured programming

1. Define and use common and embedded functions in C and C++.

Topic: Investigation of preprocessor tools

1. Preprocessor tools in C and C++.
2. Definition and use of macro definitions in C and C++.
3. Comparison of macro definitions and functions.

Topic: Study of means for creating multi-module programs

1. Conditional compilation directives.
2. Create plug-in files.
3. Static and dynamic libraries.
4. Compile multi-module programs using the GCC and Clang compilers.

Topic: Studies of the means of creating classes and objects in the C++ language

1. Class definition.
2. Class usage.
3. Defining the methods of a class.

4. Nested classes.

Topic: Investigation of Object Initialization Tools and Operations Overload in C++ Language

1. Overload of operations.
2. Constructors and destructor.
3. Constants in the classroom.
4. Array fields in the classroom.
5. Static elements of a class.

Topic: Class inheritance in C++

1. Simple open inheritance.
2. Constructors and destructors in inheritance.
3. Fields and methods in inheritance.
4. Static elements of a class in inheritance.
5. Nested classes and inheritance.
6. Assignment operators and the principle of substitution.
7. Functions-transformation operations.
8. Closed and protected inheritance.
9. Virtual and abstract methods.
10. Abstract classes.

Topic: Study of I/O Facilities in C++

1. Classification of flows.
2. Connect streams.
3. I/O operations.
4. Flow state.
5. I/O formatting.
6. File streams.
7. Buffering.
8. String streams.
9. Positioning in the flow.
10. Wide flows.

1. Criteria for assessing competencies

The grade "excellent" is given if the student has demonstrated a high ability to apply the knowledge gained in practice through solving a specific problem using algorithms and programming languages, fluently coped with the task without difficulty, showing the possession of versatile techniques and skills for its implementation, did not make mistakes and inaccuracies;

The grade "good" is given if the student has demonstrated the ability to apply the knowledge gained in practice through solving a specific problem using algorithms and programming languages, coped with the task, showing the possession of the necessary techniques and skills for its implementation, while making no more than one mistake;

A "satisfactory" grade is given if the student has demonstrated a mediocre ability to apply the acquired knowledge in practice through solving a specific problem using algorithms and programming languages, has difficulty coping with the task, while making no more than two mistakes;

The grade "unsatisfactory" is given if the student has not demonstrated the ability to apply the acquired knowledge in practice through solving a specific problem using algorithms and programming languages, has not coped with the task or has made three or more serious mistakes in solving it.

2. Description of the grading scale

The maximum possible score for the entire current control is set at **55**. The current control measure is considered passed if the student received at least 60% of the maximum score established for this control. The rating score given to the student for the current control measure passed by the student within the time limits established by the schedule of control measures is determined as follows:

Level of completion of the control task	Rating score (% of the maximum score for the control task)
Excellent	100
Good	80
Satisfactory	60
Unsatisfactory	0

3. Methodological materials that determine the procedures for assessing knowledge, skills, and (or) work experience that characterize the stages of competence formation

The procedure for conducting this assessment includes:

- interview;
- defense of practical works.

The proposed tasks allow you to test the competencies **of OPK-1**.

For the practical lesson, the student must prepare answers to questions, work out additional scientific sources, make a summary of theoretical material (if provided).

The maximum number of points is received by the student if he completes the practical task, knows the material, is able to express thoughts logically and clearly, and shows independence in completing the task.

The grounds for reducing the score are:

- poor knowledge of the topic and incompletely completed task;
- uncompleted task for laboratory work;
- lack of ability to apply theoretical knowledge to solve practical problems.

When checking the task, the following are evaluated

- active participation in the work;
- confident mastery of the material;
- the ability to express one's thoughts clearly and logically;
- the ability to creatively approach the solution of tasks;
- independent thinking,
- independent oral speech, fully revealing the essence of the problems of the interview.