

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

АЛГОРИТМИЗАЦИЯ И ПРОГРАММИРОВАНИЕ

Методические указания по выполнению лабораторных работ

Направление подготовки 03.03.02 Физика
Направленность (профиль)
«Фундаментальная и прикладная физика»

**Ставрополь
2026**

Оглавление

ВВЕДЕНИЕ.....	5
Лабораторная работа № 1 Запись арифметических выражений	6
Теоретическая часть Основные понятия алгоритмизации и программирования	6
1.1. Этапы решения задач на ЭВМ	6
1.2. Основы алгоритмизации.....	9
1.3. Теоретические основы программирования.....	17
1.4. Правила записи в C++ арифметических выражений	17
Практическая часть.....	19
Лабораторная работа № 2 Программирование алгоритмов линейной структуры	23
Теоретическая часть Базовые средства языка программирования C++	23
2.1. Общая характеристика языка программирования C++	23
2.2. Основные понятия языка	24
2.3. Данные и способы их организации	26
2.4. Стандартные простые типы данных.....	28
2.5. Структура программы.....	31
2.6. Операторы	34
2.7. Организация ввода/вывода данных	36
Практическая часть.....	42
Лабораторная работа № 3 Программирование алгоритмов разветвляющейся структуры	52
Теоретическая часть Операторы ветвления	52
3.1. Простые и составные условия.....	52
3.2. Составной оператор	53
3.3. Условная операция (?:)	54
3.4. Условный оператор if.....	55
3.5. Оператор switch	57
3.6. Оператор перехода goto	58
Практическая часть.....	59
Лабораторная работа № 4 Программирование алгоритмов циклической структуры	68
Теоретическая часть Циклы.....	68
4.1. Оператор цикла с параметром (for)	69
4.2. Оператор цикла с предусловием (while)	70
4.3. Оператор цикла с постусловием (do while)	71
4.4. Вложенные циклы	72
Практическая часть.....	74
Лабораторная работа №5 Использование функций для решения прикладных задач	83
Теоретическая часть Подпрограммы	83

5.1.	Понятие подпрограммы	83
5.2.	Формальные и фактические параметры	84
5.3.	Локальные и глобальные переменные	84
5.4.	Функции	86
Практическая часть.....		89
Лабораторная работа №6 Программирование рекурсивных алгоритмов.....		94
Теоретическая часть Итерация и рекурсия.....		94
6.1.	Понятие итеративного процесса	94
6.2.	Понятие рекурсии.....	94
Практическая часть.....		96
Лабораторная работа №7 Использование числовых одномерных массивов		103
Теоретическая часть Одномерные массивы		103
7.1.	Понятие структурированного типа данных	103
7.2.	Понятие и описание типа массив	103
7.3.	Одномерные массивы	105
7.4.	Основные действия над элементами массивов	106
Практическая часть.....		108
Лабораторная работа №8 Двумерные массивы		114
Теоретическая часть Двумерные массивы		114
Практическая часть.....		118
Лабораторная работа № 9 Задачи сортировки и поиска.....		123
Теоретическая часть Алгоритмы решения задач внутренней сортировки и алгоритмы поиска информации		123
9.1.	Сложность алгоритмов	123
9.2.	Постановка задачи поиска	124
9.3.	Последовательный (линейный) поиск.....	124
9.4.	Бинарный поиск.....	125
9.5.	Постановка задачи сортировки данных	125
9.6.	Прямые и быстрые методы внутренней сортировки	126
9.7.	Сортировка вставками	127
9.8.	Сортировка с помощью прямого выбора.....	128
9.9.	Сортировка с помощью прямого обмена	129
Практическая часть.....		130
Лабораторная работа №10 Применение массивов и указателей для решения прикладных задач		132
Теоретическая часть Указатели и массивы		132
10.1.	Понятие статической и динамической переменной.....	132
10.2.	Указатели.....	132
10.3.	Взаимосвязь между массивами и указателями	134
10.4.	Порядок объявления динамических массивов.....	136
10.5.	Передача массивов в качестве параметров функции	137
Практическая часть.....		139

Лабораторная работа № 11 Исследование способов работы с функциями...	148
Теоретическая часть Особенности работы с функциями	148
11.1. Способы передачи параметров в функцию	148
11.2. Передача имен функций в качестве параметров	150
11.3. Перегрузка функций.....	151
Практическая часть.....	152
Лабораторная работа № 12 Обработка символьных массивов	157
Теоретическая часть Строки как массив элементов типа char	157
12.1. Способы представления строк в СИ; реализация некоторых типовых операций над строками	157
12.2. Основные функции Си для работы со строками	160
Практическая часть.....	161
Лабораторная работа № 13 Программирование задач на обработку данных строкового типа	166
Теоретическая часть Строки как объект специального класса string.....	166
13.1. Описание строк в C++ и типовые операции над ними	166
13.2. Основные функции класса string для работы со строками.....	167
Практическая часть.....	169
Лабораторная работа № 14 Применение структур для решения прикладных задач	174
Теоретическая часть Структуры	174
14.1. Порядок объявления и инициализации структур	174
14.2. Программирование с использованием структур	176
14.3. Использование функций для работы с производными типами данных.....	176
Практическая часть.....	177
Лабораторная работа № 15 Исследование методов доступа к файлам данных .	186
Теоретическая часть Файлы.....	186
15.1. Подход с использованием возможностей языка Си.....	186
15.2. Подход с использованием возможностей языка C++	189
Практическая часть.....	190
СПИСОК ЛИТЕРАТУРЫ	195
Приложение 1 Содержание отчета, порядок выполнения лабораторных работ и оборудование и материалы	196
Приложение 2 Базовые функции	197
Приложение 3 Правила поведения в компьютерном классе	198

ВВЕДЕНИЕ

Учебное пособие: лабораторный практикум по дисциплине «Алгоритмизация и программирование» подготовлено в соответствии с Федеральным государственным образовательным стандартом высшего образования.

Дисциплина «Алгоритмизация и программирование» имеет **целью** формирование теоретических знаний, практических навыков и умений в использовании современных информационных технологий, в изучении основ функционирования ЭВМ и систем, в организации информационных процессов в различных сферах.

Изучение дисциплины способствует пониманию студентами основ организационной, функциональной и физической структуры информационных процессов, формированию у них инженерного мышления, развитию способностей комплексного подхода к решению проблем обработки информации.

Учебное пособие может оказать помощь студентам в самостоятельном изучении программного материала, самоконтроле, выработке способности к разработке алгоритмических и программных решений в области системного и прикладного программирования, математических, информационных и имитационных моделей, созданию информационных ресурсов глобальных сетей, прикладных баз данных, тестов и средств тестирования систем и средств. Оно может быть использовано для организации самостоятельной работы студентов как очного, так и заочного отделений.

Пособие представляет собой материалы по организации лабораторных занятий по дисциплине «Алгоритмизация и программирование» для направления 03.03.02 «Физика».

При разработке пособия в соответствии с требованиями ФГОС ВО для реализации компетентностного подхода широко использованы активные и интерактивные формы обучения.

Лабораторная работа № 1

Запись арифметических выражений

ЦЕЛЬ РАБОТЫ: закрепление знаний по теоретическим основам алгоритмизации и программирования, приобретение навыков использования арифметических операций, функций и правил записи арифметических выражений согласно синтаксису языка C++.

Теоретическая часть

Основные понятия алгоритмизации и программирования

1.1. Этапы решения задач на ЭВМ

Задачи, решаемые с помощью ЭВМ, можно классифицировать по различным критериям: по типу информации и информационным технологиям, по способу поиска решения (простые и переборные), по характеру целей (задачи оптимизации, управления, обучения, информационного поиска), по функциональному назначению, а также по уровню достижения цели и уровню их решения, в т. ч. по уровню автоматизации этапов их решения.

Задача становится разрешимой, если найдено правило, способ получения результата. В информатике такое правило называют алгоритмом.

Решение задачи на ЭВМ состоит из нескольких этапов, среди которых основными являются следующие:

1. Постановка задачи:

- сбор информации о задаче;
- формулировка условия задачи;
- определение конечных целей решения задачи;
- определение формы выдачи результатов;
- описание данных (их типов, диапазонов величин, структуры и т. п.).

2. Формализация (анализ и исследование задачи, модели, представление ее в виде уравнений, соотношений, ограничений и т.п.):

- анализ существующих аналогов задачи;
- анализ технических и программных средств;
- разработка математической модели;
- разработка структур данных.

Понятие моделирования

При решении задачи обычно исследуют не реальный объект, а его модель.

Модель – искусственно созданный объект, обладающий всеми существенными признаками реального объекта, явления или процесса.

Моделирование – это метод познания, состоящий в создании и исследовании моделей.

Цели моделирования:

- 1) понять сущность изучаемого объекта;
- 2) научиться управлять объектом и определять наилучшие способы управления;
- 3) прогнозировать прямые или косвенные последствия;
- 4) решать прикладные задачи.

Математическая модель – это система математических соотношений (данных) – формул, уравнений, неравенств и т.д. и отношений между ними, описывающих поведение объекта с некоторой степенью точности и отражающих существенные свойства моделируемого процесса (объекта или явления).

Порядок составления математической модели:

- выделить реальный объект, на котором будет основываться математическая модель, и из множества его свойств, закономерностей, внутренних связей, отдельных характеристик явления и параметров выделяем те, которые являются существенными для решаемой задачи, и отбросить несущественные;
- определить, что считать исходными данными и результатами;
- подобрать математический объект с тем же числом подобных параметров, отражающий суть реального объекта; записать математические соотношения, связывающие результаты с исходными данными.

3. Выбор метода решения.

4. Разработка алгоритма:

- выбор метода проектирования алгоритма;
- выбор формы записи алгоритма (блок-схемы, псевдокод и др.);
- выбор тестов и метода тестирования;
- проектирование алгоритма.

5. Программирование:

- выбор языка программирования;
- уточнение способов организации данных;
- запись алгоритма на выбранном языке программирования.

6. Тестирование, отладка и исправление обнаруженных ошибок:

- синтаксическая отладка;
- отладка семантики и логической структуры;
- тестовые расчёты и анализ результатов тестирования;
- совершенствование программы.

Отладка программы

Отладка программы – это процесс поиска и устранения ошибок в программе, производимый по результатам ее прогона на компьютере.

В современных программных системах отладка осуществляется с использованием специальных программных средств, называемых отладчиками. Эти средства позволяют **исследовать внутреннее поведение программы.**

Программа-отладчик обычно обеспечивает следующие возможности:

- **пошаговое исполнение программы** с остановкой после каждой команды (оператора);
- **просмотр текущего значения любой переменной или нахождение значения любого выражения**, в том числе, с использованием стандартных функций; при необходимости можно установить новое значение переменной;
- **установку в программе "контрольных точек"**, т.е. точек, в которых программа временно прекращает свое выполнение, так что можно оценить промежуточные результаты, и др.

При отладке программ **важно помнить следующее:**

- в начале процесса отладки надо использовать **простые тестовые данные**;
- возникающие затруднения следует четко **разделять и устранять строго поочередно**;
- **не нужно считать причиной ошибок машину**, так как современные машины и трансляторы обладают чрезвычайно высокой надежностью.

Тест и тестирование программы

Тестирование – это испытание, проверка правильности работы программы в целом либо ее составных частей.

При отладке происходит локализация и устранение синтаксических ошибок и явных ошибок кодирования, в процессе же тестирования проверяется работоспособность программы, не содержащей явных ошибок. Тестирование устанавливает факт наличия ошибок, а отладка выясняет причину неправильной работы программы.

Как бы ни была тщательно отлажена программа, решающим этапом, устанавливающим ее пригодность для работы, является контроль программы по результатам ее выполнения на системе тестов.

Программу условно можно считать правильной, если её запуск для выбранной системы тестовых исходных данных во всех случаях дает правильные результаты. Но, как справедливо указывал известный теоретик программирования Э. Дейкстра, тестирование может показать лишь наличие ошибок, но не их отсутствие. Нередки случаи, когда новые входные данные вызывают "отказ" или получение неверных результатов работы программы, которая считалась полностью отлаженной.

Для реализации метода тестов должны быть изготовлены или заранее известны эталонные результаты. Вычислять эталонные результаты нужно обязательно до, а не после получения машинных результатов. В противном случае имеется опасность невольной подгонки вычисляемых значений под желаемые, полученные ранее на машине.

7. Анализ результатов решения задачи и уточнение математической модели с повторным выполнением этапов 2 - 6 (при необходимости).

8. Сопровождение программы: это работы, связанные с обслуживанием программ в процессе их эксплуатации:

- доработка программы для решения конкретных задач;
- составление документации к решённой задаче, к математической модели, к алгоритму, к программе, к набору тестов, к использованию программы.

1.2. Основы алгоритмизации

Алгоритм – это метод (способ) решения задачи, записанный по определённым правилам, в виде конечной последовательности однозначных предписаний, исполнение которых позволяет с помощью конечного числа шагов получить решение задачи, однозначно определяемое исходными данными из некоторого множества значений.

Свойства алгоритма

1. **Дискретность** (прерывность, раздельность). Алгоритм должен представлять процесс решения задачи как последовательное выполнение конечного числа простых (или ранее определенных) законченных действий шагов.
2. **Понятность** для исполнителя – т.е. исполнитель алгоритма должен знать, как его выполнять.
3. **Определенность** (точность, детерминированность). Каждое правило алгоритма должно быть четким и однозначным, содержать действия над известными данными. Каждое действие должно быть понятно исполнителю (для каждого алгоритма предполагается конкретный исполнитель).

Замечание. Часто под свойством детерминированности алгоритма понимается одновременное выполнение свойств точности и понятности.

4. **Результативность** (или конечность). Алгоритм должен приводить к решению задачи, получение определенного результата за конечное число шагов.
5. **Правильность.** Способность алгоритма обеспечить получение именно того результата, который требуется. Неправильность может объясняться неполнотой наших представлений о свойствах объекта или упущением в решении. Для доказательства правильности алгоритма задача часто делится на блоки и правильность доказывается для каждого блока, хотя такая проверка не является полной.
6. **Массовость.** Алгоритм решения задачи разрабатывается в общем виде, т.е. он должен быть применим для некоторого класса задач, различающихся лишь исходными данными. При этом исходные данные могут выбираться из некоторой области, которая называется областью применимости алгоритма.
7. **Универсальность.** Алгоритм должен быть составлен так, чтобы им мог воспользоваться любой исполнитель для решения аналогичной задачи. (Например, правила сложения и умножения чисел годятся для любых чисел, а не для каких-то конкретных.)
8. **Эффективность.** Выбор алгоритмы, который будет выполнен за минимальное время, с минимальными затратами ресурсов.

Таким образом, исполнитель действует **формально**, т.е. отвлекается от содержания поставленной задачи, а только строго выполняет некоторые правила, инструкции и вместе с тем получать нужный результат.

Критерии качества алгоритма

1. **Связанность.** Определяется количеством промежуточных результатов. Чем выше количество промежуточных результатов, тем ниже связанность.
2. **Объем алгоритма.** Это количество операций или шагов, которые необходимо выполнить, а также сложность этих шагов.
3. **Логическая сложность.** Определяется количеством ветвлений и циклов.

Порядок выполнения алгоритма

1. Действия в алгоритме выполняются в порядке их записи.
2. Нельзя менять местами никакие два действия алгоритма.
3. Нельзя не закончив одного действия переходить к следующему.

Способы описания алгоритмов

1. **Словесно-формульный.** Описание алгоритма с помощью слов и формул на естественном языке.

Словесный способ не имеет широкого распространения по следующим причинам:

- такие описания строго не формализуемы;
- страдают многословностью записей;
- допускают неоднозначность толкования отдельных предписаний. «Он встретил ее на поле с цветами».

Пример. Составить алгоритм начисления зарплаты согласно следующему правилу: если стаж работы сотрудника менее 5 лет, то зарплата 130 руб., при стаже работы от 5 до 15 лет – 180 руб., при стаже свыше 15 лет зарплата повышается с каждым годом на 10 руб.

Словесно-формульное описание алгоритма решения задачи:

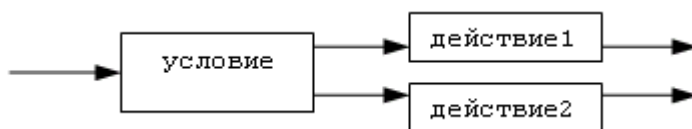
1. Ввести ST , перейти к п. 2.
2. Если $ST < 5$, то $ZP := 130$, перейти к п. 4, иначе — перейти к п. 3.
3. Если $ST < 15$, то $ZP := 180$, перейти к п. 4, иначе $ZP := 180 + (ST - 15)10$, перейти к п. 4.
4. Вывести (отпечатать) значение ZP , перейти к п. 5.
5. Вычисления прекратить.

2. **Табличный.** Алгоритм представляется в форме таблицы и расчётных формул (физика, химия и т. д.).

3. **Структурограмма**

условие	
то	иначе
действие1	действие2

4. Синтаксическая диаграмма (формулы Бэкуса-Наура)



5. **Псевдокоды.** Полуформализованные описания алгоритмов на условном алгоритмическом языке, включающие в себя как элементы языка программирования, так и фразы естественного языка, общепринятые математические обозначения и др.

Псевдокод занимает промежуточное место между естественным и формальным языками. С одной стороны, он близок к обычному естественному языку, поэтому алгоритмы могут на нем записываться и читаться как обычный текст. С другой стороны, в псевдокоде используются некоторые формальные конструкции и математическая символика, что приближает запись алгоритма к общепринятой математической записи.

В псевдокоде не приняты строгие синтаксические правила для записи команд, присущие формальным языкам, что облегчает запись алгоритма на стадии его проектирования и дает возможность использовать более широкий набор команд. Однако в псевдокоде, так же, как и в формальных языках, есть служебные слова, смысл которых определен раз и навсегда.

Единого или формального определения псевдокода не существует, поэтому возможны различные псевдокоды, отличающиеся набором служебных слов и основных (базовых) конструкций.

Примером псевдокода является школьный алгоритмический язык.

Алгоритмический язык – это средство для записи алгоритмов в аналитическом виде, промежуточном между записью алгоритма на естественном (человеческом) языке и записью на языке ЭВМ (языке программирования).

Запись алгоритма решения задачи примера 1 на алгоритмическом языке:

```
алг зарплата(цел. ST, вещ ZP)
  арг ST
  рез ZP
нач
  если ST<5
    то ZP:=150
  иначе
    если ST<=15
      то ZP:=180
    иначе ZP:=180+(ST-15)*10
  все
все
кон
```

6. **Программный.** Описание алгоритма с помощью языков программирования.
7. **Графический.** Алгоритм изображается в виде последовательности связанных между собой функциональных блоков, каждый из которых соответствует выполнению одного или нескольких действий. Такое графическое представление называется схемой алгоритма или **блок-схемой**.

Блок-схема алгоритма представляет собой систему связанных геометрических фигур.

Правила построения блок-схем

1. В блок-схеме каждому типу действий (вводу исходных данных, вычислению значений выражений, проверке условий, управлению повторением действий, окончанию обработки и т.п.) соответствует геометрическая фигура, представленная в виде **блочного символа**. Для наглядности операции разного вида изображаются в схеме различными геометрическими фигурами.
2. Блочные символы соединяются **линиями переходов**, определяющими очередность выполнения действий. Порядок выполнения действий указывается стрелками, соединяющими блоки.
3. В схеме блоки стараются размещать сверху вниз, в порядке их выполнения.
4. Все повороты соединительных линий выполняются под углом 90 градусов.

В таблице приведены наиболее часто употребляемые символы.

Название	Обозначение и пример заполнения	Выполняемая функция (пояснение)
Начало/конец (вход/выход)		Начало или конец программы, вход или выход в подпрограмму
Блоки ввода/вывода		Ввод-вывод данных
		Вывод данных на печатающее устройство
Блок вычислений		Арифметический блок определяет вычислительное действие или последовательность действий
Логический блок		Логический блок проверяет истинность или ложность условия и выбирает направления выполнения алгоритма в зависимости от условия. В блоке должны быть указаны вопрос, условие или сравнение, которые он определяет.
Предопределенный процесс		Вычисления по стандартной или пользовательской подпрограмме
Блок модификации		Выполнение действий, изменяющих пункты алгоритма, начало цикла. Внутри блока записывается параметр цикла, для которого указываются его начальное значение, граничное условие и шаг изменения значения параметра для каждого повторения.
Межстраничный соединитель		Указание связи между частями схемы, расположенной на разных страницах

Общие правила построения схемы алгоритма задачи

1. Выявить исходные данные, результаты, назначить им имена.
2. Выбрать метод (порядок) решения задачи.
3. Разбить метод решения задачи на этапы (с учетом возможностей ЭВМ).
4. Изобразить каждый этап в виде соответствующего блока схемы алгоритма и указать стрелками порядок их выполнения.
5. В полученной схеме при любом варианте вычислений:
 - а) предусмотреть выдачу результатов или сообщений об их отсутствии;
 - б) обеспечить возможность после выполнения любой операции, так или иначе, перейти к блоку *Останов* (к выходу схемы).

Эти правила и есть «**Основные принципы алгоритмизации**». Будем считать, что знание и применение настоящих «принципов» обязательно при составлении алгоритма любой задачи.

Типы алгоритмов

- структурированные;
- неструктурированные (т.е. с нарушением структуры – с операторами безусловного перехода);
- вспомогательные (используемые в составе других алгоритмов).

Виды алгоритмов

- линейный алгоритм;
- алгоритм ветвления;
- циклический алгоритм;
- алгоритм с подпрограммами;
- смешанные (т.е. содержащие и циклы, и ветвление, и функции).
- рекурсивный алгоритм обращается к самому себе, пока не выполнится определенное условие.

Базовые алгоритмические конструкции

Структурное программирование – методология разработки программного обеспечения, в основе которой лежит представление программы в виде иерархической структуры блоков. Предложена в 70-х годах XX века Э. Дейкстрой, разработана и дополнена Н. Виртом. Основывается на теореме о структуре.

Согласно **теореме о структуре** (теорема Бёма – Якопини, 1966 г) логическая структура любого алгоритма может быть представлена комбинацией трех базовых структур: *следование, ветвление, цикл*.

Характерной особенностью базовых структур является наличие в них одного входа и одного выхода.

1. **Базовая структура следование (линейный алгоритм)**. Образуется из последовательности действий, следующих одно за другим:

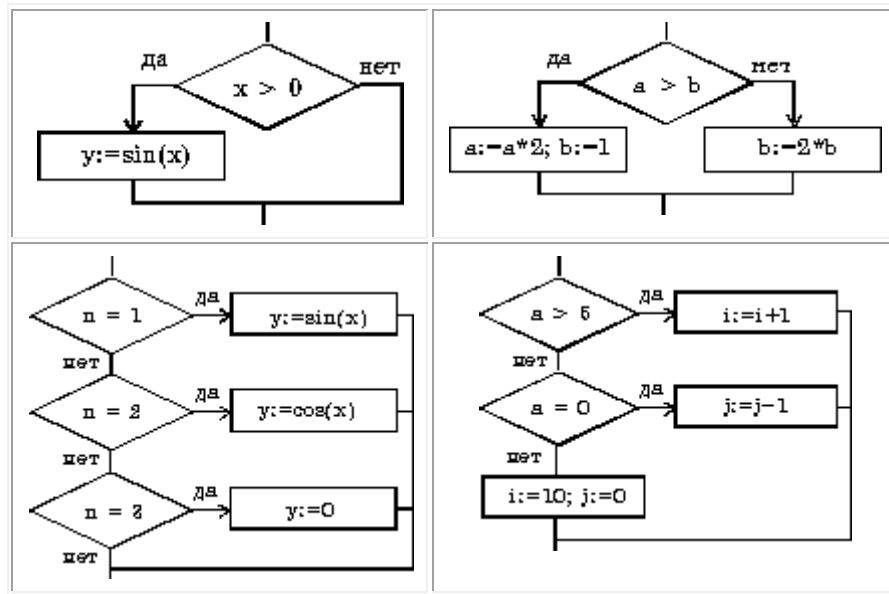


2. **Базовая структура ветвление (алгоритм ветвления)** обеспечивает в зависимости от результата проверки условия выбор одного из альтернативных путей работы алгоритма. Условие – вопрос, имеющий два варианта ответа: **да** или **нет**. Каждый из путей ведет к **общему выходу**, так что работа алгоритма будет продолжаться независимо от того, какой путь будет выбран. Запись ветвления выполняется в двух формах: полной и неполной.

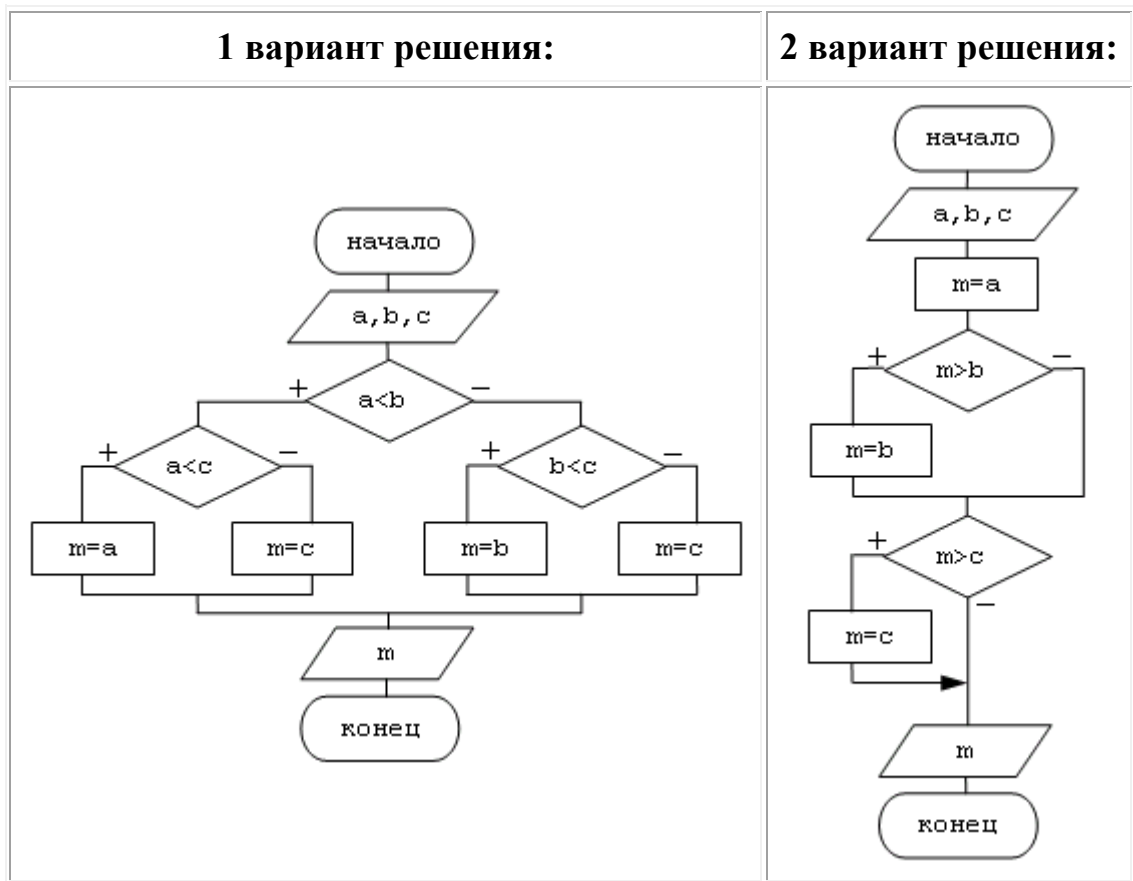
Структура **ветвление** существует в четырех основных вариантах:

Неполная форма записи	Полная форма записи
1. если-то <pre> graph TD Start(()) --> Cond{условие} Cond -- "+" --> Act1[действие 1] Act1 --> Join(()) Cond -- "-" --> Join Join --> Exit(()) </pre>	2. если-то-иначе <pre> graph TD Start(()) --> Cond{условие} Cond -- "+" --> Act1[действие 1] Act1 --> Act2[действие 2] Act2 -.-> ActN[действие n] ActN --> Join1(()) Cond -- "-" --> Act1p[действие 1'] Act1p --> Act2p[действие 2'] Act2p -.-> ActNp[действие n'] ActNp --> Join2(()) Join1 --> Exit(()) Join2 --> Exit </pre>
3. выбор <pre> graph TD Start(()) --> Cond1{условие 1} Cond1 -- да --> Act1[действие 1] Act1 --> Cond2{условие 2} Cond2 -- да --> Act2[действие 2] Act2 --> CondN{условие N} CondN -- да --> ActN[действие N] ActN --> Exit(()) Cond1 -- нет --> Exit Cond2 -- нет --> Exit CondN -- нет --> Exit </pre>	4. выбор-иначе <pre> graph TD Start(()) --> Cond1{условие 1} Cond1 -- да --> Act1[действие 1] Act1 --> Cond2{условие 2} Cond2 -- да --> Act2[действие 2] Act2 --> CondN{условие N} CondN -- да --> ActN[действие N] ActN --> Exit(()) Cond1 -- нет --> Cond2 Cond2 -- нет --> CondN CondN -- нет --> ActN1[действие N+1] ActN1 --> Exit </pre>

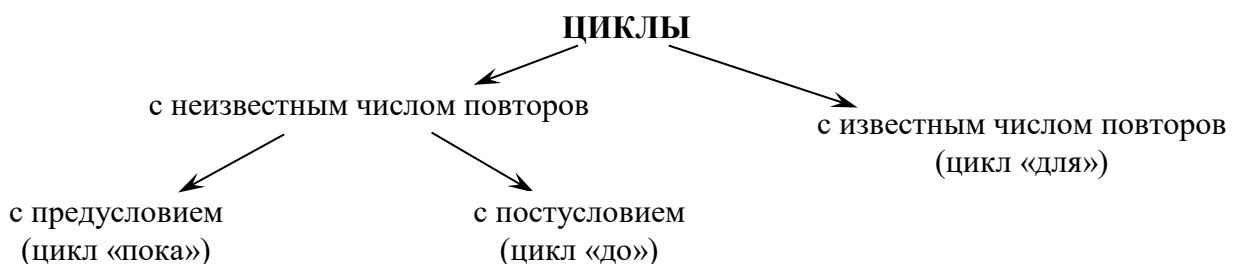
Примеры команды *если*



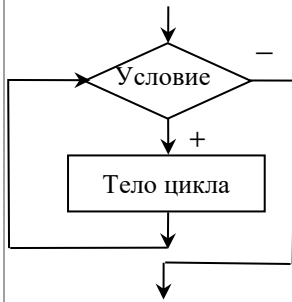
Пример: найти наименьшее из трех чисел.



3. Базовая структура **цикл** обеспечивает многократное выполнение некоторой совокупности действий, называемой **телом цикла**, над новыми данными.



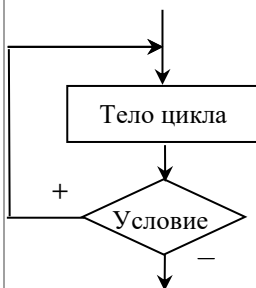
Цикл типа «пока»



Выполнение цикла «пока» начинается с проверки условия, поэтому такую разновидность циклов называют циклами **с предусловием**. Переход к выполнению действия осуществляется только в том случае, если условие выполняется, в противном случае происходит выход из цикла. Тело цикла выполняется до тех пор, пока выполняется условие. Условие цикла необходимо подобрать так, чтобы действия, выполняемые в цикле, привели к нарушению его истинности, иначе произойдет закичивание.

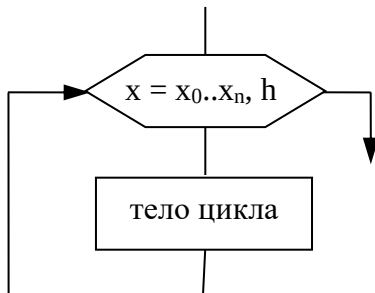
Закичивание - бесконечное повторение выполняемых действий.

Цикл типа «до»



Исполнение цикла начинается с выполнения действия. Таким образом, тело цикла будет реализовано хотя бы один раз. После этого происходит проверка условия. Поэтому цикл "до" называют циклом **с постусловием**. Если условие выполняется, то происходит возврат к выполнению действий, иначе осуществляется выход из цикла. Для предотвращения закичивания необходимо предусмотреть действия, приводящие к ложности условия.

Цикл типа «для»



Цикл с параметром, или цикл со счетчиком, или арифметический цикл - это цикл с заранее известным числом повторов. Предписывает выполнять тело цикла для всех значений некоторой переменной (параметра цикла) в заданном диапазоне.

x_0 – начальное значение параметра; h – шаг; x_n – последнее значение параметра.

Для создания циклов с параметром необходимо использовать правила:

1. Параметр цикла, его начальное и конечное значения и шаг должны быть одного типа.
2. Запрещено изменять в теле цикла начальное, текущее и конечное значения для параметра.
3. Запрещено входить в цикл, минуя блок модификации.
4. После выхода из цикла значение переменной параметра неопределенно и не может использоваться в дальнейших вычислениях.
5. Из цикла можно выйти, не закончив его, тогда переменная параметр сохраняет свое последнее значение.

1.3. Теоретические основы программирования

Программирование – это раздел информатики, изучающий методы и приемы составления программ для компьютеров. Кроме того, программирование – это подготовка задачи к решению ее на компьютере.

Программа – последовательность действий, которые должен выполнить компьютер в строго указанной очередности для достижения конкретного результата.

Языки программирования – это совокупность средств и правил представления алгоритма в виде, приемлемом для компьютера.

Системы программирования – это набор средств ввода, редактирования, трансляции и выполнения программ на ЭВМ.

Транслятор – это устройство или комплекс программ, обеспечивающий перевод программы, написанной на символическом языке, в совокупность машинных команд, либо передающие/преобразующие данные или другую программу.

Например, транслятор воспринимает операторы одного языка и вырабатывает соответствующие операторы другого языка.

Компилятор – это транслятор, обеспечивающий перевод программы, написанной на алгоритмическом языке, в совокупность машинных команд без ее выполнения в компьютере.

Компилятор оценивает исходный текст в соответствии с синтаксической конструкцией языка и переводит на машинный язык.

Например, компилятор берет программу, написанную на языке C, и преобразует ее в программу на языке ассемблера.

Интерпретатор – это транслятор, обеспечивающий перевод каждой конструкции алгоритмического языка в машинные команды и одновременное выполнение этой конструкции в компьютере.

При исполнении программных операторов, интерпретатор должен сначала сканировать каждый оператор с целью прочтения его содержимого, а затем выполнить запрошенную операцию.

1.4. Правила записи в C++ арифметических выражений

Арифметические выражения

Выражение состоит из операторов и операндов. Операндами могут быть, в свою очередь, выражения или одни из его частных случаев – числа (константы) или переменные, **операторы** обозначают выполняемые над ними действия (+ сложение, - вычитание, * умножение, / деление (для целых операндов – целая часть от деления), % остаток от деления (только для целых), ...).

Все основные операции языка C++ можно разбить на следующие группы:

- | | |
|---------------------------------|---------------------------|
| - арифметические операции; | - операции со строками; |
| - логические операции; | - операции присваивания; |
| - операции отношения; | - операция sizeof; |
| - операции с битами информации; | - условная операция (?:). |

Примеры выражений:

$(a + 0.12)/6$

$x \ \&\& \ || \ !z$

$(t * \sin(x) - 1.05e4) / ((2 * k + 2) * (2 * k + 3))$

В выражение могут входить операнды различных типов, чего, строго говоря, не следует допускать. Если операнды имеют одинаковый тип, то результат операции будет иметь тот же тип. Если операнды различного типа, перед вычислениями выполняются преобразования типов по определенным правилам, обеспечивающим преобразование более коротких типов в более длинные для сохранения значимости и точности.

Порядок вычисления выражений определяется рангом (приоритетом) входящих в него операций (табл. 3). Принятый в C++ ранг операций наиболее близок к математическому, также как и принятый порядок их вычисления. Так, умножение и деление (мультипликативные операции) старше сложения и вычитания (аддитивные операции). Унарные операции $+$ и $-$ старше бинарных, т.е., знак операнда вычисляется в первую очередь. Операции типа присваивания младше прочих, что позволяет выполнить их только после того, как значение выражения вычислено полностью. Операции отношения младше арифметических операций, что позволяет использовать естественную запись логических выражений, например, $x > 0 \ \&\& \ y > 0$. Здесь в первую очередь вычисляются значения отношений, которые затем являются операндами конъюнкции.

Таблица 1 – Порядок вычисления выражений

Группа	Тип действий	Операции или элементы
1	Вычисления в круглых скобках	()
2	Вычисления значений функции	Функции
3	Унарные операции	++, --, ~, !, унарные - и +, &, * (разадресация), ...
4	Операции типа умножения	*, /, %
5	Операции типа сложения	+, -
6	Операции отношения	<, <=, >, >=, ==, !=

Арифметические выражения строятся из операндов, арифметических операций и круглых скобок.

Круглые скобки используются для заключения в них части выражения, значения которой необходимо выполнить в первую очередь. В выражении может быть любое количество круглых скобок, причем количество открывающих круглых скобок должно быть равно количеству закрывающих. Части выражений, заключенные в круглые скобки, должны быть либо не пересекающимися, либо вложенными друг в друга.

Арифметические выражения записываются по следующим правилам:

- Запись ведётся в строчку.
- Нельзя опускать знак умножения между сомножителями.
- Для обозначения переменных используются буквы латинского алфавита.
- Операции выполняются в соответствии с *приоритетами*: сначала вычисление функций, затем умножение и деление, потом сложение и вычитание.

- Если в одном выражении записано несколько операций одинакового приоритета, унарные операции, условная операция и операции присваивания выполняются *справа налево*, остальные – *слева направо*. Например, $a = b = c$ означает $a = (b = c)$, а $a + b + c$ означает $(a + b) + c$.
- Для изменения порядка действий используются круглые скобки.
- При использовании стандартных функций аргумент обязательно заключается в круглые скобки.
- В языке C++ предусмотрены базовые математические операции (см. прил. 2).

Примеры записи выражений:

Математическое выражение	Запись на C++
1. $10x + 3\sqrt{\cos x}$	1. <code>10 * x + 3 * sqrtf(cos(x))</code>
2. $\frac{\sin 2x - 1}{x^2 - 1}$	2. <code>(sin(2*x) - 1) / (pow(x, 2) - 1)</code>
3. $ x + 2 \operatorname{ctg} x$	3. <code>abs(x) + 2 * cos(x) / sin(x)</code>

Практическая часть

ВЫПОЛНЕНИЕ РАБОТЫ: записать выражения по правилам языка C++.

Задание I

1. $\sin 5x + 2 \operatorname{tg} x^2$	5. $\frac{2xy + 4x^2}{5y}$;
2. $2x + 3y^2 - \sin 3x$	6. $4\sqrt{\cos 3b - 2\sin b}$;
3. $\sqrt{\frac{\sin y^2}{2\cos 2y}} + 2 \operatorname{tgy} - 25 $	7. $-2\cos x^2 + 2y^7$;
4. $\sqrt{1 - \frac{2}{x^2}} + x^5 - 3x $	8. $\frac{ 2a + \operatorname{tg} 3a }{\sqrt{1 + b^8} + 11}$

Задание II

Выполнить согласно варианту:

1.	$a = \frac{2 \cos\left(x - \frac{\pi}{6}\right)}{0.5 + \sin^2 y},$	$b = 1 + \frac{z^2}{3 + \frac{z^2}{5}}.$
----	--	--

2.	$s = \left x^{\frac{y}{x}} - 3\sqrt[3]{\frac{y}{x}} \right + \ln x,$	$t = (y - x) \frac{y - z(y - x)}{1 + (y - x)^2}.$
3.	$s = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!},$	$t = x(\sin x^3 + \cos^2 y).$
4.	$y = e^{-bt} \sin(at + b) - \sqrt{ bt + a },$	$s = b \sin(at^2 \cos 2t) - 1.$
5.	$w = \sqrt{x^2 + b} - \frac{b^2 \sin^3(x + a)}{x},$	$y = \cos^2 x^3 - \frac{x}{\sqrt{a^2 + b^2}}.$
6.	$s = x^3 \cdot tg^2(x + b)^2 + \frac{\alpha}{\sqrt{x + b}},$	$d = \frac{bx^2 - a}{e^{ax} - 1}.$
7.	$z = \frac{x^2(x + 1)}{b} - \sin^2(x + a),$	$s = \sqrt{\frac{xb}{a}} - \cos^2(x + b)^3.$
8.	$y = \sin^3(x^2 + a)^2 - \sqrt{\frac{x}{b}},$	$z = \frac{x^2}{a} + \cos(x + b)^3.$
9.	$f = \sqrt[3]{m \cdot tg t + c \cdot \sin t },$	$z = m \cos(bt \cdot \sin t) + c.$
10.	$y = b tg^2 x - \frac{a}{\sin\left(\frac{x}{a}\right)},$	$d = a \cdot e^{-\sqrt{a}} \cdot \cos\left(\frac{bx}{a}\right).$
11.	$f = \ln(a + x^2) + \sin^2\left(\frac{x}{b}\right),$	$z = e^{-cx} \cdot \frac{x + \sqrt{x + a}}{x - \sqrt{ x - b }}.$
12.	$y = \frac{a^{2x} + b^{-x} \cdot \cos(a + b)x}{x + 1},$	$k = e^{\frac{ax}{\sqrt{2}}} \cdot \cos \sqrt{\frac{bx}{2}}.$
13.	$z = \sqrt{ax \cdot \sin 2x + e^{-2x} \cdot (x + b)},$	$w = e^3 \cdot \sin bx - \frac{x^3}{a}.$
14.	$u = \frac{a^2 x + e^{-x} \cdot \cos bx}{bx - e^{-x} \cdot \sin bx + 1},$	$f = e^{2x} \cdot \ln(a + x) - b^{3x} \cdot \ln(b - x).$

15.	$z = \frac{\sin x}{\sqrt{1 + m^2 \sin^2 x}} - cm \cdot \ln(mx),$	$s = e^{-ax} \cdot \sqrt{x+1} + e^{-bx} \cdot \sqrt{x+1.5}.$
16.	$a = e^{-mx} \cdot \frac{\sqrt{m+x^2}}{ m^2 - x^2 },$	$b = \ln(c^2 + \sqrt{c+y}) + e^{-cy}.$
17.	$z = x^3 \cdot \sin^2(x+b) + \frac{x}{(x+b^2)},$	$s = (m^2 + y^2) \cdot e^{my} + \sqrt{3m + \ln(m+4y)}.$
18.	$l = \frac{a \cos^3 bx}{\sin a + bx \cos b - e^a},$	$w = \frac{\sin(x^3 - a) + b x }{\cos(ax + b)}.$
19.	$y = \cos^2(x + a^2)^2 + \sqrt{\frac{x}{b^2}},$	$z = \frac{a^3}{3x^2} + e^{(x+b)^3}.$
20.	$s = \left x^{\frac{x}{y}} - 5\sqrt{\frac{x}{y}} \right ,$	$t = (x-y) \frac{\sin(x+y) - \operatorname{tg}(y-x) \cdot z}{1 + (x-y)^2}$
21.	$s = \frac{e^{my}}{2+5x} + \sqrt{\frac{\ln(m-3)}{ 2a }},$	$v = \sqrt{ bt-a } - e^{\sqrt{bt}} \cdot \sin(bt+a).$
22.	$y = 1 + \frac{b \sin(at^2 \cos 3t)}{12x},$	$f = \sqrt[3]{m \cdot \operatorname{tg} t + c \sin 3t }$
23.	$y = a \cdot \operatorname{tg}^2(x+a) - \frac{b}{\sin^2\left(\frac{a}{x}\right)},$	$d = b \cdot e^{-\sqrt{b}} \cdot \cos\left(\frac{ax}{b} + 1.4\right).$
24.	$z = \sqrt{\frac{x^2}{b}} - \ln(a^2 + x^2),$	$f = e^{-cx} \cdot \frac{x + \sqrt{ x-b }}{x - \sqrt{x+a}}.$
25.	$s = \frac{(m^2 + y^2)}{5a} + \sqrt{ m + \operatorname{tg}(m-2y)},$	$f = \frac{e^{2x}}{6 \cdot \ln(a+x)} - b^{3x} \cdot \sin(b-x).$

Контрольные вопросы

1. Что такое алгоритм?
2. Перечислите свойства алгоритмов.

3. Способы записи алгоритма.
4. Перечислите базовые алгоритмические конструкции. Какова блок-схема следования?
5. Какова блок-схема полной (неполной) формы команды ветвления?
6. Какова блок-схема цикла с предусловием?
7. Какова блок-схема цикла с постусловием?
8. Какова блок-схема цикла с параметром?
9. Что такое программа?
10. Что такое транслятор?
11. Что такое компилятор?
12. Что такое интерпретатор?
13. Что такое оператор и операнд?
14. В каких случаях при записи математических выражений на языке C++ используются круглые скобки?
15. Как используются стандартные математические функции в C++?

СПИСОК ЛИТЕРАТУРЫ (Материалы для подготовки к занятию)

1. Конспект лекций
2. [2] стр. 13-17.

Лабораторная работа № 2

Программирование алгоритмов линейной структуры

ЦЕЛЬ РАБОТЫ: закрепление знаний о типах данных, преобразованиях типов, приобретение навыков составления и отладки программ линейной структуры, написанных на языке C++.

Формируемые компетенции или их части: вырабатывается навык разработки математических моделей, а также формируется способность к разработке и применению алгоритмических и программных решений (линейной структуры) в области прикладного программирования (формируется часть компетенций ОПК-3 и ПК-7).

Теоретическая часть

Базовые средства языка программирования C++

2.1. Общая характеристика языка программирования C++

C++ – компилируемый статически типизированный язык программирования общего назначения.

Поддерживает такие парадигмы программирования как процедурное программирование, объектно-ориентированное программирование, обобщённое программирование, обеспечивает модульность, отдельную компиляцию, обработку исключений, абстракцию данных, объявление типов (классов) объектов, виртуальные функции. Стандартная библиотека включает, в том числе, общеупотребительные контейнеры и алгоритмы. C++ сочетает свойства как высокоуровневых, так и низкоуровневых языков. В сравнении с его предшественником – языком C, – наибольшее внимание уделено поддержке объектно-ориентированного и обобщённого программирования.

C++ широко используется для разработки программного обеспечения, являясь одним из самых популярных языков программирования. Область его применения включает создание операционных систем, разнообразных прикладных программ, драйверов устройств, приложений для встраиваемых систем, высокопроизводительных серверов, а также развлекательных приложений (игр). Существует множество реализаций языка C++, как бесплатных, так и коммерческих и для различных платформ. Например, на платформе x86 это GCC, Visual C++, Intel C++ Compiler, Embarcadero (Borland) C++ Builder и другие. C++ оказал огромное влияние на другие языки программирования, в первую очередь на Java и C#.

Синтаксис C++ унаследован от языка C. Одним из принципов разработки было сохранение совместимости с C. Тем не менее, C++ не является в строгом смысле надмножеством C; множество программ, которые могут одинаково успешно транслироваться как компиляторами C, так и компиляторами C++, довольно велико, но не включает все возможные программы на C [1].

2.2. Основные понятия языка

Основу любого языка программирования образуют три, его составляющие: алфавит, синтаксис и семантика.

Алфавит - это фиксированный для данного языка программирования набор основных символов (букв алфавита), из которых состоит любой текст на этом языке. Никакие другие символы в тексте, не допускаются. Все символы в тексте кодируются с использованием стандартной кодовой таблицы ASCII. Каждый символ кода ASCII имеет неотрицательный порядковый номер, т.е. множество всех символов является упорядоченным.

Алфавит C++ включает:

1. Прописные и строчные латинские буквы и знак подчеркивания.
2. Арабские цифры от 0 до 9.
3. Специальные знаки:
“ { , | [] () + - / % * . \ ` : ? < = > ! & # ~ ; ^
4. Пробельные символы: пробел, символы табуляции, символы перехода на новую строку.

Синтаксис – это система правил, определяющих допустимые конструкции букв алфавита. С помощью этих конструкций, представляются отдельные компоненты алгоритма и алгоритм в целом, записанные на данном языке программирования. Таким образом, для каждой последовательности символов алфавита синтаксис позволяет ответить, на вопрос, является ли она текстом на данном языке программирования или нет.

Синтаксис C++ устанавливает, как можно на этом языке сформировать корректный текст и писать программы. Синтаксически правильная программа компилируется без ошибок.

Из символов алфавита формируются *лексемы* языка:

- **идентификаторы;**
- **ключевые (зарезервированные) слова;**
- **знаки операций;**
- **константы;**
- **разделители (скобки, точка, запятая, пробельные символы).**

Лексема, или элементарная конструкция - минимальная единица языка, имеющая самостоятельный смысл. Границы лексем определяются другими лексемами, такими, как разделители или знаки операций.

Одним из важнейших синтаксических понятий любого языка программирования является понятие **идентификатора**.

Идентификатор – *имя программного объекта*.

В идентификаторе могут использоваться латинские буквы, цифры и знак подчеркивания. Прописные и строчные буквы различаются, например, alpha, Alpha и ALPHA – три различных имени. Первым символом идентификатора может быть буква или знак подчеркивания, но не цифра. Пробелы внутри имен не допускаются.

Для улучшения читаемости программы следует давать объектам осмысленные имена. Существует соглашение о правилах создания имен, называемое вен-

герской нотацией (поскольку предложил ее сотрудник компании Microsoft венгр по национальности), по которому каждое слово, составляющее идентификатор, начинается с прописной буквы, а вначале ставится префикс, соответствующий типу величины, например, `lpfnSetFirstDialog`. Другая традиция – разделять слова, составляющие имя, знаками подчеркивания: `max_length`, `number_of_galosh` [2].

Длина идентификатора по стандарту не ограничена, но некоторые компиляторы и компоновщики налагают на нее ограничения. Идентификатор создается на этапе объявления переменной, функции, типа, и т. п., после этого его можно использовать в последующих операторах программы. При выборе идентификатора необходимо иметь в виду следующее:

- идентификатор не должен совпадать с ключевыми словами и именами используемых стандартных объектов языка;
- не рекомендуется начинать идентификаторы с символа подчеркивания, поскольку они могут совпасть с именами системных функций или переменных, и, кроме того, это снижает мобильность программы;
- на идентификаторы, используемые для определения внешних переменных, налагаются ограничения компоновщика.

Примеры правильных идентификаторов: `value25`, `My_File`, `index`, `Operation`. Примеры синтаксически неправильных идентификаторов:

`1_Side` (имя не должно начинаться с цифры),
`Том 2` (не допускается пробел),
`New-Name` (не допускается дефис),
`break` (служебное слово),
`exp` (предопределенное имя функции-экспоненты).

Заметим, что имена, константы, служебные слова нельзя писать слитно друг с другом, они должны отделяться один от другого пробелом, переводом строки или комментарием.

Служебные (зарезервированные, ключевые) **слова** – это идентификаторы, имеющие специальное значение для компилятора. Их можно использовать только в том смысле, в котором они определены. Например, `const` – ключевое слово, обозначающее невозможность модификации значения, `int` – ключевое слово, используемое для обозначения целого типа данных.

Знаки операций. Знак операции – это один или более символов, определяющих действие над операндами. Внутри знака операции пробелы не допускаются. Операции делятся на унарные, бинарные и тернарные по количеству участвующих в них операндов. Один и тот же знак может интерпретироваться по-разному в зависимости от контекста. Все знаки операций за исключением `[ ]`, `( )` и `? :` представляют собой отдельные лексемы.

Семантика языка – это смысловое значение слов. В программировании – начальное смысловое значение операторов, основных конструкций языка, т.п. [1].

Пример. Первый код: `i=0; while(i<5){i++;}`
Второй код: `i=0; do{i++;}while(i<5);`

Логически эти два фрагмента кода выполняют одно и то же, результаты их работы идентичны. В то же время семантически это два разных цикла.

Таким образом,

- **синтаксис** – формальный набор правил, определяющий способ построения любых конструкций языка;
- **семантика** – множество правил, определяющих смысл синтаксических конструкций.

Комментарии – это текст, который игнорируется при компиляции, что бывает полезно программистам для пояснения тех или иных участков кода. Компилятор обрабатывает комментарии как пустое пространство. Комментарии можно использовать и для временного отключения части кода при отладке.

Комментарии в программе на C++ либо начинаются с двух символов «прямая косая черта» (//однострочный комментарий) и заканчивается символом перехода на новую строку, либо заключается между символами – скобками /*многострочный комментарий*/. Внутри комментария можно использовать любые допустимые на данном компьютере символы, а не только символы из алфавита языка C++, поскольку компилятор комментарии игнорирует. Вложенные комментарии – скобки стандартом не допускаются, хотя в некоторых компиляторах разрешены.

2.3. Данные и способы их организации

Данные – это информация, представленная в виде пригодном для обработки автоматическими средствами, в частности ЭВМ, при возможном участии человека.

В программах обрабатываемые данные фигурируют в качестве значений тех или иных программных объектов. Данные, которые зафиксированы в тексте и не изменяются в процессе выполнения программы, называются **константами**; остальные данные являются значениями объектов, называемых **переменными**, поскольку значения этих объектов могут изменяться в процессе выполнения программы.

Данные, с которыми работает программа, хранятся в оперативной памяти. Компилятору необходимо точно знать: сколько места они занимают, как именно закодированы и какие действия над ними можно выполнять. Все это задается при описании данных с помощью типа.

Тип данных однозначно определяет:

1. Формат представления данных в памяти компьютера.
2. Множество допустимых значений, принимаемое переменной или константой, принадлежащей к выбранному типу.
3. Множество допустимых действий применимых к этому типу (операции и функции).

Например, целые и вещественные числа, занимают одинаковый объем памяти (**long int** и **float**), но имеют разные диапазоны возможных значений.

Таблица 2 – Классификация типов данных C++

Базовые	Определяемые программистом	
Простые (скалярные)	Простые	Составные
Логический (bool) Целые (short int, int, long int) Вещественные (float, double, long double) Символьные (char, wchar_t) Пустой тип (void)	Перечисления Указатели Ссылки	Массивы Структуры Объединения Классы

Константы в языке Си++

Итак, константа представляет значение, которое не может быть изменено. Константы обладают типом, и тип определяется записью константы. Синтаксис языка выделяет пять типов констант: целые, действительные (вещественные), символьные, перечислимые, нулевой указатель.

Например:

```

const int k=1;                // целая именованная константа
const float Pi = 3.1415926, Eps = 0.1e-5; // действительные константы
const char a = 'a';          // символьная константа
enum DAY {SUNDAY, MONDAY, TUESDAY, WEDNESDAY,
          THURSDAY, FRIDAY, SATURDAY}; // перечисление
char *pc = nullptr;          // нулевой указатель

```

Порядок объявления и инициализации переменных

Переменная – это именованная область памяти, в которой хранятся данные определенного типа. Каждая переменная должна быть объявлена в тексте программы перед первым ее использованием. Начинаящим рекомендуется объявлять переменные в начале тела программы перед первым исполнимым оператором. При объявлении переменной указывается ее тип и имя:

```
имя_типа    имя_переменной;
```

При описании можно присвоить переменной начальное значение, это называется **инициализацией**.

Например:

```

int d = 5; // Целочисленная переменная d проинициализирована значением 5
float b;   // Объявлена вещественная переменная b
char s = '+'; /* Переменная s объявлена как символьная и проинициализирована
              значением '+' */

```

Можно одновременно объявить несколько переменных, тогда в списке имен они отделяются друг от друга запятой, например:

```

int      a, b, c;    // Целые со знаком
char     ch, sh;     // Однобайтовые символьные
long     l, m, k;    // Длинные целые [-2147483648 – 2147483647]
float    x, y, z;    // Вещественные
long double u,v,w;   // Длинные вещественные с двойной точностью

```

Итак, перед использованием переменной необходимо определить четыре характеристики:

- имя;
- тип;
- значение (не обязательно);
- область видимости.

Имя переменной (идентификатор) обозначает в тексте программы величину, изменяющую свое значение. Для каждой переменной в памяти компьютера выделяется некоторая область памяти, способная хранить значение. Имя позволяет осуществить доступ к области памяти, хранящей значение переменной.

Тип переменной определяет размер выделяемой памяти и способ хранения значения.

Значение переменной. Если переменная не проинициализирована начальным значением, то перед первым ее использованием, значение переменной будет не определено. Независимо от того было задано или нет начальное значение переменной (при ее описании или при вводе с клавиатуры, например) ее величина может изменяться в ходе выполнения программы.

Область видимости. Переменные в языке Си классифицируются по области видимости и делятся на две категории: локальные и глобальные.

Локальная переменная – это такая переменная, которая объявлена внутри какого-либо блока (под блоком в языке Си понимается последовательность операторов, заключенная в фигурные скобки). Память под эту переменную выделяется в момент выполнения оператора ее объявления. По завершении работы блока эта переменная разрушается, а занимаемая ей память высвобождается. То есть, время жизни такой переменной ограничено тем блоком, в котором она объявлена.

Глобальная переменная – это такая переменная, которая объявлена вне любого блока. Следовательно, время ее жизни – время жизни самой программы.

Еще одним отличием локальной переменной от глобальной является процедура их инициализации. Любая глобальная переменная при объявлении автоматически обнуляется, локальная – нет. Поэтому, если начальное значение локальной переменной при объявлении еще не известно, лучше ее принудительно обнулить.

Резюмируя все вышесказанное, можно сделать вывод, что прежде чем описывать какие-либо данные, необходимо выполнить следующие действия:

- выбрать имя переменной (описать идентификатор);
- определить исходя из необходимого диапазона представления чисел тип переменной;
- определить область видимости переменной;
- обнулить или проинициализировать переменную начальным значением.

2.4. Стандартные простые типы данных

В любом алгоритмическом языке каждая константа, переменная, результат вычисления выражения или функции должны иметь определенный тип. Все типы языка C++ можно разделить на основные и составные. В языке C++ определено шесть основных типов данных для представления целых, вещественных, сим-

вольных и логических величин. На основе этих типов программист может вводить описание составных типов.

Основные (стандартные) типы данных часто называют *арифметическими*, поскольку их можно использовать в арифметических операциях. Для описания основных типов определены следующие ключевые слова:

- **int** (целый);
- **char** (символьный);
- **wchar_t** (расширенный символьный);
- **bool** (логический);
- **float** (вещественный);
- **double** (вещественный с двойной точностью).

Первые четыре типа называют целочисленными (*целыми*), последние два – *типами с плавающей точкой*. Код, который формирует компилятор для обработки целых величин, отличается от кода для величин с плавающей точкой.

Существует четыре спецификатора типа, уточняющих внутреннее представление и диапазон значений стандартных типов:

- **short** (короткий);
- **signed** (знаковый);
- **long** (длинный);
- **unsigned** (беззнаковый).

По умолчанию все целочисленные типы считаются знаковыми, то есть спецификатор **signed** можно опускать.

Базовыми в системе типов языка C++ являются простые типы, представленные в следующей таблице:

Таблица 3 – Стандартные простые типы данных

Простые типы	Тип	Название	Размер, байт	Диапазон
Логический	bool	булевское (логическое)	1	false, true (0, 1)
Символьный	signed char	символ	1	-128 ... 127
	unsigned char	беззнаковый символьный	1	набор допустимых символов с кодами 0 ... 255
Целые	short int	короткое целое (short)	2	-32 768 ... 32 767 ($-2^{15} .. 2^{15} - 1$)
	unsigned short int	беззнаковое короткое целое (слово)	2	0 ... 65 535 ($0 .. 2^{16} - 1$)
	long int	длинное целое (long)	4	-2 147 483 648 ... 2 147 483 647
	unsigned long int	беззнаковое длинное целое	4	0 ... 4294967295
Вещественные	float	вещественный	4	3.4e-38 ... 3.4e+38
	double	двойной точности	8	1.7e-308 ... 1.7e+308
	long double	расширенный	10	3.4e-4932 ... 3.4e+4923

Для вещественных типов в таблице приведены абсолютные величины минимальных и максимальных значений.

В стандарте ANSI диапазоны значений для основных типов не задаются, определяются только соотношения между их размерами, например:

```
sizeof (float) <=sizeof(double) <=sizeof(long double)
sizeof (char)<=sizeof(short) <=sizeof(int)<=sizeof(long)
```

Производным типом (определяемым программистом) является такой тип, который должен быть описан пользователем перед употреблением. К ним относятся массивы, перечисления, функции, структуры, ссылки, объединения и классы.

Первичными в иерархии типов являются стандартные скалярные типы, к числу которых относятся целые, вещественные, символьные, булевские типы и их модификации, учитывающие архитектурные особенности аппаратуры.

1. Булевский (логический) тип обозначается стандартным идентификатором **bool**. В C++ имеются две константы булевской типа, которые обозначаются также посредством стандартных идентификаторов: **true** (истина) и **false** (ложь). Внутренняя форма представления значения **false**-0 (нуль). Любое другое значение интерпретируется как **true**. При преобразовании к целому типу **true** имеет значение 1. Над значениями булевского типа допустимы операции сравнения, причем, считается, что **false** < **true**. Кроме того, имеются четыре, стандартные логические операции, обозначаемые символами **&&** (конъюнкция), **||** (дизъюнкция) и **!** (не – логическое отрицание). По приоритету эти операции имеют последовательность **! - && - ||**. Значения булевского типа занимают один байт памяти.

2. Символьный тип имеет стандартный идентификатор **char**. Значениями переменных этого типа являются символы из множества ASCII. Это множество состоит 256 различных символов, упорядоченных определенным образом, и содержит символы заглавных и строчных букв, цифр и различных других символов, включая специальные управляющие символы. Если символьное значение имеет графическое изображение, оно представляется соответствующим знаком, заключенным в апострофы, например: **'*'**, **'1'**, **'.'**. Тип **char**, как другие целые типы, может быть со знаком или без знака. Величины типа **char** применяются также для хранения целых чисел, не превышающих границы указанных диапазонов (см. табл. 3).

Расширенный символьный тип (wchar_t) предназначен для работы с набором символов, для кодировки которых недостаточно 1 байта, например, Unicode. Размер этого типа зависит от реализации; как правило, он соответствует типу **short**. Строковые константы типа **wchar_t** записываются с префиксом **L**, например, **L"Gates"**.

3. Целые типы – эта группа типов обозначает множества целых чисел в различных диапазонах. В C++ имеется пять целых типов, различающихся диапазоном значений и объемом занимаемой оперативной памяти. Целые типы обозначаются идентификаторами **int**, **short int**, **long int**, **unsigned short int**, **unsigned long int**: их характеристики приведены в таблице 3. Размер типа **int** не определяется стандартом, а зависит от компьютера, компилятора и операционной системы. Для 16-разрядного процессора под величины этого типа отводится 2 байта, для 32-разрядного – 4 байта. Поэтому при написании программ, переносимых на различные платформы, нельзя делать предположений о размере типа **int**. Для его получения необходимо пользоваться операцией **sizeof**, результатом которой является размер типа в байтах.

Число относится к целому типу тогда и только тогда, когда в его записи не содержится ни десятичной точки, ни признака порядка *e*.

Константам, встречающимся в программе, приписывается тот или иной тип в соответствии с их видом. Если этот тип по каким-либо причинам не устраивает программиста, он может явно указать требуемый тип с помощью суффиксов *L*, *l* (*long*) и *U*, *u* (*unsigned*). Например, константа *32L* будет иметь тип *long* и занимать 4 байта. Можно использовать суффиксы *L* и *U* одновременно, например, *0x22UL* или *05Lu*.

Над целыми значениями допустимы четыре арифметических действия, имеющие привычный смысл и обозначаемые соответственно символами: *+* – сложение, *-* – вычитание, *** – умножение, */* – целая часть от деления. Кроме того, в C++ можно использовать еще унарные операции (*++*, *--*, унарный минус и плюс, ...), операцию "типа деления": *%* – нахождение остатка от деления, операции присваивания (*=*, **=*, */=*, *%=*, *+=*, *-=*). Все приведенные операции при целых операциях дают целый результат. Также над целочисленными данными допустимо выполнять битовые, логические операции и операции сравнения.

4. Вещественные типы - эта группа типов обозначает множества вещественных значений в различных диапазонах. C++ поддерживает три вещественных типа, которые обозначаются стандартными идентификаторами ***float***, ***double***, ***long double*** (см. табл. 3). Вещественные числа могут изображаться в двух формах: в форме с фиксированной точкой (целая и дробная части отделяются друг от друга точкой, например, *-19.324*, *0.1281*) и в форме с плавающей точкой (*<мантисса>e<порядок>*, например, *16.2e21*, *2e-48*).

Константы с плавающей точкой имеют по умолчанию тип *double*. Можно явно указать тип константы с помощью суффиксов *F*, *f* (*float*) и *L*, *l* (*long*). Например, константа *2E+6L* будет иметь тип *long double*, а константа *1.82f* - тип *float*.

Над числами вещественных типов допустимы четыре арифметические операции: сложение, вычитание, умножение и деление; операции присваивания и сравнения.

Для данных вещественных типов, как и для данных целых типов, возможна ситуация, когда результат арифметических действий выходит за пределы допустимого диапазона. В этом случае компьютер выдаст результат, преобразованный к типу, описанному для данного результата и не имеющий ничего общего с истинным значением.

5. Тип *void*. Кроме перечисленных, к основным типам языка относится тип ***void***, но множество значений этого типа пусто. Он используется для определения функций, которые не возвращают значения, для указания пустого списка аргументов функции, как базовый тип для указателей и в операции приведения типов.

2.5. Структура программы

Программа на языке C++ состоит из *функций, описаний и директив препроцессора*. Одна из функций должна иметь имя *main*. Выполнение программы начинается с первого оператора этой функции.

Формат простейшего определения функции:

```
тип_возвращаемого_значения  имя_функции ([параметры]) // заголовок функции
{
// Блок тела функции:
// определения объектов;
// исполняемые операторы;
// return выражение;           // если функция возвращает значение.
}
```

Как правило, функция используется для вычисления какого-либо значения, поэтому перед именем функции указывается его тип (тип возвращаемого результата). Если функция не должна возвращать значение, указывается тип **void**. Тело функции является блоком и, следовательно, заключается в фигурные скобки. Любая функция, кроме `main`, вызывается из другой функции. Примером функций являются библиотечные функции, например, `sin(x)`, `fabs(a)`.

Пример структуры простейшей программы	Текст программы, состоящей из одной функции и содержащей компоненты, которые почти все могут отсутствовать
1. Препроцессорные директивы	<code>#include <iostream> /* подключение директивы (заголовочного файла) для организации ввода-вывода в C++ */</code>
2. Описания	<code>using namespace std; // описание пространства имен std</code>
3. Определение главной функции	<code>int main() // заголовок главной функции { // Блок тела функции // здесь находится программный код }</code>

Блок (составной оператор) – это произвольная последовательность определений и операторов, заключенная в фигурные скобки. Блок используется для укрупнения структуры программы. Точка с запятой в конце блока не ставится.

Текст программы на C++ обладает структурой. Общие принципы, позволяющие написать синтаксически правильную программу, таковы:

1. **Каждый оператор заканчивается знаком «;».** Обычная ошибка начинающего – это знак «;», завершающий заголовки функций или операторов цикла. В первом случае синтаксическая ошибка распознается как отсутствие тела функции, во втором случае телом цикла является пустой оператор, что синтаксической ошибкой не является, и программа выполняется.
2. Прежде чем использовать функции, переменные, типы данных, следует объявить их или подключить файлы с их объявлениями (сделать известными компилятору). В C++ объявление переменной (объекта) возможно не только в начале программы, но и в любом месте текста до первого обращения к ней. Областью действия такого объекта является только непосредственно охватывающий его блок. Как правило, так объявляют рабочие переменные.
3. Чтобы вызов функции мог быть выполнен, функция должна быть определена, т. е. описаны действия, которые она осуществляет.

4. При определении функции полезно придерживаться следующих рекомендаций:
 - объявить и проинициализировать входные и выходные переменные;
 - промежуточные переменные объявлять и инициализировать в начале блоков или перед операторами, в которых они используются.
5. В любой последовательности действий нужно стремиться к триаде: инициализация (ввод), обработка, возвращение значения (вывод).

Пример структуры программы, содержащей 2 функции: `main` и `mysum`.

1	Зона комментариев	<code>// моя первая программа</code>
2	Зона директив препроцессора	<code>#include <iostream.h></code> <code>#include <conio.h></code>
3	Зона объявления прототипов пользовательских функций и глобальных переменных и структур данных	<code>int a = 10;</code> <code>int mysum(int x);</code>
4	ТОЧКА ВХОДА В ПРОГРАММУ Зона функции <code>main()</code>	<code>int main()</code> { <code>int b = 15, c;</code> <code>c = mysum(b);</code> <code>cout<<c;</code> <code>_getch();</code> }
5	Зона определения пользовательских функций	<code>int mysum(int x)</code> { <code>int z = x + a;</code> <code>return z;</code> }

При анализе приведенной программы можно выделить следующие крупные части типового исходного модуля:

- *подключение* заголовочных файлов библиотек;
- *объявления* пользовательских символических констант, вспомогательных функций, классов и типов данных, глобальных переменных;
- *заголовок главной функции*;
- *определение главной функции* – блок, содержащий:
 - *объявление* локальных переменных и констант и их инициализацию (присвоение начальных значений);
 - *ввод* исходных данных (диалог с пользователем);
 - *обработка* – обращение к функциям, вычисление выражений, выполнение операторов;
 - *вывод* результата;
 - *возвращение* кода завершения главной функции;
- *определения* пользовательских вспомогательных функций и методов классов; для каждой функции после заголовка размещается блок, содержащий:
- *объявление* локальных переменных и их инициализацию;

- *обработку* формальных параметров с использованием локальных переменных;
- *возвращение* результата.

Подобное структурирование текста исходного модуля, который размещается в *одном* файле с расширением *crr*, не единственно возможное. Но следование данному образцу облегчает понимание и отладку программ.

Правила оформления текста программы, направленные на облегчение понимания смысла и повышение наглядности, таковы:

- разделять логические части программы пустыми строками;
- разделять операнды и операции пробелами;
- для каждой фигурной скобки отводить отдельную строку;
- в каждой строке должно быть, как правило, не более одного оператора;
- ограничивать длину строки 60-70 символами;
- отступами слева отражать вложенность операторов и блоков;
- длинные операторы располагать в нескольких строках;
- проводить алгоритмизацию так, чтобы определение одной функции занимало, как правило, не более одного экрана текста;
- стремиться использовать типовые заготовки фрагментов программ, включая и типовую структуру блока и программы в целом.

Конечно, эти правила нужно использовать совместно с *правилами именования* элементов программы и *комментирования* текста программы:

- **комментарии в тексте необходимы;**
- **имена объектов программы выбираются осмысленно.** Каждое имя подчеркивает назначение и логику объекта, например, имена библиотечных функций *sin*, *abs*, *printf* и прочих говорят сами за себя. Имена объектов, введенные программистом, подчеркивают их абстрактный смысл, например, *Count*, *Square*, *Point.x*, *Point.y* и так далее.

2.6. Операторы

Оператор – это основной элемент языка программирования, представляющий собой законченную фразу и определяющий некоторый логически завершённый этап обработки данных.

Операторы условно можно подразделить на две категории: *исполняемые* – с их помощью реализуется алгоритм решаемой задачи, и *описательные*, необходимые для определения типов пользователя и объявления объектов программы, например, переменных.

Среди исполняемых операторов можно выделить:

- простые операторы (оператор-выражение, операторы передачи управления);
- составные операторы (составной оператор, блок)
- структурированные операторы (условный, выбора, цикла).

Рассмотрим **простые операторы**

Оператор «выражение»

Любое выражение, завершающееся точкой с запятой, рассматривается как оператор, выполнение которого заключается в вычислении выражения. Частным случаем выражения является *пустой оператор* (;). Он используется, когда по синтаксису оператор требуется, а по смыслу нет.

Примеры:

```
x ++;           // выражение – оператор; выполняется операция инкремента
c = a * 1.5;     // выражение с операцией присваивания
a *= b + c;      // выполняется умножение с присваиванием: a = a * (b + c)
fun(i, k);       // выполняется вызов функции
```

Оператора присваивания в языке C++ нет, его заменяет выражение, в составе которого есть **операция присваивания**.

Общий вид:

идентификатор = выражение;

Пример: `x = y + 4; // выражение с операцией присваивания`

Выполняется в два этапа:

- а) вычисляется значение выражения, стоящего в правой части от знака присваивания;
- б) вычисленное значение присваивается переменной, стоящей в левой части от знака присваивания, при этом предыдущее значение этой переменной теряется.

Рекомендуется строго относиться к типам данных, не смешивать типы в выражениях, следить, чтобы тип левого операнда присваивания соответствовал типу выражения правой части.

При выполнении присваивания, если типы левой и правой части не совпадают, происходит неявное преобразование. C++ всегда пытается это сделать, и упрощенно можно считать, что преобразование происходит без потери данных от меньшего типа к большему, например от `int` к `float` или от `char` к `int`, и с потерей данных от большего типа к меньшему, например, от `float` к `int`. Это легко понять, если вспомнить, что тип данного, это объем занятой им памяти.

Явное преобразование типов выполняется при присваивании вида:

имя = (тип) выражение;

Например:

```
int      a = 5, b = 2;
float    c;
c = a/b;
```

Хочется думать, что значение `c` будет равно 2.5, ведь оно вещественное, но порядок операций таков, что деление старше присваивания, и оно выполняется с операндами целого типа, и его результат равен 2, то есть приведение типа будет выполнено только при присваивании.

Для явного преобразования типов используется известный прием:

```
c = (float) a/(float) b;    // c = 2.5
```

Операция присваивания правоассоциативна, т.е. вычисления выполняются справа налево, поэтому допускается запись цепочек присваиваний, например:

```
x = y = z = 1;    // Каждая переменная будет равна 1.
```

В *сложных операциях присваивания* (+=, *=, /= и т.п.) при вычислении выражения, стоящего в правой части, используется и L-значение из левой части. Например, при сложении с присваиванием ко второму операнду прибавляется первый, и результат записывается в первый операнд, то есть выражение `a += b` является более компактной записью выражения `a = a + b`.

Еще один вид операторов, который можно отнести к операторам-выражениям, это *оператор обращения к функции* (вызов функции). Его операндам являются параметры функции.

Формат оператора:

имя функции ([список фактических параметров] – может отсутствовать);

Например:

```
scanf ("%d%f", &my_int, &my_float);           // ВВОД ДАННЫХ  
printf ("Целое = %d, Вещественное = %f\n", my_int, my_float) // ВЫВОД
```

Пробелы в строке текста являются значащими, то есть, если ввести целое 5 и дробное 9.9, то строка вывода будет иметь вид:

```
Целое = 5, Вещественное = 9.900000
```

Операторы передачи управления рассмотрим позже (п. 3.6, п. 4.1)

2.7. Организация ввода/вывода данных

Ввести данное – означает присвоить произвольное значение переменной во время выполнения программы. **Вывести данное** – означает напечатать на экране значение переменной при выполнении программы.

В C++ нет встроенных средств ввода/вывода – он осуществляется с помощью функций, типов и объектов, содержащихся в стандартных библиотеках. Используется два способа: функции, унаследованные из языка C, и объекты C++.

Организация ввода/вывода в стиле C

Простейший из способов ввода и вывода (обмена) данных – это форматированный, с определением правил размещения данных во входном – выходном потоке. Для реализации такого обмена необходима библиотека `stdio.h` (standart input output library), которая подключается к программе директивой `#include <stdio.h>`

Для ввода значения данного с клавиатуры (с эхоповтором на экране) используется функция `scanf`, синтаксис которой:

```
scanf ("форматная строка", список_ввода);
```

Здесь «список ввода» – имена переменных, значения которых будут введены с клавиатуры при выполнении функции `scanf`. Имена переменных предваряются символом `&`, который является признаком адресной операции, и означает, что введенное значение пересылается по адресу, определенному именем переменной. При вводе данные отделяются пробелами, или Enter'ом.

Для вывода значения данного на экран используется функция `printf`, синтаксис которой:

```
printf ("форматная строка", список_вывода);
```

Здесь «список вывода» – список имен переменных и выражений (в том числе констант), значения которых появятся на экране при выполнении функции `printf`.

Форматная (управляющая) строка, – это строка символов внутри двойных кавычек, содержащая управляющие символы и текст. При вводе данных функция `scanf` читает посимвольно текст из входного потока, распознает лексемы и преобразует их в машинное представление в соответствии с признаком формата, сопоставленного переменной, ожидающей данное. При выводе функция `printf` берет машинное представление значения переменной, и соответственно признаку формата, преобразует в текстовое представление, и выводит на экран.

Число управляющих символов равно числу объектов в списке ввода-вывода. Управляющий символ имеет признак `%`, и одно из следующих значений [3]:

<code>%d</code>	– ввод - вывод целого десятичного числа	<code>(int);</code>
<code>%u</code>	– ввод - вывод целого без знака	<code>(unsigned);</code>
<code>%f</code>	– ввод - вывод числа с плавающей точкой	<code>(float и double);</code>
<code>%e</code>	– ввод - вывод числа в экспоненциальной форме	<code>(double и float);</code>
<code>%c</code>	– ввод - вывод символа	<code>(char);</code>
<code>%l</code>	– ввод - вывод длинного значения	<code>(long);</code>

и другие.

При вводе и выводе необходимо строгое соответствие типа вводимого данного управляющему символу формата.

Пример форматированного ввода и вывода.

```
#include <stdio.h>

void main(void)
{
    int      my_int;
    float    my_float;
    printf("\nВведите целое и дробное число\n");
    scanf ("%d", &my_int);
    scanf ("%f", &my_float);
    printf ("%d %f", my_int, my_float);
}
```

При запуске программы она выведет на экран строку – приглашение ко вводу данных, затем при выполнении каждого `scanf` будет ожидать ввода данных. Пользователь должен ввести требуемое количество данных, отделяя их друг от друга пробелами или нажатием клавиши `Enter`. При завершении ввода данные тут же будут выведены на экран самым примитивным образом. Так, если ввести целое 5 и дробное 9.9, то строка вывода будет иметь вид:

```
5 9.900000
```

Поскольку при вводе данного функция `scanf` находится в состоянии ожидания ввода, рекомендуется каждый ввод предварять строкой, выводящей на экран приглашение для ввода данного, в котором пользователю подробно объясняют,

что и как он должен сделать, чтобы правильно ввести данные. Этот простой прием существенно улучшит интерфейс любой программы.

При выводе данных для улучшения читабельности рекомендуется использовать некоторые приемы.

1. Управляющие символы потока, например:

\n для перевода строки при выводе;
\t для выполнения табуляции.

2. Произвольный текст в форматной строке для приглашения на ввод данного и для пояснений при выводе, например, функция вывода может быть записана так:

```
printf ("Целое = %d, Вещественное = %f\n", my_int, my_float);
```

Пробелы в строке текста являются значащими. Теперь, если ввести целое 5 и дробное 9.9, то строка вывода будет иметь вид:

```
Целое = 5, Вещественное = 9.900000
```

3. Модификаторы форматов. Они используются для оформления вывода. По умолчанию (без модификаторов) данные выводятся в поле минимальной ширины с точностью 6 знаков после запятой, число прижимается к правому краю поля. Выводом можно управлять.

а) Ширина поля, это строка цифр, определяющая наименьший размер поля вывода (позиционирование). Число, не входящее в поле, игнорируется.

б) Точность вывода, это две цифры, определяющие общий размер поля вывода и число знаков после запятой. Используется для вещественных чисел.

В примерах обозначим знаком _ пробелы, которые будут в строке вывода.

```
printf ("Целое = %4d, Вещественное = %5.2f\n", my_int, my_float);
```

Если ввести значения 10 и 2.3, то строка вывода будет иметь вид:

```
Целое = _10, Вещественное = _2.30
```

Если ввести значения 19951 и 12.9999, то строка вывода будет иметь вид:

```
Целое = 19951, Вещественное = 13.00
```

Можно сделать вывод, что целое число, несмотря на ограничения поля вывода 4 символами, выведется без изменений, а вещественное число будет округлено до сотых.

в) Знак минус используется для выравнивания числа влево внутри поля вывода.

```
printf ("Целое = %-4d, Вещественное = %-5.2f\n", my_int, my_float);
```

Если ввести значения 2 и 2.36666, то строка вывода будет иметь вид:

```
Целое = 2___, Вещественное = 2.37_
```

Если ввести значения -1999 и 12.9999, то строка вывода будет иметь вид:

```
Целое = -1999, Вещественное = 13.00
```

Пример использования форматированного ввода-вывода.

```

#include <stdio.h>

#define STR "Программа"           // Для иллюстрации вывода строк.

void main (void)
{
    // Вывод целого числа 336
    printf ("%d\n", 336); // 336
    printf ("%2d\n", 336); // 336, формат 2d игнорируется
    printf ("%8d\n", 336); // _____336 // ширина поля вывода = 8
    printf ("% -8d\n", 336); // 336_____ // прижато влево ширина поля 8
    printf("\n"); // Пропуск строки при выводе
    // Вывод вещественного числа 12.345
    printf ("%f\n", 12.345); // 12.345000
    printf ("%e\n", 12.345); // 1.234500e+01
    printf ("%10.1f\n", 12.345); // _____12.3
    printf ("% -12.1f\n", 12.345); // 12.3
    printf("\n");
    // Вывод строки символов по формату s
    printf ("%s\n", STR); // Программа
    printf ("%12s\n", STR); // _____Программа
    printf ("%12.5s\n", STR); // _____Прогр
    printf ("% -12.5s\n", STR); // Прогр
    printf("\n");
}

```

Организация ввода/вывода в стиле C++

Для управления вводом/выводом данных в стиле C++ применяется заголовочный файл <iostream.h>. В нем определены стандартные объекты-потoki cin для ввода с клавиатуры и cout для вывода на экран, а также операции помещения в поток « и чтения из потока ».

```

cin«список_ввода; // ввод значения данного с клавиатуры
cout«список_вывода; // вывода значения данного на экран

```

Пример программы, использующей функции ввода/вывода в стиле C:

```

#include <stdio.h>

int main()
{
    int i;
    printf("Введите целое число\n");
    scanf("%d",&i);
    printf("Вы ввели число %d, спасибо!",i);
    return 0;
}

```

А вот как выглядит та же программа с использованием библиотеки классов C++:

```

#include <iostream.h>

int main()
{
    int i;
    cout«"Введите целое число\n";
}

```

```

    cin>>i;
    cout<<"Вы ввели число "<i<<"", спасибо!";
    return 0;
}

```

Форматированный ввод/вывод в C++

Возможность управлять вводом-выводом в C++, обеспечивают форматирующие функции-члены, флаги и манипуляторы. Флаги, функции и манипуляторы выполняют одну и ту же задачу – задают определённый формат ввода/вывода информации в потоках. Ввод/вывод на экран/с экрана в C++ осуществляется с помощью операторов `cin` и `cout` соответственно, а значит манипуляторы форматирования используются совместно с данными операторами ввода/вывода. Различие между функциями, флагами и манипуляторами форматирования состоит в способе их применения. Теперь рассмотрим способы применения объектов форматирования.

```

//Основные форматирующие функции-члены:
cout.fill('/*symbol*/*'); // устанавливает символ заполнитель
//где symbol – символ-заполнитель, символ передаётся в одинарных кавычках

cout.width(/*width_field*/); // задает ширину поля
//где width_field – количество позиций (одна позиция вмещает один символ)

cout.precision(/*number*/); //задает количество знаков после десятичной
// точки, где number – количество знаков после десятичной точки

```

Доступ к функциям осуществляется через операцию **точка**, а в круглых скобках передаётся аргумент. Аргумент функции `fill()` может передаваться в виде символа, обрамленного одинарными кавычками или в виде числа (код символа). Одних функций не достаточно для форматирования потоков ввода/вывода, поэтому в C++ предусмотрен ещё один способ форматирования – флаги.

Флаги форматирования позволяют включить или выключить один из параметров ввода/вывода. Чтобы установить флаг ввода/вывода, необходимо вызвать функцию `setf()`, если необходимо отключить флаг вывода, то используется функция `unsetf()`. Далее показаны конструкции установки и снятия флагов вывода.

```

// установка флага вывода
cout.setf( ios::/*name_flag*/ );
// где name_flag – это имя флага

```

Доступ к функциям оператора вывода выполняется через операцию **точка**. Метод `setf()` принимает один аргумент – имя флага. Флаги вывода объявлены в классе `ios`, поэтому, перед тем, как обратиться к флагу, необходимо написать имя класса – `ios`, после которого, с помощью операции разрешения области действия, вызвать нужный флаг.

```

// снятие флага вывода
cout.unsetf( ios::/*name_flag*/ );
// где name_flag – это имя флага

```

Если при вводе/выводе необходимо установить/снять несколько флагов, то можно воспользоваться поразрядной логической операцией ИЛИ |. В этом случае конструкция языка C++ будет такой:

```
// установка нескольких флагов
cout.setf( ios::*name_flag1*/ | ios::*name_flag2*/ | ios::*name_flag_n*/ );
// снятие нескольких флагов
cout.unsetf( ios::*name_flag1*/ | ios::*name_flag2*/ | ios::*name_flag_n*/ );
```

В таблице 4 описаны некоторые основные флаги форматирования, а также показаны примеры их использования.

Таблица 4 – Некоторые флаги форматирования в C++

Флаг	Назначение	Пример	Результат
scientific	Вывод чисел с плавающей точкой в экспоненциальной форме	<code>cout.setf(ios::scientific); double value = 1024.165; cout << value << endl;</code>	1.024165e+003
fixed	Вывод чисел с плавающей точкой в фиксированной форме (по умолчанию)	<code>double value = 1024.165; cout << value << endl;</code>	1024.165
right	Выравнивание по правой границе (по умолчанию). Сначала необходимо установить ширину поля (ширина поля должна быть заведомо большей чем, длинна выводимой строки).	<code>cout.width(40); cout << «cppstudio.com» << endl;</code>	__cppstudio.com
left	Выравнивание по левой границе. Сначала необходимо установить ширину поля(ширина поля должна быть заведомо большей чем, длинна выводимой строки).	<code>cout.setf(ios::left); cout.width(40); cout << «cppstudio.com» << endl;</code>	cppstudio.com__

Ещё один способ форматирования – форматирование с помощью манипуляторов. Манипулятор – объект особого типа, который управляет потоками ввода/вывода, для форматирования передаваемой в потоки информации. Отчасти манипуляторы дополняют функционал, для форматирования ввода/вывода. Но большинство манипуляторов выполняют точно, то же самое, что и функции с флагами форматирования. Есть случаи, когда проще пользоваться флагами или функциями форматирования, а иногда удобнее использовать манипуляторы форматирования. Именно по этому в C++ предусмотрено несколько средств форматирования ввода/вывода.

В таблице 5 показаны основные манипуляторы форматирования C++.

Таблица 5 – Некоторые манипуляторы форматирования в C++

Манипулятор	Назначение	Пример	Результат
<code>endl</code>	Переход на новую строку при выводе	<code>cout << «website:» << endl << «cppstudio.com»;</code>	<code>website: cppstudio.com</code>
<code>scientific</code>	Вывод чисел с плавающей точкой в экспоненциальной форме	<code>double value = 1024.165; cout << scientific << value << endl;</code>	<code>1.024165e+003</code>
<code>fixed</code>	Вывод чисел с плавающей точкой в фиксированной форме (по умолчанию).	<code>double value = 1024.165; cout << fixed << value << endl;</code>	<code>1024.165</code>
<code>setw(int number)</code>	Установить ширину поля, где number - количество позиций, символов (выравнивание по умолчанию по правой границе). Манипулятор с параметром.	<code>cout << setw(40) << «cppstudio.com» << endl;</code>	<code>__cppstudio.com</code>
<code>right</code>	Выравнивание по правой границе (по умолчанию). Сначала необходимо установить ширину поля(ширина поля должна быть заведомо большей чем, длинна выводимой строки).	<code>cout << setw(40) << right << «cppstudio.com» << endl;</code>	<code>__cppstudio.com</code>
<code>left</code>	Выравнивание по левой границе. Сначала необходимо установить ширину поля (ширина поля должна быть заведомо большей чем, длинна выводимой строки).	<code>cout << setw(40) << left << «cppstudio.com» << endl;</code>	<code>cppstudio.com__</code>
<code>setprecision(int count)</code>	Задаёт количество знаков после запятой, где count - количество знаков после десятичной точки	<code>cout << fixed << setprecision(3) << (13.5 / 2) << endl;</code>	<code>6.750</code>
<code>setfill(int symbol)</code>	Установить символ заполнитель. Если ширина поля больше, чем выводимая величина, то свободные места поля будут наполняться символом symbol - символ заполнитель	<code>cout << setfill('0') << setw(4) << 15 << ends << endl;</code>	<code>0015</code>

При использовании манипуляторов необходимо подключить заголовочный файл `<iomanip>`

Практическая часть

ВЫПОЛНЕНИЕ РАБОТЫ: в соответствии с вариантом составить и реализовать программы (задание I- III).

Задание I

Написать программу для расчета по двум формулам. Предварительно подготовить тестовые примеры для второй формулы с помощью калькулятора (результаты вычисления по обеим формулам должны совпадать). Список стандартных математических функций C++ приведен в приложении 2. Отсутствующие в языке функции выразить через имеющиеся.

Выполнить согласно варианту:

1.	$z_1 = 2 \sin^2(3\pi - 2\alpha) \cos^2(5\pi + 2\alpha);$	$z_2 = \frac{1}{4} - \frac{1}{4} \sin\left(\frac{5}{2}\pi - 8\alpha\right).$
2.	$z_1 = \cos \alpha + \sin \alpha + \cos 3\alpha + \sin 3\alpha;$	$z_2 = 2\sqrt{2} \cos \alpha \cdot \sin\left(\frac{\pi}{4} + 2\alpha\right).$
3.	$z_1 = \frac{\sin 2\alpha + \sin 5\alpha - \sin 3\alpha}{\cos \alpha + 1 - 2 \sin^2 2\alpha};$	$z_2 = 2 \sin \alpha.$
4.	$z_1 = \frac{\sin \alpha + \sin 5\alpha - \sin 3\alpha}{\cos \alpha - \cos 3\alpha + \cos 5\alpha};$	$z_2 = \operatorname{tg} 3\alpha.$
5.	$z_1 = 1 - \frac{1}{4} \sin^2 2\alpha + \cos 2\alpha;$	$z_2 = \cos^2 \alpha + \cos^4 \alpha.$
6.	$z_1 = \cos \alpha + \cos 2\alpha + \cos 6\alpha + \cos 7\alpha;$	$z_2 = 4 \cos \frac{\alpha}{2} \cdot \cos \frac{5}{2}\alpha \cdot \cos 4\alpha.$
7.	$z_1 = \cos^2\left(\frac{3}{8}\pi - \frac{\alpha}{4}\right) - \cos^2\left(\frac{11}{8}\pi + \frac{\alpha}{4}\right);$	$z_2 = \frac{\sqrt{2}}{2} \sin \frac{\alpha}{2}.$
8.	$z_1 = \cos^4 x + \sin^2 y + \frac{1}{4} \sin^2 2x - 1;$	$z_2 = \sin(y + x) \cdot \sin(y - x).$
9.	$z_1 = (\cos \alpha - \cos \beta)^2 - (\sin \alpha - \sin \beta)^2;$	$z_2 = -4 \sin^2 \frac{\alpha - \beta}{2} \cdot \cos(\alpha + \beta).$
10.	$z_1 = \frac{\sin\left(\frac{\pi}{2} + 3\alpha\right)}{1 - \sin(3\alpha - \pi)};$	$z_2 = \operatorname{ctg}\left(\frac{5}{4}\pi + \frac{3}{2}\alpha\right).$
11.	$z_1 = \frac{1 - 2 \sin^2 \alpha}{1 + \sin 2\alpha};$	$z_2 = \frac{1 - \operatorname{tg} \alpha}{1 + \operatorname{tg} \alpha}.$

12. $z_1 = \frac{\sin 4\alpha}{1 + \cos 4\alpha} \cdot \frac{\cos 2\alpha}{1 + \cos 2\alpha};$	$z_2 = \operatorname{ctg}\left(\frac{3}{2}\pi - \alpha\right).$
13. $z_1 = \frac{\sin \alpha + \cos(2\beta - \alpha)}{\cos \alpha - \sin(2\beta - \alpha)};$	$z_2 = \frac{1 + \sin 2\beta}{\cos 2\beta}.$
14. $z_1 = \frac{\cos \alpha + \sin \alpha}{\cos \alpha - \sin \alpha};$	$z_2 = \operatorname{tg} 2\alpha + \sec 2\alpha. \left(\sec x = \frac{1}{\cos x}\right).$
15. $z_1 = \frac{\sqrt{2b + 2\sqrt{b^2 - 4}}}{\sqrt{b^2 - 4} + b + 2};$	$z_2 = \frac{1}{\sqrt{b + 2}}.$
16. $z_1 = \frac{x^2 + 2x - 3 + (x + 1)\sqrt{x^2 - 9}}{x^2 - 2x - 3 + (x - 1)\sqrt{x^2 - 9}};$	$z_2 = \sqrt{\frac{x + 3}{x - 3}}.$
17. $z_1 = \frac{\sqrt{(3m + 2)^2 - 24m}}{3\sqrt{m} - \frac{2}{\sqrt{m}}};$	$z_2 = \sqrt{m}. \quad (m > 0)$
18. $z_1 = -2 - \left(\frac{a + 2}{\sqrt{2a}} - \frac{a}{\sqrt{2a} + 2} + \frac{2}{a - \sqrt{2a}}\right) \cdot \frac{\sqrt{a} - \sqrt{2}}{a + 2};$	$z_2 = \frac{1}{\sqrt{a} - \sqrt{2}}. \quad (a > 0)$
19. $z_1 = \left(\frac{1 + a + a^2}{2a + a^2} + 2 - \frac{1 - a + a^2}{2a - a^2}\right)^{-1} (5 - 2a^2);$	$z_2 = \frac{4 - a^2}{2}.$
20. $z_1 = \frac{(m - 1)\sqrt{m} - (n - 1)\sqrt{n}}{\sqrt{m^3 n + nm + m^2 - m}};$	$z_2 = \frac{\sqrt{m} - \sqrt{n}}{m}.$
21. $z_1 = (1 + \operatorname{tg} \alpha)(1 + \operatorname{ctg} \alpha) - 2;$	$z_2 = \frac{1}{\sin \alpha \cos \alpha}.$
22. $z_1 = 1 + \operatorname{ctg}^2 \alpha + \frac{1}{\cos^2 \alpha};$	$z_2 = \frac{1}{\sin^2 \alpha \cos^2 \alpha}.$
23. $z_1 = 1 - \frac{1}{1 + \operatorname{ctg}^2 \alpha};$	$z_2 = \frac{1}{1 + \operatorname{tg}^2 \alpha}.$

24.	$z_1 = \frac{1 - 2 \cos^2 \alpha}{\sin \alpha \cos \alpha} + \operatorname{ctg} \alpha;$	$z_2 = \operatorname{tg} \alpha.$
25.	$z_1 = \sin^4 \alpha - 1 + 2 \sin^2 \alpha \cos^2 \alpha;$	$z_2 = -\cos^4 \alpha.$

Задание II

1. Составить программу вычисления объема конуса по заданному диаметру и образующей.
2. Дана сторона равностороннего треугольника. Составить программу нахождения площади этого треугольника и радиуса описанной окружности.
3. Известны радиусы двух concentрических окружностей. Составить программу нахождения площади кольца, образованного этими окружностями.
4. Составить программу нахождения суммы n членов арифметической прогрессии, для которой известен первый член, разность и число n .
5. Вычислить объем призмы, боковые грани которой - квадраты, а основанием служит равносторонний треугольник, вписанный в круг радиуса r .
6. Вычислить площадь прямоугольника, вписанного в окружность радиуса r , если отношение его сторон равно R .
7. Даны две стороны треугольника и угол между ними. Определить третью сторону и площадь.
8. Вычислить процент материала, ушедшего в отходы, если из куба с ребром a был выточен шар радиуса r ($r < a$).
9. Вычислить площадь кольца, ширина которого равна a , а отношение радиусов окружностей равно b .
10. Вычислить периметр и площадь прямоугольного треугольника, описанного около круга радиуса r , если гипотенуза треугольника равна c .
11. Вычислить диаметр трубы, пропускная способность которой позволяет заменить ею две трубы с диаметрами d_1 и d_2 .
12. Вычислить высоты треугольника со сторонами a , b и c .
13. Вычислить площадь правильного n -угольника, описанного около круга радиусом r .
14. Высота конуса равна h , а радиус основания r . Вычислить объем шара, вписанного в конус.
15. Вычислить массу свинцовой трубы, длина которой равна b м (плотность свинца равна $11,4 \text{ г/см}^3$), толщина стенок a мм, а внутренний диаметр трубы равен d мм.
16. Дан равносторонний треугольник. Вычислить сторону, высоту и площадь этого треугольника, если радиус вписанной окружности равен r .

17. Стальной вал, имеющий b мм длины и d мм в диаметре, обрабатывается на токарном станке, при этом диаметр уменьшается при обточке на S мм. Вычислить, на сколько уменьшается масса тела (плотность стали $7,4 \text{ г/см}^3$).
18. Вычислить объем призмы, боковые грани которой – квадраты. Основанием призмы служит равносторонний треугольник, заданный длиной стороны.
19. В конус с радиусом основания r и высотой h вписан цилиндр, радиус основания которого равен a . Вычислить объем цилиндра.
20. Даны две стороны треугольника d и c и угол между ними α . Найти радиусы окружностей: вписанной в треугольник и описанной около него.
21. Дана длина ребра куба. Найти объем куба и площадь его боковой поверхности.
22. Треугольник задан тремя сторонами. Вычислить его высоты.
23. Определить периметр правильного n -угольника, описанного около окружности радиуса R .
24. Смешано V_1 литров воды температуры T_1 и V_2 литрами воды температуры T_2 . Найти объем и температуру образовавшейся смеси.
25. Известна длина окружности. Найти площадь круга, ограниченного этой окружностью.
26. Даны гипотенуза и катет прямоугольного треугольника. Найти второй катет и радиус вписанной окружности.
27. Найти площадь кольца, внутренний радиус которого равен R_1 , а внешний - R_2 ($R_2 > R_1$).
28. Определить время, через которое встретятся два тела, равноускоренно движущиеся навстречу друг к другу, если известны их начальные скорости, ускорение, начальное расстояние между ними.
29. Вычислить расстояние между двумя точками с координатами (x_1, y_1) и (x_2, y_2) .
30. Даны катеты прямоугольного треугольника. Найти его гипотенузу и площадь.

Задание III

1. Радоновые ванны, применяемые для лечения, содержат $1,8 \cdot 10^6$ атомов радона в воде объемом $1,0 \text{ дм}^3$. На сколько молекул воды приходится один атом радона в лечебной ванне?
2. Сколько атомов ртути содержится в воздухе объемом $1,0 \text{ м}^3$ в помещении, зараженном ртутью, при температуре 20°C , если давление насыщенного пара ртути при этой температуре 133 мПа ?
3. Какова длина ребра куба, содержащего $1,0 \cdot 10^6$ молекул идеального газа при нормальных условиях?
4. При какой температуре молекулы гелия имеют такую же среднюю скорость, как молекулы кислорода при 23°C ?

5. Найдите среднюю кинетическую энергию поступательного движения молекул водорода, если при давлении 0.5 атм. их концентрация равна $1.5 \cdot 10^9 \text{ м}^{-3}$.
6. Газ нагревается в открытом сосуде при нормальном атмосферном давлении от 27°C до 327°C . Какое приращение получает при этом число молекул в единице объема газа?
7. В сосуде объемом 1 дм^3 содержится некоторый газ при температуре 17°C . Найти приращение давления газа, если вследствие утечки газа из него выйдет 10^{21} молекул.
8. В сосуде объемом 3.0 дм^3 находится гелий массой 4.0 мг, азот массой 70 мг и $5.0 \cdot 10^{21}$ молекул водорода. Каково давление смеси, если температура ее 27°C ?
9. Определить среднюю кинетическую энергию вращательного движения молекул водорода, содержащихся в 1.0 моль при 18°C .
10. При какой температуре молекулы кислорода имеют такую же среднюю скорость, как молекулы водорода при 25°C ?
11. Сколько молекул содержится при нормальных условиях в 1 м^3 воздуха?
12. В сосуде объемом 2.0 дм^3 находится газ под давлением 0.50 МПа. Чему равна средняя кинетическая энергия поступательного движения молекул газа?
13. Кислород массой 12 г находится при температуре 700°C , при этом 40% молекул диссоциировано на атомы. Чему равна средняя кинетическая энергия теплового движения частиц? Колебательные степени свободы молекул кислорода не возбуждаются.
14. Объём помещения 50 м^3 . температура воздуха зимой 0°C , а летом – 40°C . Какова разница в массе воздуха, заполняющего помещение зимой и летом? Принять $\mu = 29 \text{ кг/моль}$.
15. Плотность воздуха при нормальных условиях 1.3 г/л. Какова плотность воздуха при температуре 100°C и давлении $4.0 \cdot 10^5 \text{ Н/м}^2$?
16. Температура воздуха в баллоне объёмом 10 м^3 при давлении 700 мм рт. ст. была 15°C . После нагрева воздуха до 20°C часть воздуха была вытеснена. Найдите массу вытесненного воздуха.
17. Сколько частиц (атомов и молекул) находится в азоте массой 1.0 г, если степень диссоциации азота 7.0%.
18. Какое давление на стенки сосуда производит кислород, если средняя квадратичная скорость его молекул 400 м/с и концентрация молекул $2.7 \cdot 10^{19} \text{ м}^{-3}$?
19. Найдите среднюю кинетическую энергию поступательного движения молекул гелия, если при давлении 0.5 атм. их концентрация равна $1.5 \cdot 10^9 \text{ м}^{-3}$.
20. Определите температуру газа, если средняя кинетическая энергия поступательного движения его молекул равна $1.6 \cdot 10^{-19} \text{ Дж}$.
21. Каково давление газа, если в каждом кубическом сантиметре его содержится $1.0 \cdot 10^6$ молекул, а температура газа 87°C ?

22. Газ нагревается в открытом сосуде при нормальном атмосферном давлении от 22°C до 320°C . Какое приращение получает при этом число молекул в единице объема газа?
23. В сосуде объемом 15.0 дм^3 находится газ под давлением 0.50 МПа . Чему равна средняя кинетическая энергия поступательного движения молекул газа?
24. При какой температуре молекулы гелия имеют такую же среднюю скорость, как молекулы водорода при 27°C ?
25. Газ занимает объём 2 л при давлении $5 \cdot 10^5 \text{ Н/м}^2$. Определите суммарную энергию поступательного движения молекул газа.

Задания для самостоятельной работы

1. Даны два действительных числа. Найти среднее арифметическое и среднее геометрическое этих чисел.
2. Даны четыре целых числа. Найти среднее арифметическое этих чисел и среднее геометрическое этих чисел.
3. Треугольник задан длинами своих сторон. Используя формулу Герона, найти площадь этого треугольника.
4. Идет k -я секунда суток. Определить сколько полных часов H , полных минут M прошло к этому моменту.
5. Даны два момента времени одних суток: H_1, H_2 - часы, M_1, M_2 - минуты. Определить интервал между этими моментами в часах H и минутах M .
6. По трем координатам вершин некоторого треугольника найти его высоту.
7. Найти объем и площадь боковой поверхности конуса радиусом r , высотой h и образующей l .
8. Вычислить боковую поверхность пирамиды (пирамида правильная, усеченная), если известны p и p_1 - полупериметры нижнего и верхнего оснований и k - апофема.
9. Найти объем и полную поверхность полого шара, если известны r_1 и r_2 - радиусы внешней и внутренней шаровых поверхностей.
10. Вычислить площадь поверхности и объем усеченного конуса высотой h , радиусами r_1 и r_2 , образующей l .
11. В шар радиуса r вписан конус с углом α при вершине в осевом сечении конуса. Определить объем и полную поверхность конуса.
12. Вычислить объем цилиндра, вписанного в правильную шестиугольную призму, у которой каждое ребро равно a .
13. Найти площадь равнобокой трапеции с основаниями A и B и углом α при большем основании A .
14. Треугольник задан величинами своих углов и радиусом описанной окружности. Найти, стороны треугольника.

15. Найти объем правильной треугольной пирамиды, если стороны ее основания равны a и плоский угол при ее вершине равен α .
16. Вычислить радиус шара, вписанного в правильную четырехугольную пирамиду, если стороны основания пирамиды равны a и двугранный угол при основании равен α .
17. Вычислить площадь круга и равнобедренной трапеции, описанной около него, если периметр трапеции равен p , а угол при нижнем основании равен α .
18. Вычислить объем призмы, боковые грани которой – квадраты. Основанием призмы служит равносторонний треугольник, по который известно, что радиус вписанной в него окружности равен r .

Контрольные вопросы

1. Что такое программа на C++?
2. Какая программа является линейной?
3. Какова структура программы на C++?
4. Что такое идентификатор?
5. По каким правилам создаются идентификаторы в C++?
6. Что такое тип величины?
7. Какие типы величин используются в языке C++?
8. Какие типы величин относятся к простым типам?
9. В каком диапазоне могут изменяться переменные типа **int** и **float**?
10. Как описываются переменные в C++? (Привести пример.)
11. Что такое операторные скобки? Каково их назначение?
12. Как выполняется команда присваивания?
13. Какая инструкция предназначена для вывода на экран сообщений?
14. Какая инструкция предназначена для ввода данных?
15. Что такое заголовочный файл? Приведите пример заголовочного файла.
16. Каково назначение заголовочных файлов?
17. Как сохранить программу? (Устно)
18. Как осуществить компиляцию текста программы? (Устно)
19. Как осуществить запуск программы? (Устно)
20. Как считать готовую программу с диска? (Устно)

Пример выполнения лабораторной работы

Задание I. Напишите программу для расчета по двум формулам. Отсутствующие в библиотеке языка функции выразите через имеющиеся.

$$z_1 = tg(x) - ctg \frac{7\pi}{6},$$

$$z_2 = tg \frac{7\pi}{8} - \sqrt{3} \cdot tg \frac{\pi}{8} \cdot tg(x)$$

Решение

1. Математическая модель

В библиотеке `math.h` языка C++ отсутствует функция $ctg(x)$, поэтому выразим ее через имеющуюся функцию $tg(x)$, используя известное соотношение:

$$ctg(x) = \frac{1}{tg(x)}.$$

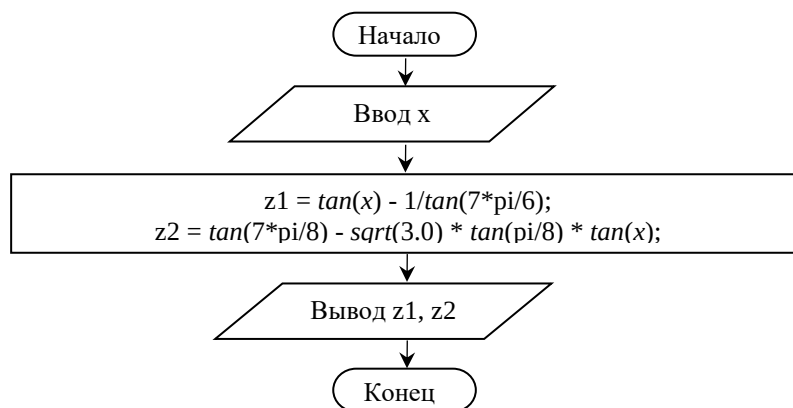
Запись выражений на языке C++ примет вид:

`z1 = tan(x) - 1/tan(7*pi/6);` `z2 = tan(7*pi/8) - sqrt(3.0) * tan(pi/8) * tan(x).`

Аргументы: x – вещественного типа.

Результаты : z_1, z_2 – вещественного типа.

2. Алгоритм (блок - схема)



3. Программа

```
//подключение заголовочных файлов
#include <iostream> //для организации ввода-вывода в C++
#include <conio.h> //для организации консольного ввода-вывода
#include <math.h> //для работы с математическими функциями

using namespace std; // описание пространства имен std

int main() // заголовок главной функции
{ // тело функции
    const double pi = 3.14159265359; // описание константы
    double z1, z2, x; // описание переменных
    cout<<" Введите x = "; // вывод сообщения на экран
    cin>>x; // ввод переменной x
    //вычисляем значения переменных z1, z2
    z1 = tan(x)-1/tan(7*pi/6); //присвоение переменной z1 значения выражения
    z2 = tan(7*pi/8)-sqrt(3.0)*tan(pi/8)*tan(x);
    cout<<" z1 = "<<z1<<" z2 = "<<z2; // вывод данных на экран
    _getch(); //функция ожидания нажатия клавиши из conio.h
    return 0; //главная функция возвращает целочисленное значение 0
}
```

4. Результат работы программы

```
Введите x = 4
z1 = -0.57
z2 = -1.24
```

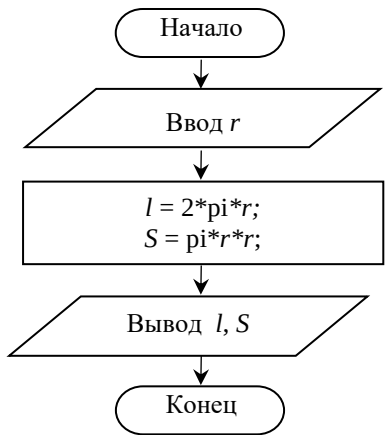
Задание II. Написать программу для вычисления длины окружности и площади круга заданного радиуса.

Решение

1. Математическая модель $l = 2 \cdot \pi \cdot r$, $S = \pi \cdot r^2$.

Аргументы: радиус окружности r , вещественного типа.

Результаты: длина окружности l и площадь круга S , вещественного типа.

2. Блок – схема	3. Программа
 <pre> graph TD Start([Начало]) --> Input[/Ввод r/] Input --> Process[l = 2*pi*r; S = pi*r*r] Process --> Output[/Вывод l, S/] Output --> End([Конец]) </pre>	<pre> //программа нахождения длины окружности и площади // круга заданного радиуса #include <iostream> #include <conio.h> using namespace std; int main()//заголовок главной функции {//описание переменных, начальные установки setlocale(LC_ALL, "Rus"); double pi = 3.14; double r, l, S; cout<<"Введите радиус: r = "; cin>>r; // ввод переменной //вычисление по формулам l = 2*pi*r; S = pi*r*r; //вывод данных cout<<endl<<"Длина окружности: l = "<<l; cout<<endl<<"Площадь окружности: S = "<<S; _getch(); //главная функция возвращает целое значение 0 return 0; } </pre>

4. Результат работы программы:

Введите радиус: $r = 5$
 Длина окружности: $l = 31.4$
 Площадь круга: $S = 78.5$

СПИСОК ЛИТЕРАТУРЫ

1. Конспект лекций
2. [2] стр. 17-31.

Лабораторная работа № 3

Программирование алгоритмов разветвляющейся структуры

ЦЕЛЬ РАБОТЫ: приобретение навыков программирования вычислительных разветвляющихся процессов с использованием условного, составного операторов, оператора выбора (множественного ветвления) и оператора безусловного перехода.

Формируемые компетенции или их части: вырабатывается навык разработки математических и информационных моделей, а также формируется способность к разработке и применению алгоритмических и программных решений (разветвляющейся структуры) в области прикладного программирования (формируется часть компетенций ОПК-3 и ПК-7).

Теоретическая часть

Операторы ветвления

3.1. Простые и составные условия

В C++ условие – выражение логического типа (**bool**), которое может принимать одно из двух значений: «истина» (**true**) или «ложь» (**false**). Внутренняя форма представления значения **false** - 0 (нуль). Любое другое значение интерпретируется как **true**. При преобразовании к целому типу **true** имеет значение 1.

Используются шесть операторов, позволяющих сравнивать между собой значения выражений, числовых переменных, а также значение переменной и константу:

> (больше),	!= (не равно),
< (меньше),	>= (больше либо равно),
== (равно),	<= (меньше либо равно).

Условия, которые составлены с использованием одного оператора сравнения, называются **простыми условиями**, например, $x + y > 0$.

Общий вид:

выражение **оператор сравнения** выражение

Из простых условий, которые являются выражениями логического типа можно строить **сложные условия**, применяя к ним, как к операндам, логические операторы: **!** (не), **&&** (и), **||** (или).

Логические операции выполняются слева направо. Если значение первого операнда достаточно, чтобы определить результат операции, второй операнд не вычисляется. Для изменения порядка действий используются круглые скобки.

Итак, **составные условия** – это простые условия, связанные при помощи логических операторов: $(x > 5) \ || \ (a + b < 0)$.

Операция &&

Логическую операцию **&&** (и) используют, если нужно, чтобы одновременно выполнялись два условия. В C++ это записывается так:

(условие1) && (условие2)

На практике эта логическая операция часто применяется для создания условия принадлежности некоторой переменной указанному промежутку. Например, составим условие, которое будет истинно тогда и только тогда, когда переменная x принадлежит промежутку от 10 до 20: $10 < x < 20$. Т.е., когда $x > 10$ **и** $x < 20$. На языке C++ это записывается так: $(x > 10) \&\& (x < 20)$.

Операция ||

Логическую операцию || (или) используют, когда хотят сформулировать условие, которое будет истинно в том случае, когда верно хотя бы одно условие из двух. В C++ эта операция используется так:

(условие1) || (условие2)

Операция !

Логическую операцию ! (не – логическое отрицание) используют, если нужно проверить условие (выражение) на ложность. Эта операция делает ложным истинное условие и истинным ложное. Запись операции:

! (условие)

В таблице 6 приводятся результаты применения логических операторов к операндам логического типа. Для наглядности вместо значения false используется 0, true – 1.

Таблица 6 – Логические операции

a	b	a && b	a b	! a
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

При записи сложных условий надо учитывать приоритет операций и, в случае необходимости, использовать круглые скобки.

3.2. Составной оператор

К составным операторам относятся собственно составные операторы и блоки.

Составной оператор – последовательность операторов, заключенная в операторные скобки. Операторы, входящие в него, должны выполняться в том порядке, в котором они записаны.

Формат оператора:

```
{
    оператор1;
    оператор2;
    ...
    операторN;
}
```

Блок отличается тем, что в его состав входят описания каких-либо объектов программы.

Например:

<pre>{// Это составной оператор n ++; S += n; }</pre>	<pre>{// Это блок int n = 0; n ++; S += n; }</pre>
---	--

Блоки, чаще всего, используются в качестве тела функции. Составной оператор используется в любом случае, когда несколько операторов следует объединить в один. Это необходимо в условных операторах и операторах цикла, согласно синтаксису которых, исполнимым является только один оператор. Чаще всего, составной оператор формирует ветвь условного оператора или тело цикла в операторах цикла.

3.3. Условная операция (?:)

Эта операция тернарная, то есть имеет три операнда.

Формат операции:

операнд_1 ? операнд_2 : операнд_3

Первый операнд может иметь арифметический тип или быть указателем. Он оценивается с точки зрения его эквивалентности нулю (операнд, равный нулю, рассматривается как **false**, не равный нулю – как **true**). Если результат вычисления операнда 1 равен **true**, то результатом условной операции будет значение второго операнда, иначе – третьего операнда. Вычисляется всегда либо второй операнд, либо третий. Их тип может различаться. Условная операция является сокращенной формой условного оператора **if**.

Пример: Вывести на экран минимальное из двух значений.

```
#include <stdio.h>
int main ( )
{
    int a = 10, b = 5, min;
    min = (b < a) ? b : a ;
    printf("Наименьшее число: %d", min);
    return 0;
}
```

Результат работы программы:

Наибольшее число: 5

Другой пример применения условной операции. Требуется, чтобы некоторая целая величина увеличивалась на 1, если ее значение не превышает n, а иначе принимала значение 1:

```
i = (i < n) ? i + 1: 1;
```

3.4. Условный оператор if

Предназначен для выбора к исполнению одного из двух возможных операторов в зависимости от выполнения некоторого условия.

Формат оператора:

if (условие) оператор 1; **else** оператор 2;

Инструкция выполняется следующим образом:

1. Вычисляется значение условия (выражения), которое может иметь арифметический тип или тип указателя.
2. Если значение выражения (условия) не равно нулю (имеет значение **true**), выполняется первый оператор, следующий за условием. Если значение выражения (условия) равно **false**, то выполняются инструкции, следующие за словом **else**.

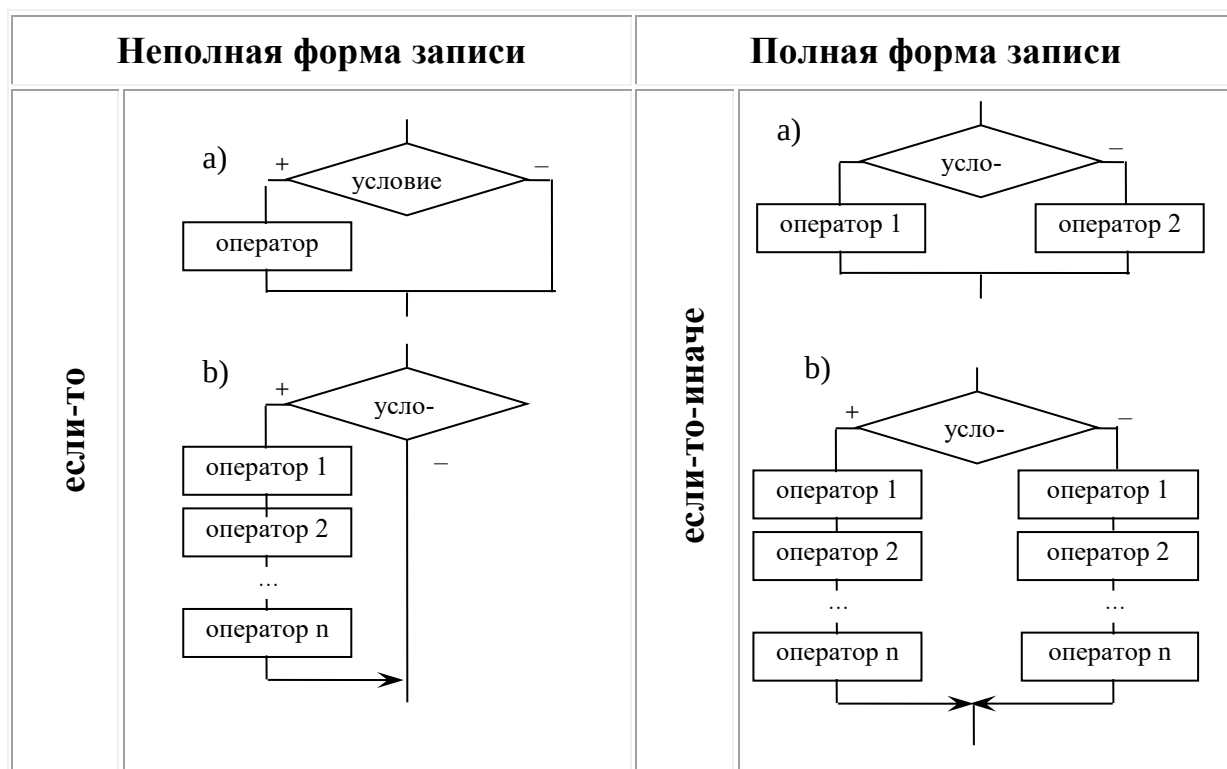
Довольно часто в случае ложности условия не нужно производить никаких действий. Допускается использование краткой формы условного оператора:

if (условие) оператор;

Если значение логического выражения истинно, то выполняются инструкции, следующие за условием. Если значение выражения ложно, то выполняется инструкция, следующая непосредственно после оператора **if**.

Если после условия или служебного слова **else** требуется выполнить несколько операторов, то их оформляют в виде составного.

Таблица 7 – Графическая интерпретация условного оператора.



Замечание. В таблице 7 приводятся блок-схемы возможных вариантов алгоритма команды «Ветвление» (условного оператора) случаи

а) для простых операторов;

б) для составных, когда при выполнении условия или при не выполнении его требуется исполнить несколько операторов: оператор 1, оператор 2, ..., оператор n. **В программе** данную серию операторов: оператор 1, оператор 2, ..., оператор n оформляют в виде составного оператора или блока, заключив их в операторные скобки.

Структура команды «Ветвление» называется вложенной, если после условия или служебного слова **else** используются вновь условные операторы. Число вложений может быть произвольным. При этом справедливо следующее правило: служебное слово **else** всегда относится к ближайшему **if**.

Пример. Составить программу нахождения минимального из трех чисел, введенных с клавиатуры.

1 вариант решения:	2 вариант решения:
Программа на C++	
<pre>//min_of_3_numbers; #include <conio.h> #include <iostream> using namespace std; int main(){ setlocale(LC_ALL, "Rus"); int a, b, c, max; cout<<"\nВведите 3 числа через пробел: "; cin>>a>>b>>c; if (a < b) if (a < c) max = a; // случай, когда a < b else max = c; else if (b < c) max = b; // случай, когда a >= b else max = c; cout<<"Минимальное число = "<< max;</pre>	<pre>//min_of_3_numbers; #include <conio.h> #include <iostream> using namespace std; int main() { setlocale(LC_ALL, "Rus"); int a, b, c, max; cout<<"\nВведите 3 числа через пробел: "; cin>>a>>b>>c; max = a; if (b < max) max = b; if (c < max) max = c; cout<<"Минимальное число = "<<max;</pre>

1 вариант решения:	2 вариант решения:
<pre> _getch(); return 0; } </pre>	<pre> _getch(); return 0; } </pre>

Результат работы программы:

Введите 3 числа через пробел:

12 5 -7

Минимальное число = -7

3.5. Оператор switch

Если в программе нужно реализовать выбор из более, чем двух вариантов, то можно использовать инструкцию множественного ветвления (переключатель).

Оператор **switch** (переключатель) предназначен для разветвления процесса вычислений на несколько направлений. Блок-схема оператора **выбора (switch)** предложена в таблице 8.

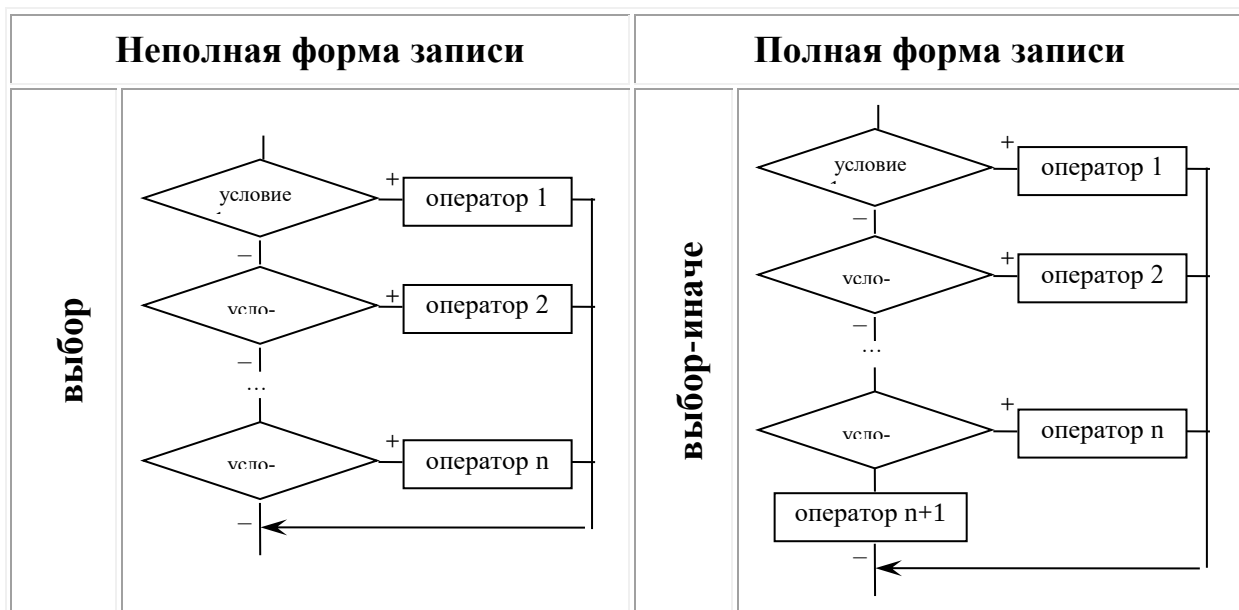
Формат оператора:

```

switch ( выражение )
{
    case константное_выражение_1: [оператор_1];
    case константное_выражение_2: [оператор_2];
    ...
    case константное_выражение_n: [оператор_n];
    [default: оператор_n + 1];
}

```

Таблица 8 – Оператор выбора (блок-схема)



Оператор выбора является обобщением условного оператора. Выполнение оператора начинается с вычисления выражения (оно должно быть целочисленным), а затем управление передается первому оператору из списка, помеченного константным выражением, значение которого совпало с вычисленным. *После*

этого, если выход из переключателя явно не указан, последовательно выполняются все остальные ветви. Затем управление передается оператору, следующему за оператором выбора. Если значение выражения не совпадает ни с одним константным выражением, то выполняется оператор $n + 1$.

Значение выражения и констант оператора **switch** должны быть одного и того же типа. Тип может быть любым целочисленным. В качестве константы допускается использование списка констант (например, **case1: case3: case5: [оператор_1];**), а в качестве оператора – составной оператор.

Если константы представляют диапазон чисел, то вместо списка можно указать первую и последнюю константу диапазона, разделив их тремя точками (справедливо для некоторых сред программирования, например, MinGW).

Выход из переключателя обычно выполняется с помощью операторов **break** или **return**. Оператор **break** выполняет выход из самого внутреннего из объемлющих его операторов **switch**, **for**, **while** и **do**. Оператор **return** выполняет выход из функции, в теле которой он записан.

3.6. Оператор перехода *goto*

Оператор безусловного перехода используется для изменения последовательности выполнения операторов.

Формат оператора:

goto метка;

В теле той же функции должна присутствовать ровно одна конструкция вида:

метка: оператор;

Оператор **goto** передает управление на помеченный оператор.

Метка – это обычный идентификатор, областью видимости которого является функция, в теле которой он задан.

Использование оператора безусловного перехода оправданно, как правило, в двух случаях:

- принудительный выход вниз по тексту программы из нескольких вложенных циклов или переключателей;
- переход из нескольких мест функции в одно (например, если перед выходом из функции необходимо всегда выполнять какие-либо действия).

Во всех остальных случаях следует привести алгоритм к структурному виду, то есть преобразовать его так, чтобы он мог быть записан с помощью базовых конструкций.

Использование оператора безусловного перехода считается плохим стилем программирования, т.к. нарушает принципы структурного и модульного программирования, по которым все блоки, из которых состоит программа, должны иметь только один вход и один выход.

Пример: Дано число m ($1 \leq m \leq 7$). Вывести на экран название дня недели, который соответствует этому номеру.

блок-схема	программа
<pre> graph TD Start([начало]) --> Input[/ВВОД n/] Input --> D1{n = 1} D1 -- + --> P1[/'понедельник'/] D1 -- - --> D2{n = 2} D2 -- + --> P2[/'вторник'/] D2 -- - --> D3{n = 3} D3 -- + --> P3[/'среда'/] D3 -- - --> D4{n = 4} D4 -- + --> P4[/'четверг'/] D4 -- - --> D5{n = 5} D5 -- + --> P5[/'пятница'/] D5 -- - --> D6{n = 6} D6 -- + --> P6[/'суббота'/] D6 -- - --> D7{n = 7} D7 -- + --> P7[/'воскресенье'/] D7 -- - --> P8[/'неверный номер'/] P1 --> End([конец]) P2 --> End P3 --> End P4 --> End P5 --> End P6 --> End P7 --> End P8 --> Input </pre>	<pre> //switch; #include <conio.h> #include <iostream> using namespace std; int main() { setlocale(LC_ALL, "Rus"); int n; metka:cout<<"\nВведите номер дня недели:"; cin>>n; switch (n) { case 1 : cout<<"\nПонедельник";break; case 2 : cout<<"\nВторник";break; case 3 : cout<<"\nСреда";break; case 4 : cout<<"\nЧетверг";break; case 5 : cout<<"\nПятница";break; case 6 : cout<<"\nСуббота";break; case 7 : cout<<"\nВоскресенье";break; default : cout<<"\nНеверный ввод\n"; goto metka; } _getch(); return 0; } </pre> Результат работы программы: Введите номер дня недели: 12 Неверный ввод Введите номер дня недели: 2 Вторник

Практическая часть

ВЫПОЛНЕНИЕ РАБОТЫ: в соответствии с вариантом составить и реализовать программы (задание I- III).

Задание I

Решить задачу с использованием конструкции ветвления [5]:

1. Расстояние от дома до школы S км. До начала урока осталось t минут. Ученик идет со скоростью v . Определить, придёт ли он раньше звонка, после звонка или во время.
2. Даны действительные числа x , y и z . Найти $\min(x, y, z)$.

3. Вычислить значение функции $f = \begin{cases} 2x\sqrt{x}, & \text{если } x > 1, \\ \frac{x^2}{5}, & \text{если } x \leq 0. \end{cases}$
4. Возвести в квадрат большее из трёх заданных чисел.
5. Определить, есть ли среди трёх заданных чисел чётные.
6. Вывести на экран три заданных числа в порядке убывания.
7. Определить, принадлежит ли точка $A(a_1, a_2)$ кольцу, определяемому окружностями: $x^2 + y^2 = 1$ и $x^2 + y^2 = 16$.
8. Вычислить значение функции $y = \begin{cases} \ln^3(x^2 + 1), & \text{если } x \leq -3, \\ \arctg \frac{1}{2x+1}, & \text{если } x > 3. \end{cases}$
9. Две окружности заданы координатами центра и радиусами. Сколько точек пересечения имеют эти окружности?
10. В два сосуда, один из которых имеет форму шара радиуса R_1 , а другой – форму куба с ребром A , доверху налита вода. Сравните количество воды в сосудах.
11. Вычислить значение функции: $f = \begin{cases} x \sin^2 x, & \text{если } x < 1, \\ e^x, & \text{если } x \geq 2. \end{cases}$
12. Даны три произвольных действительных числа. Можно ли построить треугольник с такими длинами сторон?
13. Какая из точек $A(a_1, a_2)$ или $B(b_1, b_2)$ находится дальше от начала координат?
14. Две прямые заданы уравнениями: $a_1x + b_1y + c_1 = 0$ и $a_2x + b_2y + c_2 = 0$. Определить, пересекаются ли они.
15. Попадёт ли точка $A(a_1, a_2)$ в окружность заданного радиуса с центром в начале координат?

$$16. \text{ Вычислить значение функции } y = \begin{cases} ax^2 + bx + c, & \text{если } x < 1.2, \\ \frac{a}{x} + \sqrt{x^2 + 1}, & \text{если } x = 1.2, \\ \frac{a + bx}{\sqrt{x^2 + 1}}, & \text{если } x > 1.2, \end{cases}$$

где $a = 2.8$, $b = -0.3$, $c = 4$.

17. В заборе выпилена дыра прямоугольной формы с известными размерами. Определить, пройдёт ли в эту дыру мяч заданного радиуса.
18. Две окружности заданы координатами центра и радиусами. Определить, пересекаются ли они, касаются друг друга или не имеют общих точек.
19. В григорианском календаре каждый год, номер которого делится на 4, является високосным, за исключением тех, которые делятся на 100 и не делятся

на 400 нацело. Определить число дней в году по номеру года. Т.о. 1900 г. - не високосный, 2000 г. – високосный.

20. Определить, есть ли среди четырёх заданных чисел кратные пяти.

21. Вычислить значение функции $y = \begin{cases} at^2 \ln t, & \text{если } 1 \leq t \leq 2, \\ 1, & \text{если } t < 1, \\ e^{at} \cos bt, & \text{если } t > 2. \end{cases}$, где $a = -0.5, b = 2$.

22. Из трех действительных чисел a, b и c выбрать те, модули которых больше 4.

23. Даны три произвольных действительных числа. Можно ли построить треугольник с такими длинами сторон? Если да, то вывести на экран является ли заданный треугольник равносторонним, равнобедренным или разносторонним.

24. Даны действительные числа x и y . Найти $U = \max^2(x^2y, xy^2)$.

25. Решить квадратное уравнение.

Задание II

Решить задачу с использованием конструкции выбора [5]:

1. Дано число c . Распечатать величину этого числа в словесной форме, учитывая его знак. Предусмотреть, что $-7 \leq c \leq 7$.
2. Дано число m ($1 \leq m \leq 12$). Определить, сколько дней в месяце с номером m . Год не високосный.
3. С клавиатуры вводится натуральное число n – количество граней выпуклого многогранника. В зависимости от значения n вывести на экран фразу «Выпуклый многогранник имеет n граней», согласовав окончание слова "грань" с числом n . Предусмотреть, что $0 \leq n \leq 25$. Если выпуклого многогранника с заданным количеством граней не существует, выдать соответствующее сообщение.
4. С клавиатуры вводится цифра m (от 1 до 4). Вывести на экран названия месяцев, соответствующих времени года с номером m (считать зиму временем года № 1).
5. С клавиатуры вводится натуральное число n – количество граней выпуклого многогранника. В зависимости от значения n вывести на экран в словесной форме название правильного выпуклого многогранника или сообщить, что правильного выпуклого многогранника с таким количеством граней не существует.

Предусмотреть, что $4 \leq n \leq 20$.

Примечание. Существует пять выпуклых правильных многогранников: тетраэдр (4 грани); гексаэдр или куб (6 граней); октаэдр (8 граней); додекаэдр (12 граней); икосаэдр (20 граней)

6. Дано натуральное число $n < 25$. Вывести на экран фразу «мне n лет (год или года)» соответственно.

7. Найти значение функции:
$$f = \begin{cases} 2\sin^2 x, & \text{если } -1 \leq x \leq 1, \\ 3x^3, & \text{если } -10 < x < -1, \\ 0 & \text{в остальных случаях.} \end{cases}$$
8. Дано число m ($1 \leq m \leq 12$). Определить, к какому времени года относится месяц с номером m .
9. Вводится целое число $|C| < 5$. Если $C > 0$, то вывести на экран название этого числа в словесной форме. В противном случае сообщить, что число отрицательное.
10. С клавиатуры вводится натуральное число n – количество углов многоугольника. В зависимости от значения n вывести на экран название многоугольника или сообщить, что многоугольника с таким количеством углов не существует.
- Предусмотреть, что $0 \leq n \leq 6$.
11. С клавиатуры вводится натуральное число n . В зависимости от значения остатка r при делении числа n на 7 вывести на экран число n в виде $n = 7k + r$, где r представить в словесной форме.
12. Дано число m . Определить полугодие, на которое приходится месяц m и количество дней в этом месяце. Предусмотреть, что $1 \leq m \leq 12$. Год не високосный.
13. Дано число n . Напечатать фразу "Мы успешно сдали n экзаменов", согласовав окончание слова "экзамен" с числом n . Предусмотреть, что $1 \leq n \leq 25$.
14. Определить время года, к которому относится месяц m и определить количество дней в этом месяце. Предусмотреть, что $1 \leq m \leq 12$. Год високосный.
15. Дано число c . Распечатать величину этого числа в словесной форме. Предусмотреть, что $0 \leq c \leq 7$.
16. Даны два числа D – день и M – месяц. Определить K – порядковый номер того дня високосного года, который имеет дату D и M .
17. Даны три числа D , M и G , определяющие день, месяц и год. Проверить образуют ли они правильную дату и вывести соответствующее сообщение. Например, 31.06.90 - неправильная дата. Год не високосный.
18. С клавиатуры вводится цифра от 0 до 9. Вывести на экран название этой цифры в словесной форме.
19. В понедельник фирма работает с 9-00 до 16-00; во вторник, среду, четверг, пятницу – с 8-00 до 19-00; в субботу – с 10-00 до 15-00; воскресенье – выходной. По заданному номеру дня недели определить часы работы.

$$20. \text{ Найти значение функции: } y = \begin{cases} x\sqrt{x-a^2}, & \text{если } x = a+1; \\ \frac{a}{x} - \frac{1}{2}\cos 2x, & \text{если } 0 \leq x \leq a; \text{ где } a = 4. \\ e^x, & \text{если } -a \leq x < 0; \\ 1 & \text{в иных случаях.} \end{cases}$$

21. С клавиатуры вводится натуральное число n . В зависимости от значения остатка r при делении числа n на 5 вывести на экран число n в виде $n = 5k + r$, где r представить в словесной форме.
22. На первом курсе студенты четырёх групп сдают зачёт по информатике. На втором курсе группа № 1 сдаёт экзамен по дисциплине «Информационные технологии в экономике», группа № 2 – экзамен по дисциплине «Информационные технологии в физике», группа № 3 – экзамен по дисциплине «Информационные технологии в географии», группа № 4 – экзамен по дисциплине «Информационные технологии в математических исследованиях». Задать с клавиатуры номер курса и номер группы и вывести на экран сообщение, какой зачёт или экзамен группа будет сдавать.
23. С клавиатуры задано целое число $N < 5$. Если оно положительное, то вывести на экран его название в словесной форме. В противном случае найти квадрат этого числа.
24. Студенты убирают яблоки. Нормы следующие: студент 1 курса должен собрать в день не менее 50 кг яблок, студент второго курса – не менее 60 кг в день, студент 3 курса – не менее 70 кг в день. Зная курс, на котором учится студент, определить, выполнил ли он дневную норму.
25. Дано число m ($1 \leq m \leq 12$). Определить, сколько дней в месяце с номером m . Учесть високосный или не високосный год.

Задания для самостоятельной работы

1. Определить D и M -дату K -го по счету дня високосного года. Месяц вывести в словесной форме. D - день, M - месяц. Предусмотреть, что $1 \leq K \leq 366$.
2. Компания по снабжению электроэнергией взимает плату с клиентов по тарифу:
 - 7 р. за 1 кВт/ч за первые 250 кВт/ч;
 - 17 р. за кВт/ч, если потребление свыше 250, но не превышает 300 кВт/ч;
 - 20 р. за кВт/ч, если потребление свыше 300 кВт/ч.
 Потребитель израсходовал n кВт/ч. Подсчитать плату.
3. Школьники сдают нормы по прыжкам в длину. Если длина прыжка больше 2,50 м, то оценка – «5», если от 2 м до 2,5 м - оценка «4»; от 1,5 м до 2 м - оценка «3»; если меньше 1,5 м – «2». Выставить школьнику оценку, если известна длина его прыжка. Примечание: 1 м = 100 см.

4. При покупке товара на сумму от 200 до 500 руб предоставляется скидка 3%, при покупке товара на сумму от 500 до 800 – скидка 5%, при покупке товара на сумму от 800 до 1000 руб – скидка 7%, свыше 1000 руб – скидка 10%. Покупатель приобрел 8 рулонов обоев по цене X_1 и две банки краски по цене X_2 . Сколько он заплатил?
5. Студенты убирают урожай помидоров. При сборе до 50 кг в день работа оплачивается из расчёта 30 руб. за 1 кг; при сборе от 50 до 75 кг в день – 50 руб. за 1 кг; при сборе от 75 до 90 кг в день – 65 руб. за 1 кг; при сборе свыше 90 кг в день – 70 руб. за 1 кг плюс 120 руб. премия. Студент собрал X кг за день. Определить его заработок.
6. Ученики начальной школы сдают технику чтения. В 1 классе скорость чтения до 20 слов в мин считается низкой, от 21 до 40 слов в мин – средней, от 41 до 50 – выше среднего, более 51 слова в мин – высокой. Во втором классе скорость чтения до 30 слов в мин считается низкой, от 31 до 60 слов в мин – средней, от 61 до 70 – выше среднего, более 71 слова в мин – высокой. Задать с клавиатуры номер класса и количество прочитанных учеником в минуту слов. Оценить скорость чтения.
7. Студенты убирают яблоки. Студентам 1 курса при сборе до 50 кг в день работа оплачивается из расчёта 30 коп. за 1 кг; при сборе от 50 до 75 кг в день – 50 коп. за 1 кг; при сборе от 75 до 90 кг в день – 65 коп. за 1 кг; при сборе свыше 90 кг в день – 70 коп. за 1 кг плюс 20 руб премия. Студентам 2 курса при сборе до 60 кг в день работа оплачивается из расчёта 35 коп. за 1 кг; при сборе от 61 до 80 кг в день – 55 коп. за 1 кг; при сборе от 81 до 95 кг в день – 65 коп. за 1 кг; при сборе свыше 95 кг в день – 70 коп. за 1 кг плюс 30 руб премия.
Студент собрал X кг яблок за день. Определить его заработок.
8. С клавиатуры задаётся целое число $A > -5$. Для положительного однозначного числа определить, является ли оно чётным; для положительного двузначного определить, кратно ли оно 3; для положительного трёхзначного числа определить, кратно ли оно 5; для числа, большего 999, определить, кратно ли оно 10. Если число $A < 0$, то вывести на экран его название в словесной форме.
9. Школьники сдают нормы по прыжкам в длину. Для учеников 5 класса нормы следующие: если длина прыжка больше 2,50 м, то оценка - 5, если от 2 м до 2,5 - оценка 4; от 1,5 м до 2 м - оценка 3; если меньше 1,5 м - 2. Нормы для учеников 6 класса: если длина прыжка больше 3,0 м, то оценка - 5, если от 2,5 м до 3,0 - оценка 4; от 2,0 м до 2,5 м - оценка 3; если меньше 2,0 м – оценка 2. Ученики других классов прыжки не сдают.
Выставить школьнику оценку, если известны номер его класса и длина прыжка.
10. Школьники сдают нормы по прыжкам в длину. Для учеников 2 класса минимальная длина прыжка должна быть не менее 1,2 м; для учеников 3 класса – не менее 1,5 м; для 4 класса – не менее 1,7 м; для 5 класса не менее 2,0 м.
Зная номер класса и длину прыжка ученика, определить, сдан ли норматив.

11. С клавиатуры задаётся натуральное число $N < 100$. Если оно однозначное, то вывести на экран его название в словесной форме. Если число двузначное, то сообщить об этом и определить остаток от деления его на 2, 3, 5.
12. Компания по снабжению электроэнергией взимает плату с клиентов по тарифу:
 - ✓ 7 р. за 1 кВт/ч за первые 250 кВт/ч;
 - ✓ 17 р. за кВт/ч, если потребление свыше 250, но не превышает 300 кВт/ч;
 - ✓ 20 р. за кВт/ч, если потребление свыше 300 кВт/ч.Существуют три категории потребителей. Тариф 1 категории составляет 75% общей стоимости, тариф 2 категории – половину общей стоимости. Остальные оплачивают полностью.

Потребитель израсходовал n кВт/ч. Подсчитать плату, зная его категорию.
13. С клавиатуры задано натуральное число $N < 10000$. Сообщить, является ли оно однозначным, двузначным, трёхзначным или четырёхзначным. В каждом случае определить, чётное оно или нечётное.
14. Ученики начальной школы сдают технику чтения. Нормы следующие: в 1 классе ребёнок должен читать не менее 20 слов в минуту; во втором классе – не менее 50 слов в мин; в 3 классе – не менее 60 слов в мин, в 4 классе – не менее 70 слов в минуту. Задавая с клавиатуры номер класса и количество прочитанных учеником в минуту слов, определить, сдана ли техника чтения.
15. Даны три целых числа определяющие дату: год, месяц, день. Считая, что год не високосный, определить дату следующего дня.

Контрольные вопросы

1. Что такое составной оператор?
2. Какова полная (неполная) форма команды ветвления (блок-схема)?
3. Каков алгоритм выполнения команды ветвления?
4. Каков алгоритм выполнения команды множественного ветвления (выбора)?
5. Какие операторы сравнения используются в C++?
6. Что называется простым условием? Приведите примеры.
7. Что такое составное условие? Приведите примеры.
8. Какие логические операторы допускаются при составлении сложных условий?
9. Каков результат применения оператора логическое "И" (&&)?
10. Каков результат применения оператора логическое "ИЛИ" (||)?
11. Каков результат применения оператора логическое отрицание (!)?
12. Каков общий вид (формат) инструкции «Ветвление»?
13. Каков алгоритм выполнения условной (тернарной) операции (?:)? Приведите пример.
14. Каков общий вид (формат) инструкции «Выбор»?
15. Может ли оператор ветвления содержать внутри себя другие ветвления?

Пример выполнения лабораторной работы

Задание I. Две окружности заданы радиусами. Определить какая из данных окружностей будет иметь большую площадь. Вывести значение большей площади на экран.

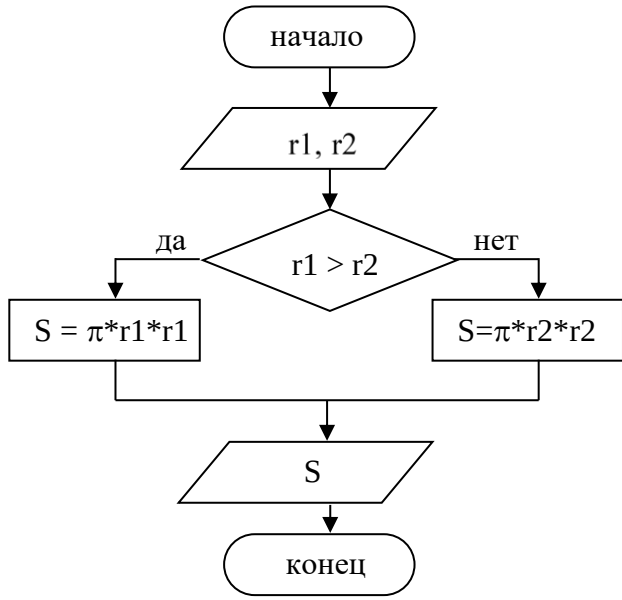
Решение

1. Математическая модель

Наибольшая площадь будет у окружности с наибольшим радиусом.

Аргументы: r_1 , r_2 вещественного типа – радиусы окружностей.

Результаты: S вещественного типа – значение наибольшей площади.

2. Алгоритм	3. Программа
 <pre> graph TD Start([начало]) --> Input[/r1, r2/] Input --> Decision{r1 > r2} Decision -- да --> S1[S = π * r1 * r1] Decision -- нет --> S2[S = π * r2 * r2] S1 --> S3[/S/] S2 --> S3 S3 --> End([конец]) </pre>	<pre> #include <iostream> #include <conio.h> #include <math.h> using namespace std; int main() { const double pi = 3.14159265359; double r1, r2, S; cout<<" Введите радиусы r1 и r2: "; cin>>r1>>r2; if (r1>r2) S = pi*pow(r1, 2.0); else S = pi*pow(r2, 2.0); cout<<" S = "<<S; _getch(); return 0; } </pre>

4. Результат работы программы

Введите радиусы r_1 и r_2 : 6 3

$S = 113.04$

Задание II. Дан номер месяца m . Вывести на экран название этого месяца.

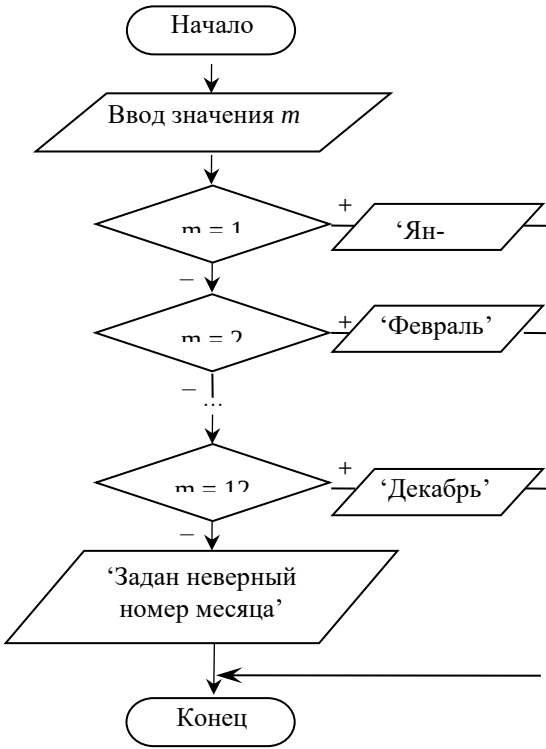
Решение

1. Математическая модель

Для решения задачи необходимо использовать инструкцию выбора.

Аргументы: m целого типа - номер месяца.

Результаты: текстовое сообщение.

2. Алгоритм	3. Программа
 <pre> graph TD Start([Начало]) --> Input[/Ввод значения m/] Input --> D1{m = 1} D1 -- + --> Out1[/Ян-/] D1 -- - --> D2{m = 2} D2 -- + --> Out2[/Февраль/] D2 -- - --> Dots[...] Dots --> D12{m = 12} D12 -- + --> Out12[/Декабрь/] D12 -- - --> Invalid[/Задан неверный номер месяца/] Out1 --> End([Конец]) Out2 --> End Out12 --> End Invalid --> End </pre>	<pre> #include <iostream> #include <conio.h> using namespace std; int main() { int a; cout<<"Введите номер месяца: "<<endl; cin>>a; switch (a) { case 1: cout<<"Январь"<<endl; break; case 2 : cout<<"Февраль"<<endl; break; case 3 : cout<<" Март "<<endl; break; case 4 : cout<<"Апрель"<<endl; break; case 5 : cout<<" Май"<<endl; break; case 6 : cout<<" Июнь"<<endl; break; case 7: cout<<" Июль"<<endl; break; case 8 : cout<<" Август"<<endl; break; case 9 : cout<<"Сентябрь"<<endl; break; case 10 : cout<<"Октябрь"<<endl; break; case 11 : cout<<"Ноябрь"<<endl; break; case 12 : cout<<"Декабрь"<<endl; break; default : cout<<" Ошибка ввода "<<endl; } _getch(); return 0; } </pre>

4. Результат работы программы

Введите номер месяца: 5
Май

СПИСОК ЛИТЕРАТУРЫ

3. Конспект лекций
4. [2] стр. 31-44.

Лабораторная работа № 4

Программирование алгоритмов циклической структуры

ЦЕЛЬ РАБОТЫ: приобретение навыков программирования вычислительных повторяющихся процессов с использованием циклических операторов.

Формируемые компетенции или их части: вырабатывается навык разработки математических и информационных моделей, а также формируется способность к разработке и применению алгоритмических и программных решений (циклической структуры) в области прикладного программирования (формируется часть компетенций ОПК-3 и ПК-7).

Теоретическая часть

Циклы

Цикл - вид конструкции, используемый для вычислений, повторяющихся многократно.

Блок, ради которого и организуется цикл, называется **телом цикла**. Остальные операторы служат для управления процессом повторения вычислений: это начальные установки, проверка условия продолжения цикла и модификация параметра цикла (рис.1). Один проход цикла называется **итерацией**.

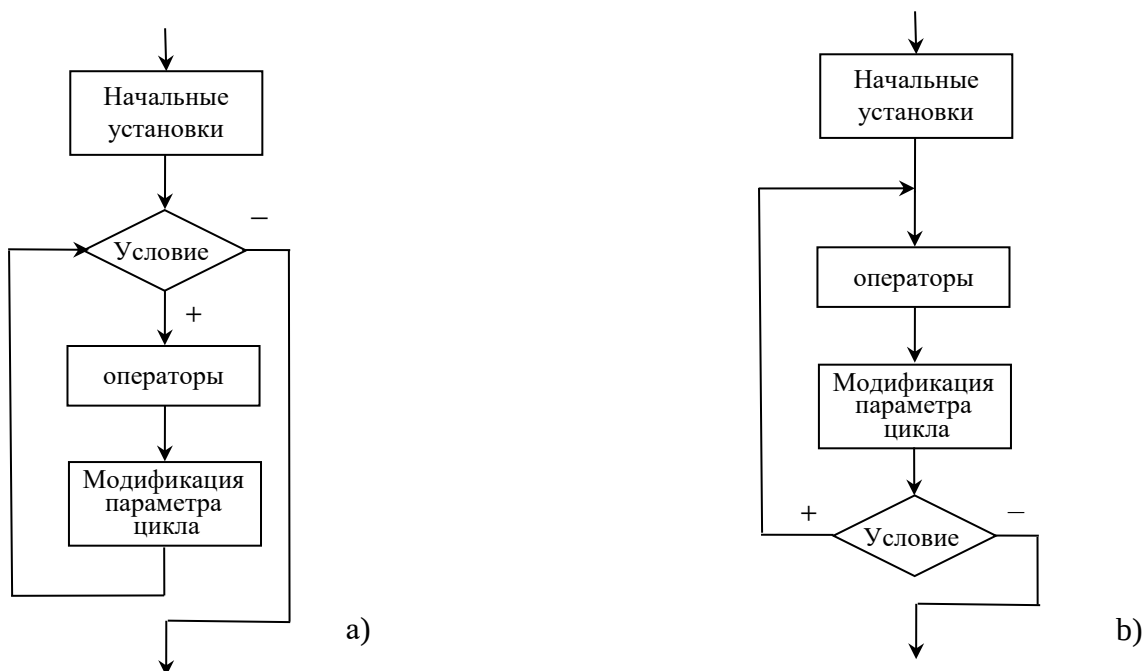


Рисунок 4.1 – Структурные схемы выполнения цикла

Начальные установки служат для того, чтобы до входа в цикл задать значения переменных, которые в нем используются.

Проверка условия продолжения цикла выполняется на каждой итерации либо до тела цикла (тогда говорят о цикле с предусловием, рис. 4.1а), либо после тела цикла (цикл с постусловием, рис 4.1б). Разница между ними состоит в том, что тело цикла с постусловием всегда выполняется хотя бы один раз, после чего проверяется надо ли его выполнять еще раз. Проверка необходимости выполнения цикла с предусловием делается до тела цикла, поэтому возможно, что он не выполнится ни разу.

Параметром цикла называется переменная, которая используется при проверке условия цикла и принудительно изменяется на каждой итерации, причем, как правило на одну и ту же величину. Если параметр цикла целочисленный, он называется счетчиком цикла.

Для реализации циклических вычислений предусмотрены три оператора: цикл с предусловием, цикл с постусловием, цикл с параметром. Первые два оператора используются в том случае, когда число повторений цикла неизвестно. Если же число повторений цикла может быть определено перед его началом, то используется оператор цикла с параметром [2].

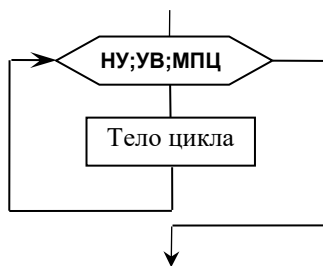
4.1. Оператор цикла с параметром (for)

Цикл с параметром (цикл со счетчиком, арифметический цикл, цикл «для») – это цикл с заранее известным числом повторений. Подходит только для программирования таких циклических фрагментов, в которых до выполнения цикла известны начальное и конечное значения счетчика повторений цикла.

Формат оператора:

for (инициализация; выражение; модификации) оператор;

Графическая интерпретация оператора цикла **for**



НУ – начальные установки (инициализация)

УВ – условное выражение

МПЦ – модификация параметра цикла

Инициализация используется для объявления и присвоения начальных значений величинам, используемым в цикле. В этой части можно записать несколько операторов, разделенных запятой.

Например, так:

```
for (int i = 0, j = 2; ...
```

или

```
int k, m;  
for (k = 1, m = 0 ...
```

Областью действия переменных, объявленных в части инициализации цикла, является цикл. Инициализация выполняется один раз в начале исполнения цикла.

Выражение определяет условие выполнения цикла: если его результат, приведенный к типу **bool**, равен **true**, цикл выполняется. Цикл с параметром реализован как цикл с предусловием.

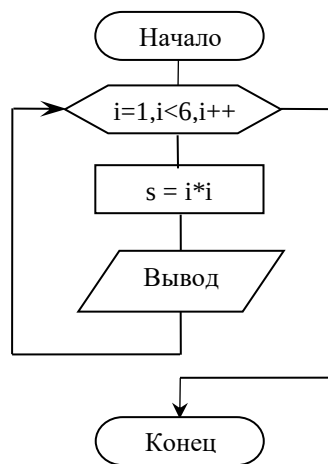
Модификации выполняются после каждой итерации цикла и служат обычно для изменения параметров цикла. В части модификации можно записать несколько операторов через запятую. Простой или составной *оператор* представляет собой тело цикла. Любая из частей оператора **for** может быть опущена (но точки с запятой надо оставить на своих местах!).

Значение переменной-счетчика можно использовать в теле цикла; изменение его значения тоже допускается, но считается плохим стилем программирования.

Замечание. Цикл с параметром не вызывает заикливания, так как предполагает конечное число повторений.

Для всех операторов цикла выход из цикла осуществляется как вследствие естественного окончания оператора цикла, так и с помощью операторов перехода (`goto` – безусловный переход, `continue` – переход к следующей итерации) и выхода (`break` – выход из цикла).

Пример: Вывести на экран таблицу квадратов первых пяти чисел.



```

//numbers_and_them_squares
#include <conio.h>
#include <iostream>
using namespace std;

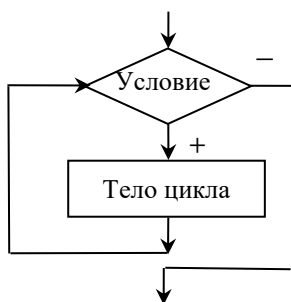
int main(){
    setlocale(LC_ALL, "Rus");
    cout<<"\nЧисла и их квадраты:\n ";
    for (int i=1; i<6; i++)
    {
        int s=i*i;
        cout<<"\n"<<i<<"^2 = "<<s;
    }
    _getch();
    return 0;
}
  
```

4.2. Оператор цикла с предусловием (while)

Формат оператора:

while (выражение) оператор;

Графическая интерпретация оператора:



Оператор **while** позволяет многократно выполнять одни и те же действия в зависимости от значения выражения (условия). Тип выражения должен быть арифметическим или приводимым к нему. Выражение вычисляется перед каждой итерацией цикла.

Работа оператора

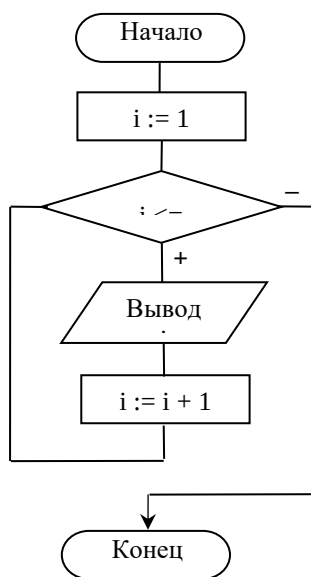
При входе в цикл вычисляется значение выражения (условия). Если при входе в цикл значение условия равно **false**, то вход в цикл не осуществляется и управление передается оператору, следующему непосредственно за оператором цикла. Если условие истинно (не равно **false**), то происходит вход в цикл и однократное выполнение тела цикла, представленного простым или составным оператором. Как только достигнут конец цикла, управление снова передается на его заголовок, где снова вычисляется значение выражения. Если значение выражения все еще равно **true**, то тело цикла выполняется еще один раз и т.

д. до тех пор, пока значение условия не станет равно **false**. Если оно постоянно будет равно **true**, то цикл будет бесконечным, т.е. произойдет заикливание.

Замечания:

- для того, чтобы тело цикла выполнилось хотя бы один раз, необходимо, чтобы перед выполнением тела цикла выражение (условие) было истинно;
- для того, чтобы цикл завершился, необходимо, чтобы в теле цикла изменялись значения переменных, входящих в условие.

Пример: Вывести на экран числа от 1 до 100.



```

//Schet_ot_1_do_100;
#include <conio.h>
#include <iostream>
using namespace std;

int main(){
    setlocale(LC_ALL, "Rus");
    cout<<"\nЧисла от 1 до 100:\n ";
    int i=1; //начальное значение параметра цикла
    while (i<=100) // перебираем первые 100 чисел
    {
        cout<<" "<<i<<" "; // вывод на экран значения i
        i++; // Увеличение параметра цикла на 1
    }
    _getch();
    return 0;
}
  
```

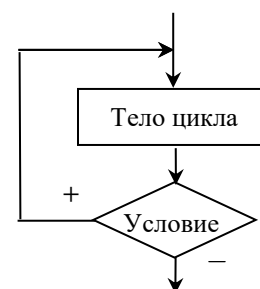
4.3. Оператор цикла с постусловием (do while)

Формат оператора:

```

do
    оператор;
while (выражение);
  
```

Графическая интерпретация оператора



Работа оператора

Сначала выполняется простой или составной оператор, составляющий тело цикла, а затем вычисляется выражение (условие). Если оно истинно (не равно **false**), тело цикла выполняется еще раз. Цикл завершается, когда выражение станет равным **false** или в теле цикла будет выполнен какой-либо оператор передачи управления. Тип выражения должен быть арифметическим или приводимым к нему.

Замечания:

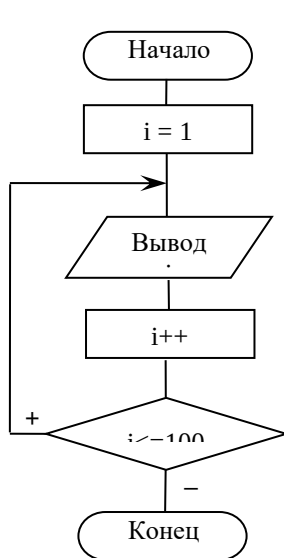
- последовательность инструкций между **do** и **while** всегда будет выполнена хотя бы один раз;

- для того, чтобы цикл завершился, необходимо, чтобы в теле цикла изменялись значения переменных, входящих в условие.

Отличие оператора цикла *while* от оператора цикла *do..while*

В операторе **while** тело цикла может не выполняться ни разу, если логическое выражение сразу окажется ложным.

Пример: Вывести на экран числа от 1 до 100.



```

//Schet_ot_1_do_100;
#include <conio.h>
#include <iostream>
using namespace std;

int main(){
    setlocale(LC_ALL, "Rus");
    cout<<"\nЧисла от 1 до 100:\n ";
    int i=1; //начальное значение параметра цикла
    do{
        cout<<" "<<i<<" "; //вывод на экран значения i
        i++; // Увеличение параметра цикла на 1
    } while (i<=100) //перебираем первые 100 чисел
    _getch();
    return 0;
}
  
```

4.4. Вложенные циклы

Тело цикла может содержать любой оператор, в том числе и оператор цикла. Структура цикла, содержащая вложенный цикл, называется *кратным циклом*. Число вложений может быть произвольным. Если цикл содержит один вложенный цикл, то он называется *двойным циклом* (рис. 4.2).

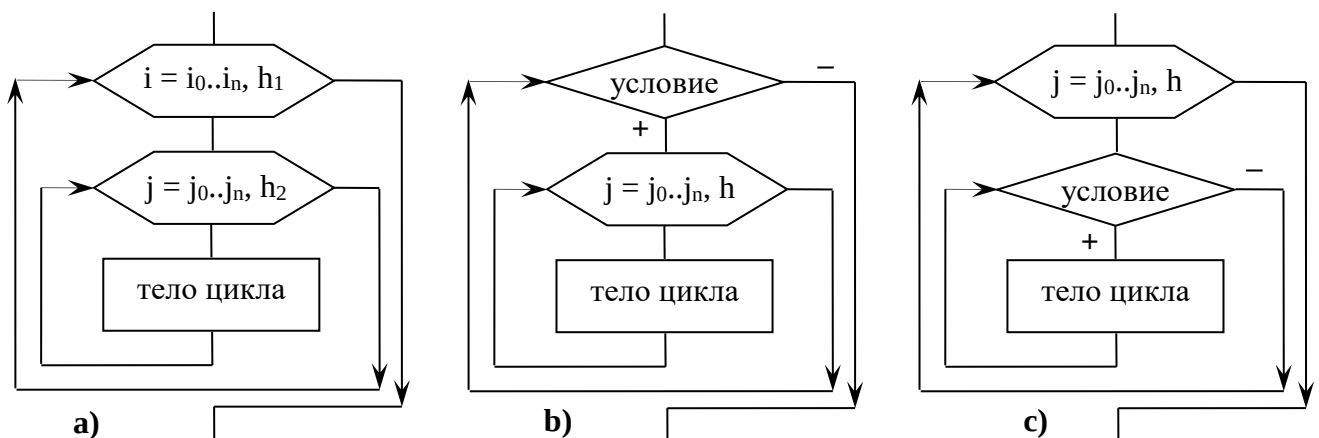


Рисунок 4.2 – Блок-схемы некоторых из возможных вариантов реализации двойного цикла

Цикл, который содержит вложенный цикл, называется *внешним*. В двойном цикле вложенный цикл называется *внутренним*. На рисунке 4.2.а предложена

блок-схема алгоритма, в котором для реализации и внешнего и внутреннего циклов использован оператор цикла с параметром (**for**). Такой вид двойного цикла используется при работе с двумерными массивами (таблицами). На рисунке 4.2.b для реализации внешнего цикла использован оператор цикла с предусловием (**while**), а внутреннего цикла – оператор цикла с параметром (**for**). На рисунке 4.2.c внешний цикл представлен оператором цикла с параметром (**for**), а внутренний – оператором цикла с предусловием (**while**).

Переменная внутреннего цикла всегда меняется быстрее, чем внешнего. Это означает, что для каждого значения внешней переменной цикла меняются все значения внутренней переменной.

Внешний и внутренний циклы могут использовать любой вид операторов цикла C++ (**while**, **do while**, **for**) (рис. 4.2).

Пример. Дано число вида $\overline{ab}$, где $a = 1..2$, $b = 0..9$. Вывести на экран все числа $\overline{ab}$.

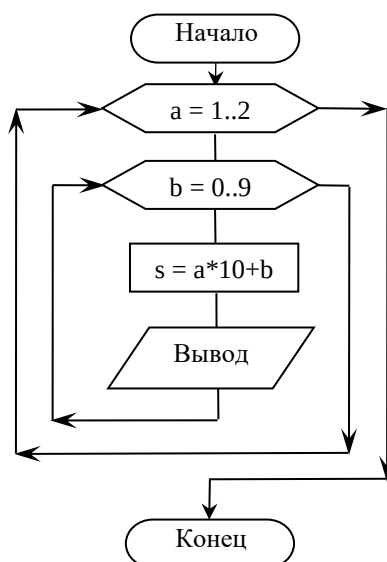
1. Математическая модель

Выпишем все эти числа в ряд:

10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29

Сначала увеличиваются единицы от 0 до 9, при этом значение десятков не меняется. Только когда перебор единиц завершён, десятки увеличиваются на 1. После этого опять происходит изменение единиц от 0 до 9. В переменной a – будут десятки, в b – единицы.

2. Блок-схема



3. Программа

```

#include <iostream>
#include <conio.h>
#include <math.h>

using namespace std;

int main()
{
    int s;

    for(int a=1;a<=2;a++)
        for(int b=1;b<=9;b++)
        {
            s = a*10+b;
            cout<<" "<<s;
        }

    _getch();
    return 0;
}
  
```

Результат работы программы

10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29

Практическая часть

ВЫПОЛНЕНИЕ РАБОТЫ: в соответствии с вариантом составить и реализовать программы (задание I- III).

Задание I

Используя оператор цикла с параметром, решить предложенную задачу:

1. Вычислить сумму и произведение первых n натуральных чисел. Предусмотреть ввод n с клавиатуры.
2. Вычислить $S = \sum_{n=1}^k (-1)^n n^2$, $k \geq 10$.
3. Вычислить сумму первых n членов ряда с общим членом $a_k = \frac{k+1}{\sqrt{k^2+2}}$, $k = \overline{1, n}$.
4. Определить средний рост девочек, и мальчиков одного класса. В классе учатся n учеников. ($n \geq 15$).
5. Вводя в цикле по 4 оценки каждого студента группы, подсчитать число студентов, не имеющих оценок 2 и 3. В группе учатся n студентов. Предусмотреть ввод n с клавиатуры.
6. Вводя в цикле по 4 оценки, полученные студентами в сессию, определить средний балл группы по всем экзаменам. В группе учатся n студентов. Предусмотреть ввод n с клавиатуры.
7. Задано n троек, чисел a , b , c . Вводя их по очереди с клавиатуры в порядке возрастания и интерпретируя их как длины сторон треугольника, определить, сколько троек может быть использовано для построения треугольника.
8. В ЭВМ по очереди поступают результаты соревнований по плаванию на дистанции 200 м, в которых участвует n спортсменов ($n > 10$). Выдать на экран дисплея лучший результат после ввода результата очередного спортсмена.
9. В ЭВМ вводятся по очереди координаты n точек ($n \geq 10$). Определить, сколько из них попадет в круг радиуса r с центром в точке (a, b) .
10. Ученикам первого класса назначается стакан молока (200 мл), если их вес составляет меньше 30 кг. Определить, сколько литров молока потребуется, ежедневно для одного класса, состоящего из n учеников. После взвешивания вес каждого ученика вводится в ЭВМ.
11. В ЭВМ вводятся номер по списку и рост учеников 10 класса. Вывести на экран номера по списку тех учеников, рост которых больше 170 см.
12. В соревнованиях по бегу на дистанции 100 м принимают участие n спортсменов. Вводя очереди номера по списку и результаты участников в ЭВМ, определить, сколько из них выполнили норму, и напечатать их номера (норма равна 13,2 сек.)

13. В ЭВМ вводятся по очереди координаты n точек. Определить, сколько из них попадет в кольцо с внутренним радиусом r_1 и внешним r_2 .
14. В ЭВМ по очереди вводятся координаты n точек. Определить, сколько из них принадлежит фигуре, ограниченной осью абсцисс и аркой синусоиды, построенной для аргумента со значениями от 0 до π .
15. Задана окружность радиуса r с центром в начале координат. Вводя последовательно в ЭВМ координаты n точек, являющихся центрами других окружностей того же радиуса, определить, сколько из этих окружностей пересекает данную.
16. Найти все трехзначные натуральные числа, сумма цифр которых равна их произведению.
17. Найти сумму целых положительных чисел, больших 20, меньших 100 и кратных 3.
18. Вычислить сумму $S = 1 + \frac{1}{2^3} + \frac{1}{3^3} + \dots + \frac{1}{n^3}$, где $n \geq 10$.
19. Вводя в цикле по 4 оценки, полученные студентами в сессию, определить число неуспевающих студентов группы. В группе учится n студентов. Предусмотреть ввод n с клавиатуры.
20. Цилиндр объема 1 куб.ед. имеет высоту h . Определить радиус основания для значений h от 1 до 6 с шагом 0.5 ед.
21. Вывести на печать последовательность с общим членом $a_k = \sin(k^2 + b)$, где $k = 0, 1, \dots, K$. Предусмотреть ввод K и b с клавиатуры. Вычислить количество элементов, удовлетворяющих условию: $0 \leq a_k \leq 0.5$.
22. В ЭВМ вводятся поочередно n точек. Определить, сколько из них принадлежит фигуре, ограниченной осью ординат и правой полуокружностью $x^2 + y^2 = r^2$.
23. В ЭВМ по очереди вводятся координаты n точек. Определить, сколько из них принадлежит фигуре, ограниченной линией $|x| + |y| = 1$.
24. Составить программу вычисления значений функции $p(x) = \frac{1}{16}(231x^6 - 315x^4 + 105x^2 - 5)$ при изменении x 0 до 1 с шагом 0.1.
25. Вычислить значение выражения $\frac{(x-2)(x-4)(x-6)\dots(x-64)}{(x-1)(x-3)(x-5)\dots(x-63)}$ для произвольного действительного числа x из области определения данного выражения.

Задание II

Двумя способами (используя, оператор цикла с предусловием и оператор цикла с постусловием) для данного значения x найти сумму ряда S с точностью до члена ряда, по абсолютной величине меньшего $\text{eps} = 0.0001$. Сравнить эту сумму со значением контрольной функции y [5].

1. $S = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \dots; x=1.5; y=e^x.$
2. $S = \sum_{n=0}^{\infty} \frac{x^n \ln^n 3}{n!} = 1 + \frac{x \ln 3}{1!} + \frac{x^2 \ln^2 3}{2!} + \dots; x=1; y=3^x.$
3. $S = \sum_{n=0}^{\infty} x^n = 1 + x + x^2 + x^3 + \dots; x=0.7; y=\frac{1}{1-x}.$
4. $S = \sum_{n=0}^{\infty} (-1)^n \frac{x^n}{2 \cdot 2^n} = \frac{1}{2} - \frac{x}{4} + \frac{x^2}{8} - \dots; x=1.2; y=\frac{1}{2+x}.$
5. $S = \sum_{n=1}^{\infty} \left(\frac{1}{2^n} + \frac{1}{3^n} \right) x^{n-1} = \frac{5}{6} + \frac{13}{36}x + \frac{35}{216}x^2 + \dots; x=-0.8; y=\frac{5-2x}{6-5x+x^2}.$
6. $S = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n}}{(2n)!} = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \dots; x=0.3; y=\cos x.$
7. $S = \sum_{n=1}^{\infty} (-1)^{n-1} \frac{x^n}{n} = x - \frac{x^2}{2} + \frac{x^3}{3} - \dots; x=0.4; y=\ln(1+x).$
8. $S = \sum_{n=1}^{\infty} (-1)^{n+1} \frac{\sin(nx)}{n} = \sin x - \frac{\sin 2x}{2} + \dots; x=-\frac{\pi}{2}; y=\frac{x}{2}.$
9. $S = \sum_{n=0}^{\infty} \frac{x^{2n+1}}{(2n+1)!} = x + \frac{x^3}{3!} + \dots; x=2; y=\frac{e^x - e^{-x}}{2}.$
10. $S = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{2n+1} = x - \frac{x^3}{3} + \frac{x^5}{5} - \dots; x=0.6; y=\operatorname{arctg} x.$
11. $S = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{(2n+1)!} = x - \frac{x^3}{3!} + \dots; x=1.4; y=\sin x.$
12. $S = \sum_{n=0}^{\infty} \frac{x^{2n}}{(2n)!} = 1 + \frac{x^2}{2!} + \frac{x^4}{4!} + \dots; x=0.5; y=\frac{(e^x + e^{-x})}{2}.$
13. $S = \sum_{n=1}^{\infty} \frac{\sin(2n-1)x}{2n-1} = \sin x + \frac{\sin 3x}{3} + \dots; x=\frac{\pi}{2}; y=\frac{\pi}{4}.$
14. $S = \sum_{n=1}^{\infty} (-1)^{n+1} \frac{\cos nx}{n} = \cos x - \frac{\cos 2x}{2} + \dots; x=\frac{\pi}{8}; y=\ln\left(2 \cos \frac{x}{2}\right).$
15. $S = \sum_{n=1}^{\infty} \frac{\cos nx}{n} = \cos x + \frac{\cos 2x}{2} + \dots; x=\pi; y=-\ln\left(2 \sin \frac{x}{2}\right).$

16. $S = 2 \sum_{n=1}^{\infty} \frac{x^{2n-1}}{2n-1} = 2 \left(x + \frac{x^3}{3} + \frac{x^5}{5} + \dots \right); x = -0.4; y = \ln \frac{1+x}{1-x}.$
17. $S = \sum_{n=1}^{\infty} \frac{1}{(2n-1)x^{2n-1}} = \frac{1}{x} + \frac{1}{3x^3} + \frac{1}{5x^5} + \dots; x = 3; y = \frac{1}{2} \ln \frac{x+1}{x-1}.$
18. $S = \sum_{n=1}^{\infty} \frac{1}{(2n-1)} \left(\frac{x-1}{x+1} \right)^{2n-1} = \frac{x-1}{x+1} + \frac{1}{3} \left(\frac{x-1}{x+1} \right)^3 + \dots; x = 0.6; y = \frac{1}{2} \ln x.$
19. $S = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n}}{n!} = 1 - \frac{x^2}{1!} + \frac{x^4}{2!} - \frac{x^6}{3!} + \dots; x = -0.7; y = e^{-x^2}.$
20. $S = \sum_{n=0}^{\infty} (n+1)x^n = 1 + 2x + 3x^2 + 4x^3 + \dots; x = -0.7; y = \frac{1}{(1-x)^2}.$
21. $S = \sum_{n=0}^{\infty} \frac{x^{2n+1}}{2n+1} = x + \frac{x^3}{3} + \frac{x^5}{5} + \frac{x^7}{7} + \dots; x = 0.35; y = \ln \sqrt{\frac{1+x}{1-x}}.$
22. $S = \sum_{n=1}^{\infty} \frac{(-1)^n}{n} (x+1)^{2n} = -(x+1)^2 + \frac{1}{2} (x+1)^4 + \dots; x = -1.2; y = \ln \frac{1}{2+2x+x^2}.$
23. $S = 1 - \sum_{n=1}^{\infty} (-1)^{n+1} \frac{1 \cdot 3 \cdot 5 \cdot \dots \cdot (2n-3)}{2 \cdot 4 \cdot 6 \cdot \dots \cdot 2n} x^n = 1 + \frac{1}{2} x - \frac{1}{2 \cdot 4} x^2 + \frac{1 \cdot 3}{2 \cdot 4 \cdot 6} x^3 - \dots;$
 $x = -0.8; y = \sqrt{1+x}.$
24. $S = 1 + \sum_{n=1}^{\infty} \frac{1 \cdot 3 \cdot \dots \cdot (2n-1)}{2 \cdot 4 \cdot \dots \cdot 2n} x^{2n} = 1 + \frac{1}{2} x^2 + \frac{1 \cdot 3}{2 \cdot 4} x^4 + \frac{1 \cdot 3 \cdot 5}{2 \cdot 4 \cdot 6} x^6 + \dots; x = 0.64;$
 $y = \frac{1}{\sqrt{1-x^2}}.$
25. $S = \sum_{n=1}^{\infty} \frac{\sin nx}{n} = \sin x + \frac{\sin 2x}{2} + \dots; x = \frac{\pi}{3}; y = \frac{\pi - x}{2}.$

Задание III

1. Какое количество теплоты потребуется для испарения 100 г воды при 100, 110, ..., 200°C?
2. Определите внутреннюю энергию 1 г гелия при температуре 10, 15, ..., 50°C.
3. В сосуде объёмом 1 см³ находится 10¹⁶ молекул газа. Определить давление газа в сосуде при температуре 0, 1, ..., 10°C.
4. Какова средняя квадратичная скорость молекул кислорода при 10, 12, ..., 20°C?

5. Сколько теплоты выделится при конденсации 100, 200, ..., 900 г водяного пара?
6. В котле ёмкостью 6.0 м³ содержится 25 кг перегретого пара. Каково давление пара, если его температура равна 200, 220, ..., 300°C?
7. Какова средняя квадратичная скорость молекул водорода при температуре 10, 15, ..., 50°C?
8. Определите внутреннюю энергию 1 г гелия при температуре 20, 22, ..., 40°C.
9. В сосуде объёмом 480 см³ при давлении $2.5 \cdot 10^4$ Н/м² находится газ. Подсчитать количество молекул газа для значений температуры 10, 11, ..., 20°C.
10. Сколько теплоты потребуется для испарения 200 г воды при температуре 100, 110, ..., 200°C?
11. Какова средняя квадратичная скорость молекул углекислого газа при температуре 0, 5, ..., 30°C ?
12. Определить среднюю квадратичную скорость молекул водорода при 0, 5, ..., 50°C.
13. В сосуде объёмом 450 см³ при давлении $2.5 \cdot 10^4$ Н/м² находится газ. Подсчитать количество молекул газа для значений температуры 10, 12, ..., 20°C.
14. Определите внутреннюю энергию 1 г водорода при температуре 20, 21, ..., 30°C.
15. Определить среднюю кинетическую энергию вращательного движения молекул водорода, содержащихся в 1.0 моль при 10, 11, ..., 20°C.
16. Плотность воздуха при нормальных условиях 1.3 г/л. Какова плотность воздуха при давлении $4.0 \cdot 10^5$ Н/м² и температуре 50, 60, ..., 100°C?
17. Определите внутреннюю энергию 1 г углекислого газа при температуре 10, 15, ..., 50°C.
18. В колбе содержится газ при нормальных условиях. Сколько молекул газа содержится в колбе, если её ёмкость равна 0.1, 0.2, ..., 0.5 л?
19. Какова средняя квадратичная скорость молекул водорода при 0, 2, ..., 20°C ?
20. Какова средняя квадратичная скорость молекул кислорода при температуре 0, 2, ..., 10°C?
21. В сосуде объёмом 400 см³ при давлении $2.5 \cdot 10^4$ Н/м² находится газ. Подсчитать количество молекул газа для значений температуры 0, 1, ..., 10°C.
22. Сколько теплоты потребуется для испарения 150 г воды при температуре 100, 105, ..., 150°C?
23. Определите внутреннюю энергию 1 г гелия при температуре 20, 21, ..., 30°C.
24. Какова средняя квадратичная скорость молекул углекислого газа при температуре 0, 5, ..., 30°C ?
25. Определите внутреннюю энергию 1 г кислорода при температуре 10, 15, ..., 50°C.

Контрольные вопросы

1. Какие базовые алгоритмические конструкции можно реализовать средствами языка C++?
2. Что называется циклом?
3. Что такое тело цикла?
4. Что такое параметр цикла?
5. Что такое итерация?
6. Что такое заикливание?
7. Какие типы циклов существуют?
8. Какие инструкции используются для реализации циклов на C++?
9. Какая инструкция используется для реализации цикла с параметром?
10. Можно ли использовать в теле цикла переменную, являющуюся параметром цикла?
11. Какая инструкция используется для реализации цикла с предусловием?
12. От чего зависит количество повторений тела цикла с предусловием?
13. Какая инструкция используется для реализации цикла с постусловием?
14. Какие из конструкций повторения могут привести к заикливанию?
15. Как работает цикл со счетчиком?
16. В каком случае в теле цикла используются операторные скобки?
17. Каковы отличия между циклами с пред- и с постусловием?

Пример выполнения лабораторной работы

Задание I. Составить программу табулирования функции

$$y = e^x \frac{2.42x^2 - 6.47x + 1.03}{x - 5.8} \text{ на отрезке } [-4, 4] \text{ с шагом } 0.2.$$

Решение

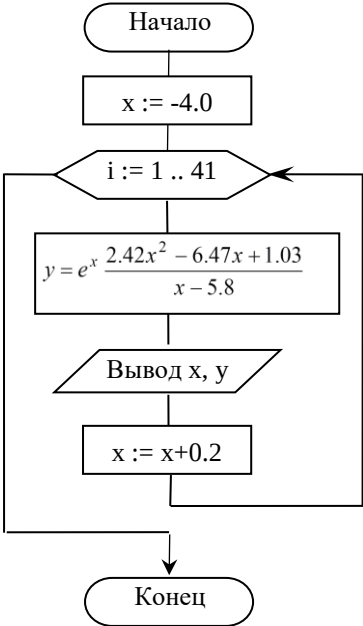
1. Математическая модель

Прежде всего, определим число повторений тела цикла: $S = \frac{4 - (-4)}{0.2} + 1 = 41.$

Аргументы: переменная x вещественного типа.

Результаты: значение заданной функции y (вещественного типа).

Промежуточные величины: счетчик i целого типа.

2. Алгоритм	3. Программа
 <pre> graph TD Start([Начало]) --> Init[x := -4.0] Init --> Loop{i := 1 .. 41} Loop --> Calc["y = e^x * (2.42x^2 - 6.47x + 1.03) / (x - 5.8)"] Calc --> Output[/Вывод x, y/] Output --> Inc[x := x + 0.2] Inc --> Loop Loop --> End([Конiec]) </pre>	<pre> #include <iostream> #include <conio.h> #include <math.h> using namespace std; int main() { double x = -4, y; for(int i=1;i<=41;i++) { y = exp(x)*(2.42*pow(x, 2.0) - 6.47*x+1.03)/(x-5.8); cout<<" x = "<<x<<" y = "<<y<<endl; x+=0.2; } _getch(); return 0; } </pre>

4. Результат работы программы

...

x = -3.6 y = -0.161865

x = -3.4 y = -0.185016

...

x = -2.2 y = -0.373639

...

Задание II. Для $x = 0.47$ найти сумму ряда

$$y = x + \left(\frac{1}{2} \cdot \frac{x^3}{3} + \frac{1 \cdot 3}{2 \cdot 4} \cdot \frac{x^5}{5} + \dots + \frac{1 \cdot 3 \cdot 5 \cdot \dots \cdot (2n-1)}{2 \cdot 4 \cdot 6 \cdot \dots \cdot 2n} \cdot \frac{x^{2n+1}}{2n+1} + \dots \right)$$

с точностью до члена ряда, меньшего (по абсолютной величине) $\text{eps} = 0.0001$, используя оператор цикла с предусловием. Сравнить полученную сумму со значением функции для контроля $y = \arcsin x$.

Решение

1. Математическая модель. Обозначим через $p1$ последовательные произведения нечетных чисел. Через $p2$ – последовательные произведения четных чисел.

В C++ (`math.h`), есть функция `arcsin x`, поэтому для вычисления контрольной функции воспользуемся соотношением: $y = \text{asin}(x)$.

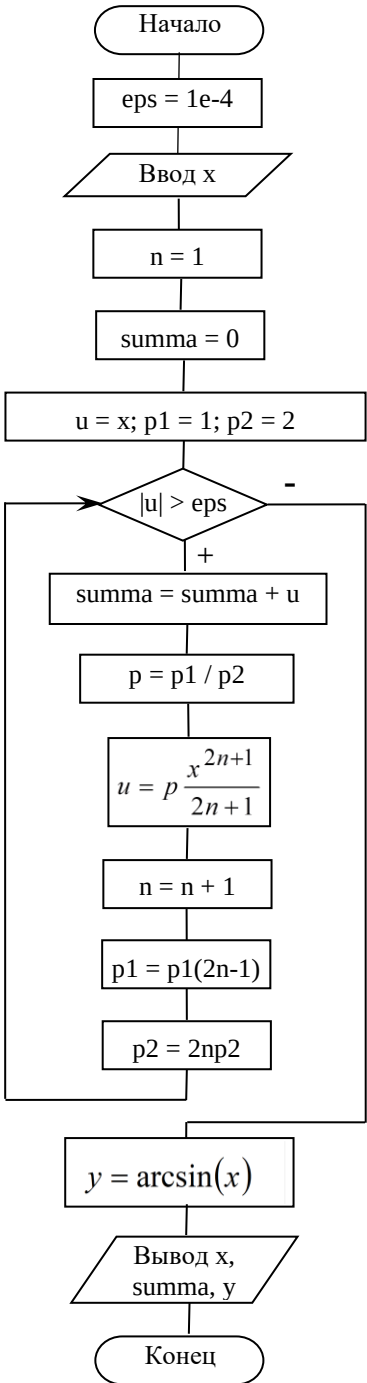
Аргументы: x – переменная вещественного типа, $-1 < x < 1$.

Результаты: `summa` – сумма членов ряда вещественного типа;
 y – значение контрольной функции (вещественного типа).

Промежуточные величины:

`eps` – константа, с которой сравнивается абсолютная величина текущего члена ряда: члены ряда представляют собой убывающую последовательность, поэтому со-

гласно условию вычисление суммы членов ряда будет продолжаться до тех пор пока абсолютное значение текущего члена ряда не окажется меньше eps;
 p1 – последовательные произведения нечетных чисел, вещественного типа;
 p2 – последовательные произведения четных чисел, вещественного типа;
 p – частное, получаемое при делении p1 на p2, вещественного типа;
 n – номер члена ряда, целого типа; u – текущий член ряда, вычисляемый по формуле $u = p \frac{x^{2n+1}}{2n+1}$, вещ. типа;

2. Алгоритм	3. Программа
 <pre> graph TD Start([Начало]) --> Eps[eps = 1e-4] Eps --> Input[/Ввод x/] Input --> N[n = 1] N --> Summa[summa = 0] Summa --> Init[u = x; p1 = 1; p2 = 2] Init --> Cond{ u > eps } Cond -- + --> SummaAdd[summa = summa + u] SummaAdd --> PDiv[p = p1 / p2] PDiv --> UCalc["u = p * (x^(2n+1) / (2n+1))"] UCalc --> NInc[n = n + 1] NInc --> P1Calc[p1 = p1 * (2n-1)] P1Calc --> P2Calc[p2 = 2n * p2] P2Calc --> Cond Cond -- - --> YCalc[y = arcsin(x)] YCalc --> Output[/Вывод x, summa, y/] Output --> End([Конец]) </pre>	<pre> // вычисление суммы членов ряда #include <iostream> #include <conio.h> #include <math.h> using namespace std; int main() { // описание и инициализация заданной точности double eps = 0.0001; // описание переменных, ввод x, начальные // установки double x, u, p1, p2, p; cout<<"Введите значение x: "; cin>>x; int n = 1; double summa = 0; u = x; p1 = 1; p2 = 2; // организация вычисления суммы членов ряда while(abs(u) > eps) { summa += u; p = p1/p2; u = p*pow(x, 2*n+1)/(2*n+1); n++; p1 = p1*(2*n-1); p2 = p2*2*n; } // вычисление значения контрольной функции double y = asin(x); // вывод результатов cout<<" Сумма данного ряда при x = "<<x<<" , с точностью 0.0001, S = "<< summa<<endl; cout<<" Значение контрольной функции, y = "<<y; _getch(); return 0; } </pre>

4. Результат работы программы

Введите значение x : 0.47

Сумма данного ряда при $x = 0.4700$ с точностью до 0.0001, $S = 0.4893$

Значение контрольной функции, $y = 0.4893$

Вывод: С точностью до 0.0001 значение суммы ряда и значение контрольной функции совпадают, следовательно, вычисление суммы членов ряда запрограммировано верно.

Замечание. Чтобы записать программу, используя цикл с постусловием, достаточно в приведенной ниже программе конструкцию **while** ($\text{abs}(u) \geq \text{eps}$) {тело цикла} заменить на **do** {тело цикла} **while** ($\text{abs}(u) \geq \text{eps}$);

СПИСОК ЛИТЕРАТУРЫ

1. Конспект лекций.
2. [2] стр. 44-51.

Лабораторная работа №5

Использование функций для решения прикладных задач

ЦЕЛЬ РАБОТЫ: закрепление теоретических знаний и приобретение навыков программирования функций.

Формируемые компетенции или их части: вырабатывается навык разработки математических и информационных моделей, а также формируется способность к разработке и применению алгоритмических и программных решений (с использованием функций) в области прикладного программирования (формируется часть компетенций ОПК-3 и ПК-7).

Теоретическая часть

Подпрограммы

5.1. Понятие подпрограммы

При решении задач часто встречаются случаи, когда для разных наборов исходных данных приходится выполнять одни и те же команды. В некоторых случаях для организации повторяющихся вычислений можно воспользоваться командой повторения. Иногда имеет смысл выделить повторяющиеся последовательности команд в самостоятельный блок, вынеся его за пределы основной программы и в нужном месте организовать вызов этого блока, указав соответствующий набор параметров. Такой блок, реализующий некоторый вполне законченный этап обработки информации и допускающий обращение к нему из различных частей основной программы, называется **подпрограммой**. Подпрограмма оформляется в виде замкнутого участка программы, имеющего чётко обозначенный вход и выход. Применение подпрограмм экономит оперативную память, т.к. для хранения переменных, используемых только в подпрограмме, память отводится только на время выполнения подпрограммы.

Большинство языков программирования содержат следующие разновидности подпрограмм:

- встроенные подпрограммы;
- библиотечные подпрограммы;
- подпрограммы, создаваемые программистом.

В С++ (как и в С) подпрограммы реализуются в виде функций. Различают библиотечные (стандартные) функции и функции, создаваемые программистом.

Библиотечные (стандартные) функции

С++ предоставляет стандартные функции, имеющие определённые имена и выполняющие стандартные действия. Эти функции, определённые в языке и объединены в заголовочные файлы или библиотеки. Чтобы использовать библиотечную функцию, необходимо подключить к программе соответствующий заголовочный файл. Чтобы вызвать такую функцию, достаточно в теле основной или пользовательской функции указать её имя и список фактических параметров.

Функции, создаваемые программистом

С++ позволяет программисту определить свою собственную функцию и в дальнейшем использовать её точно так же, как и библиотечные функции.

5.2. Формальные и фактические параметры

Подпрограмма (в случае C++ – функция) составляется формально. Используемые в ней переменные, называются **формальными**, так как они не существуют в том же качестве, как обычные переменные и константы.

Формальные параметры – это идентификаторы переменных, через которые передается информация из основной программы в подпрограмму и обратно. Их имена используются в теле подпрограммы, но на самом деле она будет обрабатывать фактические параметры, переданные при вызове, которые могут иметь совершенно другие имена. При вызове подпрограммы из основной программы формальные параметры заменяются **фактическими**, между ними устанавливается взаимно однозначное соответствие, удовлетворяющее одному условию: соответствующие друг другу параметры должны совпадать по порядку следования и по типу. Блок подпрограммы выполняется для фактического набора параметров.

Фактические параметры – это информация о том, какие значения нужно передать аргументам подпрограммы и каким именам программы передать значения результатов подпрограммы.

Описание подпрограммы задает формальную схему обработки информации, в то время как оператор вызова подпрограммы осуществляет настройку этой формальной схемы на конкретную обработку фактических данных.

5.3. Локальные и глобальные переменные

Переменная – это именованная область памяти, в которой хранятся данные определенного типа. У переменной есть имя и значение. Имя служит для обращения к области памяти, в которой хранится значение. Во время выполнения программы значение переменной можно изменять. Перед использованием любая переменная должна быть описана.

Описание переменной кроме типа и класса памяти, явно или по умолчанию задает ее область действия. Класс памяти и область действия зависят не только от собственно описания, но и от места ее размещения в тексте программы.

Область действия идентификатора – это часть программы, в которой его можно использовать для доступа связанной с ним области памяти. В зависимости от области действия переменная может быть **локальной** или **глобальной**.

Если переменная определена внутри блока (напомню, что блок ограничен фигурными скобками), она называется **локальной**, область ее действия – от точки описания до конца блока, включая все вложенные блоки. Если переменная определена вне любого блока, она называется **глобальной** и областью ее действия считается файл, в котором она определена, от точки описания до его конца.

Класс памяти определяет время жизни область видимости программного объекта (в частности переменной). Если класс памяти не указан явным образом, он определяется компилятором исходя из контекста объявления.

Время жизни может быть постоянным (в течение выполнения блока).

Областью видимости идентификатора называется часть текста программы, из которой допустим обычный доступ к связанной с идентификатором области памяти. Чаще всего область видимости совпадает с областью действия. Исключением является ситуация, когда во вложенном блоке описана переменная с таким

же именем. В этом случае внешняя переменная во вложенном блоке невидима, хотя он и входит в ее область действия. Тем не менее, к этой переменной, если она глобальна, можно обратиться, используя операцию доступа к области видимости «::».

Для задания класса памяти используются следующие спецификаторы:

- **auto** – автоматическая переменная. Память под нее выделяется в стеке и при необходимости инициализируется каждый раз при выполнении оператора, содержащего ее определение. Освобождение памяти происходит при выходе из блока, в котором описана переменная. Время ее жизни – с момента описания до конца блока. Для глобальных переменных этот спецификатор не используется, а для локальных он принимается по умолчанию, поэтому задавать его явным образом большого смысла не имеет.
- **extern** – означает, что переменная определяется в другом месте программы (в другом файле или дальше по тексту). Используется для создания переменных, доступных во всех модулях программы, в которых они объявлены.
- **static** – статическая переменная. Время жизни – постоянное. Инициализируется один раз при первом выполнении оператора, содержащего определение переменной. В зависимости от расположения оператора описания статические переменные могут быть глобальными и локальными. Глобальные статические переменные видны только в том модуле, в котором они описаны.
- **register** – аналогично auto, но память выделяется по возможности в регистрах процессора. Если такой возможности у компилятора нет, переменные обрабатываются как auto.

```
int a;                \\ 1 глобальная переменная a
int main()
{
    int b;            \\ 2 локальная переменная b
    extern int x;      \\ 3 переменная x определена в другом месте
    static int c;      \\ 4 локальная статическая переменная c
    a = 1;             \\ 5 присваивание глобальной переменной
    int a;             \\ 6 локальная переменная a
    a = a;             \\ 7 присваивание локальной переменной
    ::a = 3;           \\ 8 присваивание глобальной переменной
    return 0;
}
int x = 4;            \\ 9 определение и инициализация x
```

В этом примере глобальная переменная **a** определена вне блоков. Память под нее выделяется в сегменте данных в начале работы программы, областью действия является вся программа. Область видимости – вся программа, кроме строк 6-8, так как в первой из них определяется локальная переменная с тем же именем, область действия начинается с точки ее описания и заканчивается при выходе из блока. Переменные **b** и **c** – локальные, область их видимости – блок, но время жизни различно: память под **b** выделяется в стеке при входе в блок и осво-

бождается при выходе из него, а переменная с располагается в сегменте данных и существует все время, пока работает программа.

Если при определении начальное значение переменных явным образом не задается, компилятор присваивает глобальным и статическим переменным нулевое значение соответствующего типа. Автоматические переменные не инициализируются.

Имя переменной должно быть уникальным в своей области действия (например, в одном блоке не может быть двух переменных с одинаковыми именами).

Для определения области действия идентификаторов надо придерживаться правил:

- каждый идентификатор должен быть описан перед тем, как будет использован;
- областью действия идентификатора является блок, в котором он описан;
- все идентификаторы в блоке должны быть уникальными, т.е. не повторяться;
- один и тот же идентификатор может быть по-разному определен в каждом отдельном блоке.

Если идентификатор пользовательской функции совпадает с именем стандартной функции, то последняя недоступна в области действия функции, объявленной пользователем, т.е. стандартная функция игнорируется, а выполняется функция пользователя.

5.4. Функции

Под **функцией** понимается совокупность операторов, совместно выполняющих некоторое действие, имеющее определенную цель, оформленная в соответствии с требованиями языка программирования.

С понятием функции в языке C++ связано три следующих компонента:

- описание функции;
- прототип;
- вызов функции.

Описание функции состоит из двух частей: заголовка и тела. Описание функции имеет следующую форму записи:

```
тип_результата имя_функции ([список_параметров])
{
    /* объявления и операторы
       или, другими словами, тело функции*/
}
```

Здесь тип_результата – тип возвращаемого значения. В случае отсутствия спецификатора типа предполагается, что функция возвращает целое значение (**int**). Если функция не возвращает никакого значения, то на месте типа записывается спецификатор **void**. В списке параметров для каждого параметра должен быть указан тип. При отсутствии параметров список может быть пустым или иметь спецификатор **void**.

Тело функции представляет собой последовательность объявлений и операторов, описывающих определенный алгоритм. Важным оператором тела функ-

ции является оператор возврата в точку вызова: **return [выражение];**. Оператор **return** имеет двойное назначение. Он обеспечивает немедленный возврат в вызывающую функцию и может использоваться для передачи вычисленного значения функции. В теле функции может быть несколько операторов **return**, но может не быть и ни одного. В последнем случае возврат в вызывающую программу происходит после выполнения последнего оператора тела функции.

Прототип функции может указываться до вызова функции вместо описания функции для того, чтобы компилятор мог выполнить проверку соответствия типов аргументов и параметров. Прототип функции по форме такой же, как и заголовков функции, в конце его ставится «;». Параметры функции в прототипе могут *иметь имена, но компилятору они не нужны*.

Компилятор использует прототип функции для сравнения типов аргументов с типами параметров. Язык Си не предусматривает автоматического преобразования типов в случаях, когда аргументы не совпадают по типам с соответствующими им параметрами, т. е. язык Си обеспечивает строгий контроль типов. При наличии прототипа вызываемые функции не обязаны размещаться в одном файле с вызывающей функцией. То есть, описание тел функций можно разместить в другом файле, который необходимо подключить к исходному с использованием директивы **#include**.

Вызов функции может быть оформлен в виде оператора, если у функции *отсутствует* возвращаемое значение, или в виде выражения, если *существует* возвращаемое значение.

В первом случае оператор имеет следующий формат:

имя_функции ([список_аргументов]);

Например: Print("Hello World!");

Во втором случае выражение записывается следующим образом:

Переменная = имя_функции ([список_аргументов]);

Например: z = max(x, y);

Значение вычисленного выражения является возвращаемым значением функции. Возвращаемое значение передается в место вызова функции и является результатом ее работы.

Число и типы фактических параметров (аргументов) должны совпадать с числом и типом формальных параметров (параметров) функции. При вызове функции фактические параметры (аргументы) подставляются вместо формальных параметров (параметров). Таким образом, описание функции задает формальную схему обработки информации, в то время как оператор вызова функции осуществляет настройку этой формальной схемы на конкретную обработку фактических данных.

Ниже приводится пример функции с возвращаемым значением.

Пример 1. Составить программу, содержащую обращение к функции вычисления максимума из двух чисел.

Возможное решение данной задачи имеет вид:

//описание функции max находится в данном файле ниже тела главной функции

```

#include <conio.h>
#include <iostream>
using namespace std;

// описание пользовательской функции
int max (int a, int b)      // заголовок функции
{
    int c;                  //
    c=(a>b)?a:b;            // тело функции
    return c;              //
}

void main()
{
    setlocale(LC_ALL,"Rus");
    int x,y,z;
    cout << "\n поочередно введите x и y \n";
    cin >> x >> y;
    z = max (x, y);          // вызов функции
    cout << "z=" << z;
    return;
}

```

Описание функции находится в одном файле с главной программой.

Вызов функции является выражением в правой части оператора присваивания $z = \max(x, y)$; при выполнении которого значения аргументов x и y подставляются вместо параметров a и b соответственно (передача параметров в функцию по значению). После выполнения тела функции возвращаемое значение передается в место вызова функции и присваивается переменной z .

В рассмотренной программе функция имеет возвращаемое значение. Существуют задачи, которые не требуют передачи возвращаемого значения. Такой пример приведен ниже.

Пример 2. Использование функции без возвращаемого значения.

Составить программу, включающую в свой состав функцию, при обращении к которой обеспечивается выдача на экран символа, набранного на клавиатуре. При нажатии клавиши Enter выполнение программы завершить.

```

...
void print(char);          // прототип функции

void main()
{
    char x;
    cout << "\nВводите символы."<< " Enter – окончание ввода";
    do
    {
        x = getch();
        print (x);
    }
    while (x! == 13);
    return 0;
}

```

```
void print(char a) { cout <<a << "\t"; }
```

В функции print() отсутствует возвращаемое значение, поэтому обращение к функции осуществляется оператором вызова функции print(x);. В результате обращения к функции на экран будет выведен символ, введенный с клавиатуры с помощью оператора `x = getch()`.

Практическая часть

ВЫПОЛНЕНИЕ РАБОТЫ: освоить теоретический материал, выполнить общее для всех задание I и в соответствии с вариантом составить программу (задание II), реализовав пользовательскую функцию.

Задание I

Изучить порядок объявления и описания функций:

- реализовать (отладить и запустить программы) приведенные в теоретических сведениях примеры 1 – 2;
- оформить в тетради, записав условие, блок-схему и результат работы соответствующих программ (примеры 1 – 2).

Задание II

В соответствии с вариантом составить и реализовать программу:

1. Три прямоугольных треугольника заданы катетами. Определить, площадь какого из них больше.
2. Три отрезка заданы координатами вершин. Определить, есть ли среди них отрезки равной длины.
3. В порт в среднем приходит три корабля в день. Какова вероятность того, что в порт придёт 2 корабля? 4 корабля? Вычисление вероятности производится по формуле: $P = 3^k e^{-3} / k!$.
4. Найти среднее арифметическое действительных корней трёх квадратных уравнений.
5. Даны три различных бруска, имеющих форму кубов с заданным произвольным ребром. Из каждого выпиlena правильная четырехугольная пирамида (основание – грань куба). Вычислить объем материала, ушедшего в отходы.
6. Даны действительные числа a и b . Вычислите,

$$g(1.2, a) + g(b, a) - g(2a - 1, ab), \text{ где } g(x, y) = \frac{(x^2 + y^2)}{x^2 + 2xy + 3y^2 + 4}$$

7. Заданы длины рёбер произвольной треугольной пирамиды. Вычислить площадь полной поверхности этой пирамиды.
8. Вычислить объем «снеговика», состоящего из трех шаров с произвольными радиусами.

9. Каждое из трех колец задано внутренним и внешним радиусами. Чему равна площадь наибольшего кольца?
10. Вычислить значение выражения $a! + b! + c! + d!$.
11. Вычислить площадь «елки», состоящей из четырех треугольников с известными сторонами.
12. Найти минимальное значение среди корней трех квадратных уравнений (каждое уравнений задано тройкой коэффициентов).
13. Даны действительные числа a и b . Вычислить

$$\max(u + v^2, 3.14),$$

где $u = \max(a, b)$, $v = \max(ab, a + b)$, а $\max$ – подпрограмма для вычисления максимального значения для двух данных чисел.

14. Даны радиусы трех шаров. Вычислить среднее арифметическое объемов этих шаров.
15. Дана арифметическая прогрессия a_n , a_1 – первый ее элемент, d – разность. Найти второй, третий, четвертый и седьмой элементы и их сумму.
16. Даны действительные числа s и t . Вычислить $f(t, -2s, 1.17) + f(2.2, t, s - t)$,

$$f(x, y, z) = \frac{2x - y - \sin z}{5 + |z|}.$$

17. Найти наибольшее из пяти заданных чисел, используя подпрограмму нахождения наибольшего из двух чисел.
18. Вычислить значение функции:

$$y = \varphi(3) + \varphi(7.6) \cdot (\varphi(11.2) - \varphi(5)), \text{ если } \varphi(x) = \sqrt{2x^2 - 4x + e^x}$$

19. Вычислить значение функций

$$f(x) = 2 \sin x - 4.2 \cos x + e^x \text{ и } g(x) = \frac{2x^2}{10\sqrt{5x}}$$

для значений $x = 0,1; 0,35; 0,53$.

20. Вычислить значение функции $x = 2f(15) - 4f(-1)g(11) + g(32)$, если $f(x) = 10x + e^{2x}$; $g(x) = \sqrt{2x + x^3}$.
21. Из куба со стороной a выпилили куб со стороной b ($a > b$). Какой процент материала ушёл в отходы?
22. Каждая из трёх окружностей задана координатами центра и координатами точки на окружности. Какова наибольшая длина окружности?
23. Найти наименьшее из четырёх заданных чисел, используя подпрограмму нахождения наименьшего из двух чисел.
24. Даны действительные числа a и b . Вычислить

$$\min(2u - v^2, 4.5),$$

где $u = \min(a, b)$, $v = \min(a/b, a - b)$, а $\min$ – подпрограмма для вычисления минимального значения для двух данных чисел.

25. Вычислить большие вещественные корни квадратных уравнений $ax^2 + bx + c = 0$ и $tx^2 + px + q = 0$, если они существуют. В противном случае вывести на экран монитора соответствующее сообщение.

Контрольные вопросы

1. Что такое подпрограмма?
2. Как подпрограмму можно реализовать в C++?
3. Какие параметры называются формальными?
4. Какие параметры называются фактическими?
5. Что такое переменная?
6. Что такое область действия идентификатора?
7. Что такое локальная переменная? Что такое глобальная переменная?
8. Какова область действия локальных идентификаторов?
9. Какова область действия глобальных идентификаторов?
10. Что такое функция?
11. Чем отличается объявление функции от ее определения?
12. Перечислите составные части описания функции.
13. Что такое прототип функции?
14. Как описывается функция пользователя в программе?
15. Что такое возвращаемое значение функции?
16. Как осуществляется вызов функций из основной программы?

Пример выполнения лабораторной работы

Задание. Задавая с клавиатуры координаты вершин треугольника, определить, является ли он равносторонним.

Решение

1. Математическая модель

Треугольник является равносторонним, если длины его сторон равны. Находим длины сторон треугольника, используя подпрограмму нахождения расстояния между двумя точками.

Аргументы: $a_1, b_1, c_1, a_2, b_2, c_2$ целого типа – координаты вершин треугольника.

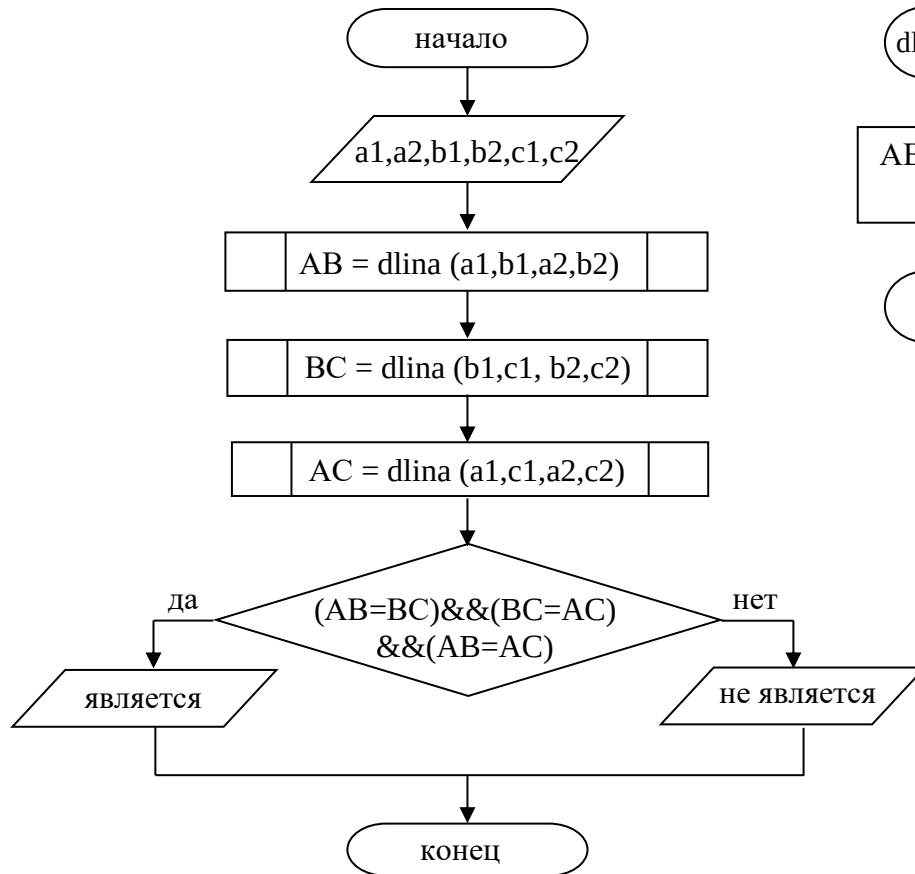
Результаты: текстовое сообщение.

Промежуточные величины: AB, BC, AC вещественного типа – длины сторон треугольника.

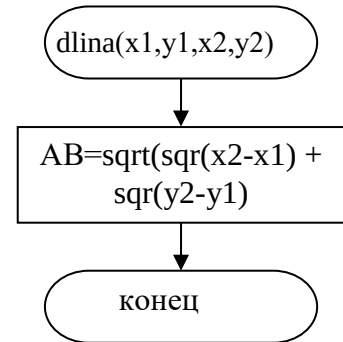
Расстояние между двумя точками $A(x_1, y_1)$ и $B(x_2, y_2)$ находим по формуле:

$$AB = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}.$$

2. Алгоритм – основная программа:



– подпрограмма:



3. Программа

```
#include <iostream>
#include <conio.h>
#include <math.h>
const double eps = 0.01;
using namespace std;

double Dlina (double x1,double y1,double x2,double y2) {
    return (sqrt((x1-y1)*(x1-y1)+(x2-y2)*(x2-y2)));
}

int main(){
    cout<<endl<<"Задайте координаты точек A, B, C"<<endl;
    double a1, a2, b1, b2, c1, c2;
    cin>>a1>>a2>>b1>>b2>>c1>>c2;

    double ab, bc, ac;
    ab = Dlina(a1, b1, a2, b2);
    bc = Dlina(b1, c1, b2, c2);
    ac = Dlina(a1, c1, a2, c2);

    if((fabs(ac-bc)<eps) && (fabs(bc-ab)<eps) && (fabs(ac-ab)<eps))
        cout<<endl<<"Треугольник является равносторонним";
    else cout<<endl<<"Треугольник не является равносторонним";
    _getch();
    return 0;
}
```

4. Результат работы программы:

Задайте координаты точек А, В, С

0 0 6 6 5 1

Треугольник не является равносторонним

СПИСОК ЛИТЕРАТУРЫ

1. Конспект лекций
2. [2] стр. 73-78.

Лабораторная работа №6

Программирование рекурсивных алгоритмов

ЦЕЛЬ РАБОТЫ: закрепление навыков программирования с использованием рекурсивных функций.

Формируемые компетенции или их части: вырабатывается навык разработки математических и информационных моделей, а также формируется способность к разработке и применению рекурсивных алгоритмических и программных решений в области прикладного программирования (формируется часть компетенций ОПК-3 и ПК-7).

Теоретическая часть

Итерация и рекурсия

6.1. Понятие итеративного процесса

Итеративный процесс можно определить как процесс вычислений, основанный на повторении последовательности операций, при котором на каждом шаге повторения используется результат предыдущего шага. Примерами итеративных алгоритмов могут служить алгоритмы вычисления факториала натурального числа, последовательности чисел Фибоначчи, вычисление суммы членов последовательности с заданным числом слагаемых и т.д. Итеративные процессы программируются обычно с помощью циклических конструкций.

Пример. Вычисление факториала числа n .

```
int factor (int n)
{
    int f = 1;
    for (int i = 2; i <= n; i++) f *= i;
    return k;
}
```

6.2. Понятие рекурсии

В то же время повторение последовательности действий можно осуществить и другим методом, основанным на принципе возможности явного включения в конструкции языка программирования конструкции того же вида. В частности, в языке программирования C++ всякая функция может вызывать сама себя. Такой вызов называется **рекурсией**. Функция, вызывающая саму себя, называется **рекурсивной**.

Вообще алгоритм решения задачи называется *рекурсивным*, если он выражается в терминах решения более простой версии той же задачи и, кроме того, в нем предусмотрено решение самого простого варианта задачи.

При рекурсивном вызове функций в C++ используется стековая организация хранения локальных параметров.

Стек является простейшей динамической структурой организации данных. Добавление элементов в стек и выборка из него выполняются из одного конца,

называемого вершиной стека. Говорят, что стек реализует принцип обслуживания LIFO (last in — first out, последним пришел - первым ушел (обслужен)).

Стеки широко применяются в системном программном обеспечении, компиляторах, в различных рекурсивных алгоритмах.

Различия между итерацией и рекурсией:

- 1) итерации необходим цикл и вспомогательные величины, в то время как рекурсия обходится без вспомогательных величин и обычно проще для понимания, короче и нагляднее итерации;
- 2) итерация требует меньше места в оперативной памяти и меньших затрат машинного времени, чем рекурсия, которой необходимы затраты на управление стеком;
- 3) при реализации рекурсии велика вероятность переполнения стека.

Поэтому при выборе способа описания функции следует вначале решить, чему отдать предпочтение – эффективности программы или ее компактности. Однако **следует избегать рекурсий там, где есть очевидное итерационное решение**. Вместе с тем существует много хороших примеров применения рекурсии (построение графических образов разнообразных узоров, работа с двоичными деревьями).

При рекурсивном описании функции последняя всегда должна содержать, по крайней мере, одну альтернативную нерекурсивную ветвь алгоритма с условием окончания вычислений, заканчивающуюся оператором возврата результата.

Различают две формы рекурсивных подпрограмм:

- **прямая рекурсия;**
- **косвенная рекурсия.**

В первом случае функция содержит вызов этой же функции.

Косвенная рекурсия возникает в том случае, когда две или более функций вызывают друг друга. В случае двух функций: некоторая функция обращается к другой функции, а та в свою очередь вызывает первоначальную функцию. Для предотвращения закливания одна или обе функции должны проверять условие завершения.

Пример: Составить рекурсивную функцию, вычисляющую факториал числа n по формуле (учесть, что $0! = 1$):

$$n! = \begin{cases} 1, & n = 1; \\ n \cdot (n-1)!, & n > 1. \end{cases}$$

```
long fact (long n) {  
    if (n==0 || n == 1) return 1;  
    return (n*fact(n-1));  
}
```

То же самое можно записать короче:

```
long fact (long n) {  
    return (n > 1) ? n*fact(n-1) : 1;  
}
```

Пример: Распечатать последовательность, состоящую из n букв А и n букв В.

```
void abn (int n){  
    cout<<"A";  
    if (n > 1) abn(n-1);  
    cout<<"B";  
    return  
}
```

Практическая часть

ВЫПОЛНЕНИЕ РАБОТЫ: в соответствии с вариантом, используя прямую рекурсию, составить и реализовать программу.

Задание

1. Описать рекурсивную функцию $NOD(A, B)$ целого типа, находящую наибольший общий делитель (НОД) двух целых положительных чисел A и B , используя алгоритм Евклида: $НОД(A, B) = НОД(B, A \bmod B)$, если $B \neq 0$; $НОД(A, 0) = A$. С помощью этой функции найти $НОД(A, B)$, $НОД(A, C)$, $НОД(A, D)$, если даны числа A, B, C, D .
2. Для $n = 12$ найти числа Фибоначчи. Числа Фибоначчи: $F(0) = 1, F(1) = 1$
 $F(n) = F(n - 2) + F(n - 1)$
3. Даны целые числа m и n , где $0 \leq m \leq n$, вычислить, используя рекурсию, число сочетаний $C(n, m)$ по формуле: $C_n^0 = C_n^n = 1, C_n^m = C_{n-1}^m + C_{n-1}^{m-1}$ при $0 \leq m \leq n$. Воспользовавшись формулой $C_n^m = \frac{n!}{m!(n-m)!}$ можно проверить правильность результата.
4. Опишите рекурсивную функцию, которая по заданным вещественному x и целому n вычисляет величину x^n согласно формуле:

$$x^n = \begin{cases} 1, & n = 0, \\ \frac{1}{x^{|n|}}, & n < 0, \\ x \cdot (x^{n-1}), & n > 0. \end{cases}$$

5. Задана последовательность положительных чисел, признаком конца которых служит отрицательное число. Используя рекурсию, подсчитать количество чисел и их сумму.
6. Напишите рекурсивную функцию $K(n)$, которая возвращает количество цифр в заданном натуральном числе n , используя формулу:

$$K(n) = \begin{cases} 1, & \text{если } n < 10, \\ K(n \div 10) + 1, & \text{если } n \geq 10. \end{cases}$$

7. Определим функцию $S(n)$, вычисляющую сумму цифр заданного натурального числа:

$$S(n) = \begin{cases} n, & \text{если } n < 10, \\ S(n \operatorname{div} 10) + n \bmod 10, & \text{если } n \geq 10. \end{cases}$$

Описать рекурсивную функцию DigitSum(K) целого типа, которая находит сумму цифр целого числа K, не используя оператор цикла. С помощью этой функции найти суммы цифр для пяти данных целых чисел.

8. Дан вектор X из n вещественных чисел. Найти минимальный элемент вектора, используя вспомогательную рекурсивную функцию, находящую минимум среди элементов вектора X, начиная с n -го.
9. Напишите рекурсивную функцию для нахождения биномиальных коэффициентов (для заданного $M \geq i \geq j > 0$ вычислите все C_i^j):

$$C_n^m = \begin{cases} 1, & \text{если } n = m \text{ или } m = 0; \\ C_{n-1}^{m-1} + C_{n-1}^m. \end{cases}$$

10. Напишите программу вычисления функции Аккермана для всех неотрицательных целых аргументов m и n :

$$A(m, n) = \begin{cases} A(0, n) = n + 1, \\ A(m, 0) = A(m - 1, 1), m > 0, \\ A(m, n) = A(m - 1, A(m, n - 1)), m, n > 0. \end{cases}$$

11. Для заданных границ интегрирования a и b вычислите значение определенного интеграла следующего вида:

$$\int e^{ax} \cos^n x dx = \begin{cases} \frac{e^{ax} \cos^{n-1} x (a \cos x + n \sin x)}{a^2 + n^2} + \frac{n(n-1)}{a^2 + n^2} \int e^{ax} \cos^{n-2} x dx, & n \geq 2, \\ \frac{-e^{ax} (\sin x + a \cos x)}{a^2 + n^2}, & n = 1, \\ \frac{e^{an}}{a}, & n = 0. \end{cases}$$

12. Для заданных границ интегрирования a и b вычислите значение определенного интеграла следующего вида:

$$\int \cos^n x dx = \begin{cases} \frac{\cos^{n-1} x \sin x}{n} + \frac{n-1}{n} \int \cos^{n-2} x dx, & n > 2, \\ \frac{x}{2} + \frac{1}{4} \sin 2x, & n = 2, \\ \sin x, & n = 1. \end{cases}$$

13. Для заданных границ интегрирования a и b вычислите значение определенного интеграла следующего вида:

$$\int \sin^n x dx = \begin{cases} -\frac{\sin^{n-1} x \cos x}{n} + \frac{n-1}{n} \int \sin^{n-2} x dx, n > 2, \\ \frac{x}{2} - \frac{1}{4} \sin 2x, n = 2, \\ -\cos x, n = 1. \end{cases}$$

14. Для заданных границ интегрирования a и b вычислите значение определенного интеграла следующего вида:

$$\int \frac{dx}{\sin^n x} = \begin{cases} -\frac{1}{n-1} \cdot \frac{\cos x}{\sin^{n-1} x} + \frac{n-2}{n-1} \int \frac{dx}{\sin^{n-2} x}, n \geq 2, \\ \ln \operatorname{tg} \frac{x}{2}, n = 1, \\ x, n = 0; \end{cases}$$

15. Для заданных границ интегрирования a и b вычислите значение определенного интеграла следующего вида:

$$\int \frac{dx}{\cos^n x} = \begin{cases} \frac{1}{n-1} \cdot \frac{\sin x}{\cos^{n-1} x} + \frac{n-2}{n-1} \int \frac{dx}{\cos^{n-2} x}, n \geq 2, \\ \ln \operatorname{tg} \left(\frac{\pi}{4} + \frac{x}{2} \right), n = 1, \\ x, n = 0. \end{cases}$$

16. Для заданных границ интегрирования a и b вычислите значение определенного интеграла следующего вида:

$$\int x^n e^{ax} dx = \begin{cases} \frac{x^n e^{ax}}{a} - \frac{n}{a} \int x^{n-1} e^{ax} dx, n > 1, \\ \frac{e^{ax}}{a^2} (ax - 1), n = 1. \end{cases}$$

17. Для заданных границ интегрирования a и b вычислите значение определенного интеграла следующего вида:

$$\int x^n a^{mx} dx = \begin{cases} \frac{x^n a^{mx}}{n \ln a} - \frac{n}{m \ln a} \int x^{n-1} a^{mx} dx, n > 1, \\ \frac{x a^{mx}}{m \ln a} - \frac{a^{mx}}{m (\ln a)^2}, n = 1. \end{cases}$$

18. Для заданных границ интегрирования a и b вычислите значение определенного интеграла следующего вида:

$$\int \ln^n x dx = \begin{cases} x \ln^n x - n \int \ln^{n-1} x dx, & n > 1, \\ x \ln x - x, & n = 1. \end{cases}$$

19. Для заданных границ интегрирования a и b вычислите значение определенного интеграла следующего вида:

$$\int x^m \ln^n x dx = \begin{cases} \frac{x^{m+1}}{m+1} \ln^n x - \frac{n}{m+1} \int x^m \ln^{n-1} x dx, & n > 1, \\ x^{m+1} \left[\frac{\ln x}{m+1} - \frac{1}{(m+1)^2} \right], & n = 1. \end{cases}$$

20. Для заданных границ интегрирования a и b вычислите значение определенного интеграла следующего вида:

$$\int \frac{dx}{(a^2 + x^2)^n} = \begin{cases} \frac{1}{2(n-1)a^2} \left[\frac{x}{(a^2 + x^2)^{n-1}} + (2n-3) \int \frac{dx}{(a^2 + x^2)^{n-1}} \right], & n > 1, \\ \frac{1}{a} \operatorname{arctg} \frac{x}{a}, & n = 1. \end{cases}$$

21. Для заданных границ интегрирования a и b вычислите значение определенного интеграла следующего вида:

$$\int \operatorname{tg}^n x dx = \begin{cases} \frac{\operatorname{tg}^{n-1} x}{n-1} - \int \operatorname{tg}^{n-2} x dx, & n \geq 2, \\ -\ln |\cos x|, & n = 1, \\ x, & n = 0. \end{cases}$$

22. Для заданных границ интегрирования a и b вычислите значение определенного интеграла следующего вида:

$$\int \operatorname{ctg}^n x dx = \begin{cases} -\frac{\operatorname{ctg}^{n-1} x}{n-1} - \int \operatorname{ctg}^{n-2} x dx, & n \geq 2, \\ \ln |\sin x|, & n = 1, \\ x, & n = 0. \end{cases}$$

23. Для заданных границ интегрирования a и b вычислите значение определенного интеграла следующего вида:

$$\int x^m \sin ax = \begin{cases} -\frac{x^m}{a} \cos ax + \frac{m}{a} \int x^{m-1} \cos ax dx, & m \geq 1, \\ -\frac{\cos ax}{a}, & m = 0. \end{cases}$$

24. Для заданных границ интегрирования a и b вычислите значение определенного интеграла следующего вида:

$$\int x^m \cos ax = \begin{cases} -\frac{x^m}{a} \sin ax + \frac{m}{a} \int x^{m-1} \sin ax dx, m \geq 1, \\ -\frac{\sin ax}{a}, m = 0. \end{cases}$$

25. Напишите рекурсивную функцию, перемножающую два целых числа, одно из которых неотрицательно, без использования операции умножения.

Контрольные вопросы

1. Какой процесс называется итеративным?
2. Каким образом реализуются итеративные процессы?
3. Какой алгоритм называется рекурсивным?
4. Какая функция называется прямо рекурсивной? Косвенно рекурсивной?
5. Что такое стек? В какой последовательности происходит заполнение стека и выбор элементов из стека?
6. Что должно обязательно присутствовать в теле рекурсивно описанной функции?
7. Перечислите различия между итерацией и рекурсией.
8. Что произойдет, если рекурсивный алгоритм будет вызывать сам себя «бесконечное» число раз?
9. Как предотвратить бесконечное выполнение рекурсивного алгоритма?
10. Верно ли, что решение задачи, реализуемое рекурсивным алгоритмом, можно выразить, используя итерацию?

Пример выполнения лабораторной работы

Задание. Для заданных границ интегрирования a и b вычислите значение определенного интеграла следующего вида:

$$\int x^n e^{cx} dx = \begin{cases} \frac{x^n e^{cx}}{c} - \frac{n}{c} \int x^{n-1} e^{cx} dx, n > 1, \\ \frac{e^{cx}}{c^2} (cx - 1), n = 1; \end{cases}$$

Решение

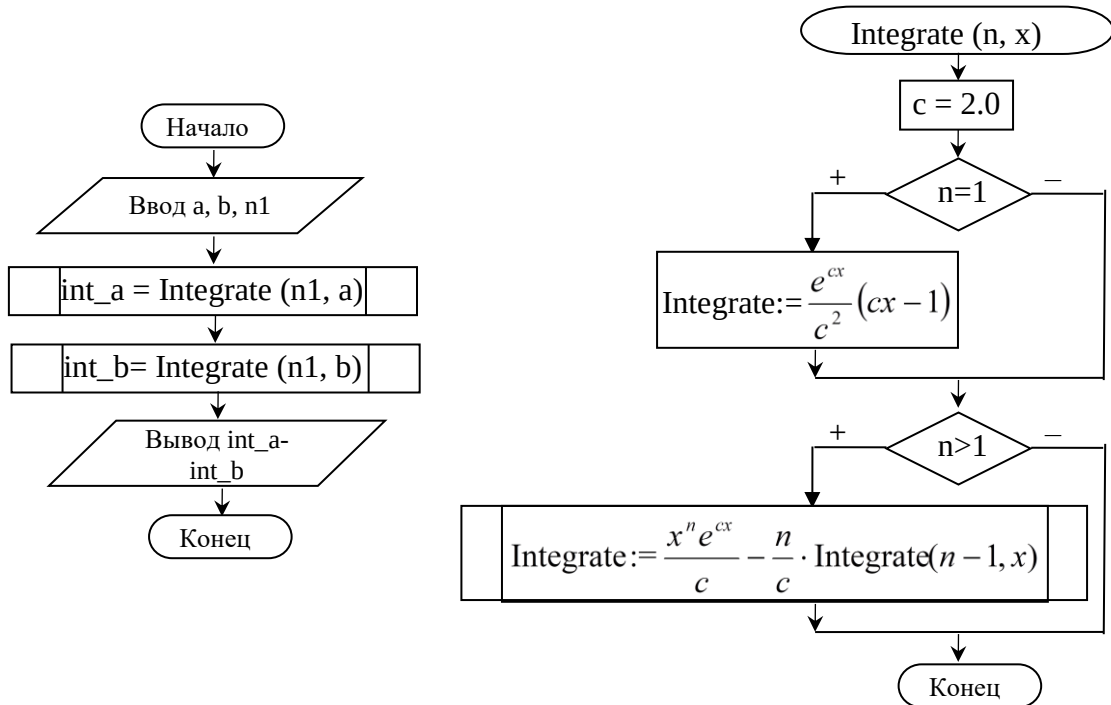
1. Математическая модель

Аргументы: границы интегрирования a , b целого типа;
степень n целого типа.

Результаты: разность $\text{int}_b - \text{int}_a$ вещественного типа.

Промежуточные величины: значение интеграла для нижней границы интегрирования `int_a`; значение интеграла для верхней границы интегрирования `int_b` (вещественного типа).

2. Алгоритм



3. Программа

```

// вычисление значения определенного интеграла
#include <iostream>
#include <conio.h>
#include <math.h>

using namespace std;

double Integrate(int n, double x);

int main()
{
    double b, a;
    int n1;
    cout<<"Введите a и b, где b больше a : ";
    cin>>a>>b;
    cout<<endl;
    cout<<"Введите степень n : ";
    cin>>n1;
    cout<<endl;

    double int_a, int_b;
    int_a = Integrate(n1, a);
    int_b = Integrate(n1, b);
}

```

```

        cout<<"Значение определённого интеграла при a = "<<a<<" и b = "<<b<<"
        равно "<<int_b-int_a;
        _getch();

        return 0;
}

double Integrate(int n, double x)
{
    double c = 2.0; // значение константы a выбирается произвольно
    if(n == 1) return (exp(c*x)/(c*c)*(c*x-1));
    if(n > 1) return (pow(x, n)*exp(c*x)/c-(n/c)*Integrate(n-1, x, c));
}

```

4. Результат работы программы

Введите a и b, где b больше a: 2 4

Введите степень n: 3

Значение определенного интеграла при a = 2 и b = 4 равно 67328,1534

СПИСОК ЛИТЕРАТУРЫ

1. Конспект лекций
2. [2] стр. 82-83.

Лабораторная работа №7

Использование числовых одномерных массивов

ЦЕЛЬ РАБОТЫ: освоение способов описания массива, приобретение навыков организации ввода-вывода и обработки массива.

Формируемые компетенции или их части: вырабатывается навык разработки математических, информационных и имитационных моделей, а также формируется способность к разработке и применению алгоритмических и программных решений в области прикладного программирования (формируется часть компетенций ОПК-3 и ПК-7).

Теоретическая часть

Одномерные массивы

7.1. Понятие структурированного типа данных

Синонимами названия «базовый тип» являются названия «стандартные», «предопределенные». Реализация этих типов заложена в стандарте языка, и программист не вправе изменить предопределенное. Производным типом (определяемым программистом) является такой тип, который должен быть описан пользователем перед употреблением. К производным типам можно отнести массивы, перечисления, функции, структуры, ссылки, объединения и классы. Полезно помнить, что тип данного определяет его размещение в памяти и набор операций.

Структурированный тип представляет собой набор элементов - данных, объединенных программистом в единый блок в соответствии с особенностями решаемой задачи. Данные, входящие в блок, могут быть однотипными или разнотипными, весь блок данных имеет общее имя. Каждый элемент отличается от других либо порядковым номером (индексом) - в массиве, либо своим именем и, возможно, типом - в структуре.

Любой из структурированных типов данных характеризуется множественностью образующих этот тип элементов. Переменная или константа структурированного типа всегда имеет несколько компонент. Каждая из этих компонент, в свою очередь, может принадлежать структурированному типу, что позволяет говорить о возможной вложенности типов.

В C++ можно выделить следующие структурированные типы: массивы, структуры, объединения, классы, файлы.

7.2. Понятие и описание типа массив

При использовании простых переменных каждой области памяти для хранения данных соответствует своё имя. Если с группой величин одинакового типа требуется выполнять однообразные действия, им дают одно имя, а различают по порядковому номеру. Это позволяет компактно записывать множество операций с помощью циклов. Конечная именованная последовательность однотипных величин называется **массивом**. Описание массива в программе отличается от описания простой переменной наличием после имени квадратных скобок, в которых задается количество элементов массива (размер).

Общий вид описания массива:

тип_данных имя_массива [количество_элементов 1]...[количество_элементов n];

Здесь:

1) тип_данных – это тип элементов массива. Это может быть любой стандартный или ранее определённый пользователем тип данных. Например, можно определить массив координат точек на плоскости, тогда один элемент массива содержит две координаты одной точки. Можно определить массив сведений о студенте, тогда один элемент массива содержит разнообразные сведения о студенте, например, фамилия, имя, возраст, адрес и прочие. В этих случаях для определения типа одного элемента массива используется также производный тип, например, массив или структура;

2) имя_массива – правильный идентификатор;

3) количество_элементов – это константа, заданная явно или именованная (константное выражение). **Использовать переменные нельзя!**;

4) n – размерность массива.

При n = 1 описание соответствует одномерному массиву (вектору),

n = 2 – двумерному массиву (матрице),

n > 2 – многомерный массив.

На размерность массивов ограничений не налагается.

Размерность массива всегда равна количеству индексов (измерений).

Под **размером** обычно понимают количество элементов массива.

Пример фиксированного массива на C/C++

```
float a[10]; // описание одномерного массива из 10 вещественных чисел
int Array[10]; // одномерный массив целых чисел с размером 10
// нумерация элементов от 0 до 9
double Array[12][15]; // Двумерный массив вещественных чисел двойной
// точности размера 12 на 15. Нумерация
// по строкам от 0 до 11, по столбцам от 0 до 14
```

Замечание. При описании массивов квадратные скобки являются элементом синтаксиса, а не указанием на необязательность конструкции.

Элементы массива нумеруются с нуля. Инициализирующие значения для массивов записываются в фигурных скобках. Значения элементам присваиваются по порядку. Если элементов в массиве больше, элементы, для которых значения не указаны, обнуляются:

```
int b[5] = {3, 2, 1}; // b[0] = 3, b[1] = 2, b[2] = 1, b[3] = 0, b[4] = 0
```

Замечание. К каждому элементу массива имеется прямой доступ. Для этого необходимо указать имя массива и номер элемента (индекс) в квадратных скобках.

Размер массива вместе с типом его элементов определяет объем памяти, необходимый для размещения массива, которое выполняется на этапе компиляции, поэтому размер может быть задан только целой положительной константой или константным выражением. Если при описании массива не указан размер, должен присутствовать инициализатор, в этом случае компилятор выделит память по количеству инициализирующих значений.

7.3. Одномерные массивы

Одномерный массив – массив, с одним параметром (измерением), характеризующим количество элементов одномерного массива. **Размерность $n = 1$.**

На рисунке 7.1 показана структура целочисленного одномерного массива *a*. Размер этого массива – 10 элементов (ячеек).

-5	2	0	-6	12	56	8	23	4	-3
<i>a</i> [0]	<i>a</i> [1]	<i>a</i> [2]	<i>a</i> [3]	<i>a</i> [4]	<i>a</i> [5]	<i>a</i> [6]	<i>a</i> [7]	<i>a</i> [8]	<i>a</i> [9]

Рисунок 7.1 – Одномерный массив в C++

Максимальный индекс одномерного массива *a* равен 9, но размер массива 10 ячеек, так как нумерация ячеек массива всегда начинается с 0. Индекс ячейки – это целое неотрицательное число, по которому можно обращаться к каждой ячейке массива и выполнять какие-либо действия над ней (ячейкой).

```
int a[10]; // пример объявления массива, изображенного на рисунке 7.1,
           // где int – целочисленный тип данных;
           // a – имя одномерного массива;
           // 10 – размер одномерного массива a.
```

Обычно при описании массива его размер задается в виде именованной константы, например:

```
const int n = 6;
int a[n], b[n]; // описано два одномерных массива из 6 целых чисел
```

Над массивом как единой структурой никакие операции не определены. Все действия выполняются с отдельными элементами массива. Причем, с элементами массива допустимы все операции, определенные для переменных того же типа.

Для обращения к элементу массива после имени массива указывается номер элемента в квадратных скобках, например: *a*[4], *b*[1].

В следующем примере подсчитывается сумма элементов массива.

```
#include <iostream.h>
int main()
{
    const int n = 10;
    int i, sum;
    int marks[n] = {3, 4, 5, 4, 4};
    for ( i = 0, sum = 0; i < n; i++) sum += marks[i];
    cout << "Сумма элементов: " << sum;
    return 0;
}
```

Размер массивов предпочтительнее задавать с помощью *именованных констант*, как это сделано в примере, поскольку при таком подходе для её изменения достаточно скорректировать значение константы всего лишь в одном месте программы.

Замечание. При обращении к элементам массива автоматический контроль выхода индекса за границу массива не производится, что может привести к ошибкам.

7.4. Основные действия над элементами массивов

1. Инициализация массива: присвоение каждому элементу начального значения:

а) инициализация:

```
int a[6] = {0, 5, -7, 100, 15}; //a[0] = 0, a[1] = 5, a[2] = -7,  
                               //a[3] = 100, a[4] = 15, a[5] = 0
```

Инициализация одномерного массива выполняется в фигурных скобках после знака равно, каждый элемент массива отделяется от предыдущего запятой. Количество констант должно соответствовать числу элементов массива. В противном случае, если размер массива больше, чем количество инициаторов, то элементы, для которых значения не указаны, обнуляются.

```
int a[]={5,-12,9,10}; //инициализация массива без определения его размера
```

В данном случае компилятор сам определит размер одномерного массива. Размер массива можно не указывать только при его инициализации, при обычном объявлении массива обязательно нужно указывать размер массива. Чтобы программно определить число элементов в таком массиве, используется операция sizeof:

```
int len; // число элементов  
len = sizeof (month ) / sizeof (int);
```

Итак, при описании массива можно выполнить начальную инициализацию значений его элементов, для этого нужно задать список инициализирующих значений. Например, в году всегда 12 месяцев, значение числа дней в каждом месяце известно, значит, такая структура может быть задана массивом:

```
int month[12] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
```

Эквивалентом инициализации является простое **присваивание** вида:

```
month[0] = 31;          // Январь  
...  
month[11] = 31;         // Декабрь
```

б) ввод элементов массива с клавиатуры:

```
const int n = 10;  
int a[n];  
for ( i = 0; i < n; i++)  
{  
    cout <<"Введите "<<i<<" элемент: ";  
    cin>>a[i];  
}
```

в) формирование массива с помощью генератора псевдослучайных чисел:

```
#include <time.h>      // для time(0)  
#include <stdlib.h>    // для srand() и rand()  
...  
const int n = 10;  
int a[n], b[n];
```

```
srand(time(0)); // Записывается один раз в начале
for ( i = 0; i < n; i++) a[i] = rand()%10; // a[i] ∈ [0, 10)
for ( i = 0; i < n; i++) b[i] = rand()%51-25; // a[i] ∈ [-25, 25]
```

Функция `srand()` применяется для обновления базы генерации при использовании функции `rand()`, генерирующей псевдослучайные числа из интервала $[0, \text{RAND_MAX}]$, где `RAND_MAX` – константа, содержащая наибольшее возможное значение для типа, выбранного в качестве базы генерации. В примере, приведенном выше генерируются элементы типа `int` и значение `RAND_MAX = 32767`

г) вычисление элементов массива по формуле:

```
for ( i = 0; i < n; i++) a[i] = 6 * i - 2;
```

2. Вывод массива на экран:

```
cout << " Массив A: " << endl;
for ( i = 0; i < n; i++) cout << a[i] << "\t";
```

3. Обработка массива

При работе с одномерными массивами существует большое количество алгоритмов, которые считаются типовыми, т.е. часто встречающимися в практических задачах: ввод-вывод массива, вычисление значений элементов массива, поиск суммы и (или) произведения элементов массива, поиск максимального (минимального) элемента массива, упорядочивание элементов массива, добавление (вставка) новых элементов в массив, удаление данных из массива и т.д.

Пример. Дан массив действительных чисел из n элементов. Найти максимальный по модулю элемент и разделить все элементы массива на полученное значение. Вывести на экран монитора массив после обработки.

Возможный текст программы:

```
#include <math.h>
#include <iostream>
using namespace std;
int main()
{
    const int n = 5;
    double x[n], max;
    int i; // далее задаём массив из n действительных чисел
    cout << "Input " << n << " numbers:" << endl;
    for(i = 0; i < n; i++)
        cin >> x[i];

    // Поиск максимального по модулю элемента массива
    // Предположим, что x[0] - это и есть максимальный
    // по модулю элемент массива:
    max = fabs(x[0]);

    // А теперь пробуем себя опровергнуть:
    for(i = 1; i < n; i++) //перебираем все элементы массива с первого
        if(fabs(x[i]) > max) // сравниваем с текущим максимальным
            max = fabs(x[i]); // выполняем переприсваивание
```

```

cout << "max=" << max << endl;    // Максимум найден

for(i = 0; i < n; i++)
    x[i] /= max;                    // Делим все элементы на max

cout << "Massiv:" << endl;        // Распечатка массива
for(i = 0; i < n; i++)
    cout << x[i] << endl;

return 0;
}

```

Практическая часть

ВЫПОЛНЕНИЕ РАБОТЫ: освоить теоретический материал, выполнить общее для всех задание I и в соответствии с вариантом составить программу (задание II), при необходимости реализовав пользовательскую функцию.

Задание I

Изучить порядок описания, ввода-вывода и обработки массивов:

1. Набрать и отладить программу нахождения суммы элементов массива, стоящих в нечетных позициях. Оформить в тетради, записав условие, код, блок-схему и результат работы, объяснить назначение операторов 0-4.

```

#include <time.h>
#include <stdlib.h>
int main()
{
    setlocale(LC_ALL,"Rus");
    srand(time(0));                // Оператор 0
    const int N=10;
    int Arr[N];
    int sum=0;                      // Оператор 1
    for (int i=0;i<N;i++)
    {
        Arr[i]= rand()%100;        // Оператор 2
        cout<<Arr[i]<<"\t";
    }
    for (int i=1;i<N;i+=2)          // Оператор 3
        sum+=Arr[i];               // Оператор 4
    cout<<"\nСумма = "<<sum<<endl;
    return 0;
}

```

2. Набрать и отладить программу нахождения суммы элементов массива, кратных 3. Оформить в тетради, записав условие, код, блок-схему и результат работы, объяснить назначение операторов 0-3.

```

#include <time.h>
#include <stdlib.h>
int main()
{
    time_t t;
    srand((unsigned)time(&t));      // Оператор 0
}

```

```

const int N=10;
int Arr[N];
int sum=0; // Оператор 1
for (int i=0;i<N;i++)
{
    Arr[i]=rand()%50; // Оператор 2
    cout<<Arr[i]<<"\t";
}
for (int i=1;i<N;i++)
    if (Arr[i]%3==0) sum+=Arr[i]; // Оператор 3
cout<<"\nSumma="<<sum<<endl;
return 0;
}

```

3. Набрать и отладить программу нахождения максимального элемента массива из N целых чисел с выводом номеров наибольших элементов. Оформить в тетради, записав условие, код, блок-схему и результат работы, объяснить назначение операторов 1-6.

```

#include <time.h>
#include <stdlib.h>
int main()
{
    srand(time(0));
    const int N=10;
    int Arr[N];
    int index[N]={0}; // Оператор 1
    int max, k=0;
    for (int i=0;i<N;i++){
        Arr[i]=rand()%100-50; // Оператор 2
        cout<<Arr[i]<<"\t";
    }
    max=Arr[0]; // Оператор 3
    for (int i=1;i<N;i++)
        if (Arr[i]>=max) // Оператор 4
        {
            max=Arr[i];
            index[k]=i+1; // Оператор 5
            k++;
        }
    k=0;
    cout<<"\nMax="<<max<<"\nИндексы: ";
    while(index[k]>0) // Оператор 6
    {
        cout<<index[k]-1<<"\t";
        k++;
    }
    return 0;
}

```

Задание II

В соответствии с вариантом составить и реализовать программу:

1. Даны два массива разных размеров. Определить, какие элементы первого массива и сколько раз встречаются во втором массиве.
2. Массив содержит $2n$ чисел. Из суммы первых n его элементов вычесть сумму последних n элементов.
3. Транспонировать массив, т.е. по $a_1, a_2, \dots, a_n$ сформировать $a_n, a_{n-1}, \dots, a_1$. В полученном массиве найти индекс минимального элемента.
4. Из заданного целочисленного массива удалить все повторяющиеся элементы, оставив только их первые вхождения, т.е. из заданного массива получить новый массив, состоящий из различных целых чисел.
5. Заменить отрицательные числа в массиве их квадратами, оставив остальные без изменения.
6. В заданном массиве найти среднее арифметическое положительных чисел, среднее арифметическое отрицательных чисел и число нулей.
7. В массиве из $2n$ чисел найти сумму квадратов элементов с четными индексами и сумму кубов элементов с нечетными индексами.
8. Из чисел $a_1, a_2, \dots, a_n$ выбрать те, которые больше по модулю заданного числа c , и образовать из них новый массив, сохранив порядок следования элементов.
9. Из массива целых чисел составить три других, в первый из которых записать числа, кратные 5, во второй - числа, кратные 7, а в третий - остальные числа.
10. Задан массив из 100 целых случайных чисел, принадлежащих промежутку $[0, 100]$. Найти сумму тех элементов массива, которые больше 15, но меньше 45, а также вычислить количество этих элементов.
11. В линейном массиве заменить все элементы на число m (m – индекс максимального элемента).
12. Дан массив, состоящий как из положительных, так и отрицательных чисел. Нужно сначала записать положительные числа, а затем отрицательные в том же порядке, как они были расположены в исходном массиве. Если есть нули, записать их в последнюю очередь.
13. Найти сумму элементов данного массива. Разделить каждый элемент исходного массива на полученное значение.
14. Вычислить сумму и разность массивов одного размера.
15. Найти среднее арифметическое значение элементов заданного массива. Преобразовать исходный массив, вычитая из каждого элемента среднее значение.
16. Даны два массива одинакового размера. Рассматривая их как арифметические векторы, найти длины этих векторов и их скалярное произведение.

17. Заданы два массива разных размеров. Объединить их в один массив, включив второй массив между k -ым и $(k + 1)$ -ым элементами первого (k вводится с клавиатуры).
18. Вычесть из положительных элементов данного массива элемент с номером k_1 а к отрицательным элементам прибавить элемент с номером k_2 . Нулевые элементы заменить 1. Номера k_1 и k_2 вводятся с клавиатуры.
19. К четным элементам целочисленного массива прибавить данное число a , а из элементов с четными номерами вычесть данное число b .
20. Дан первый член геометрической прогрессии и ее знаменатель. Сформировать одномерный массив, элементами которого служат первые n членов этой прогрессии.
21. Сформировать массив из первых 30 членов последовательности Фибоначчи.
22. Вставить одно и то же число, введенное с клавиатуры, перед каждым отрицательным элементом заданного целочисленного массива.
23. Дан массив четного размера. Поменять местами его половины следующим образом: первый элемент - с последним, второй - с предпоследним элементом и т.д.
24. Даны два целочисленных массива одинакового размера. Получить третий массив того же размера, каждый элемент которого равен большему из соответствующих элементов данных массивов.
25. Задан массив из n целых случайных чисел, принадлежащих промежутку $[-25, 25]$. Найти произведение тех элементов массива, которые больше 1, но меньше 15, а также вычислить количество четных элементов массива.

Контрольные вопросы

1. Дайте определение производного типа данных, структурированного типа.
2. Дайте определение массива.
3. Каким может быть тип элементов массива?
4. Имя, размер и размерность массива.
5. Какова структура одномерного массива?
6. Правила описания одномерного массива.
7. Как осуществляется доступ к элементам одномерного массива?
8. Как осуществляется ввод массива?
9. Какие способы ввода массива вы знаете?
10. Вывод линейного массива.

Пример выполнения задания II лабораторной работы

Задание. Массив D содержит 24 значения атмосферного давления за каждый час в течение суток. Определить, какое значение атмосферного давления было наибольшим и в какое время оно было зафиксировано.

Решение

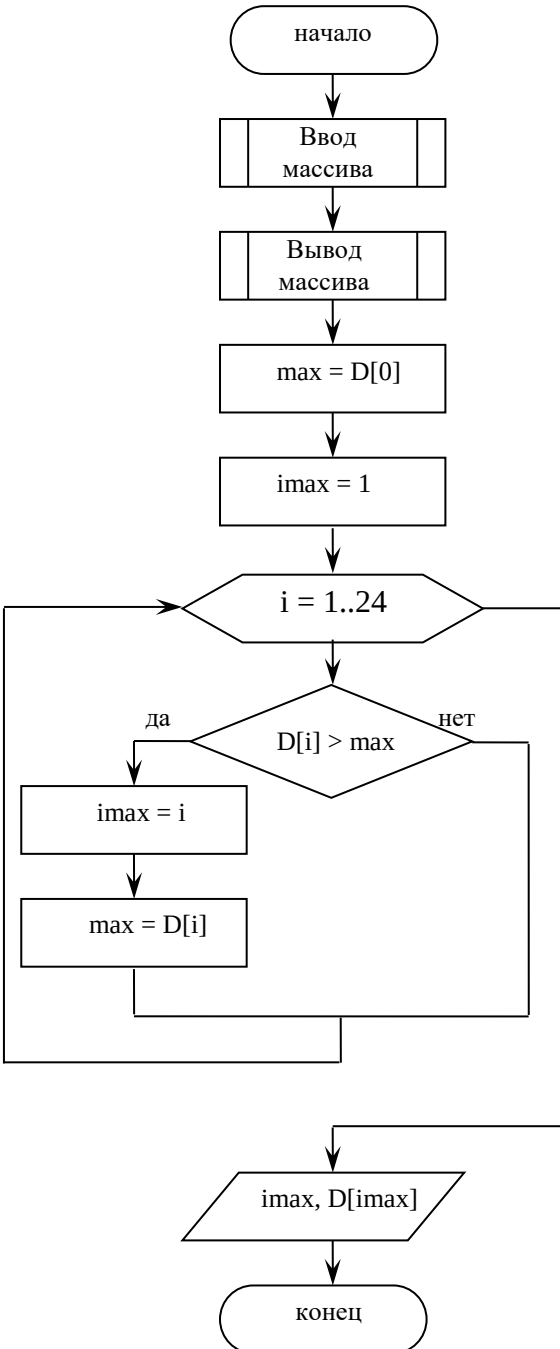
1. Математическая модель

Значением атмосферного давления являются элементы массива $D[24]$, значением времени – индексы элементов. Решение задачи сводится к поиску максимального элемента в массиве и определению его индекса.

Аргументы: $D[24]$ – массив целого типа.

Результаты: $imax$ целого типа – индекс максимального элемента;
 $D[imax]$ целого типа – значение максимального элемента.

Промежуточная величина: max целого типа – максимальное значение элемента массива.

2. Алгоритм	3. Программа
 <pre>graph TD Start([начало]) --> Input[/Ввод массива/] Input --> Output[/Вывод массива/] Output --> Init1[max = D[0]] Init1 --> Init2[imax = 1] Init2 --> LoopStart{i = 1..24} LoopStart --> Decision{D[i] > max} Decision -- да --> Update1[imax = i] Update1 --> Update2[max = D[i]] Update2 --> LoopStart Decision -- нет --> LoopStart LoopStart --> OutputBox[/imax, D[imax]/] OutputBox --> End([конец])</pre>	<pre>#include <iostream> #include <conio.h> #include <math.h> using namespace std; void Input(int mass[24]); void Print(int mass[24]); int main() { int D[24]; cout<<" Введите массив "<<endl; Input(D); cout<<" Массив значений давле- ния:"<<endl; Print(D); int max = D[0], imax = 1; for(int i=0; i<24; i++) { if (D[i]>max) { imax = i+1; max = D[i]; } } cout<<" Максимальное значение давления : "<<D[imax-1]<<" в "<<imax<<" ч. "; _getch(); return 0; } void Input(int mass[24]) { for(int i=0; i<24; i++) { cin>>mass[i]; } } void Print(int mass[24]) { for(int i=0; i<24; i++) { cout<<mass[i]<<" "; } cout<<endl; }</pre>

4. Результат работы программы:

Массив значений давления:

748	743	756	748	741	760	748	743	757	748	756	756	751	760	754
756	752	758	758	757	750	747	759	750						

Максимальное значение давления 760 в 6 ч.

СПИСОК ЛИТЕРАТУРЫ

1. Конспект лекций.
2. [2] стр. 58-61.

Лабораторная работа №8

Двумерные массивы

ЦЕЛЬ РАБОТЫ: приобретение навыков объявления, организации ввода-вывода и обработки двумерного массива.

Формируемые компетенции или их части: вырабатывается навык разработки математических и информационных моделей, а также формируется способность к разработке и применению алгоритмических и программных решений в области прикладного программирования (формируется часть компетенций ОПК-3 и ПК-7).

Теоретическая часть

Двумерные массивы

Итак, **массив** представляет собой структуру данных, позволяющую хранить под одним именем совокупность данных любого, но только одного какого-то типа. Массив характеризуется своим именем, типом хранимых элементов, размером (числом хранимых элементов), нумерацией элементов и размерностью.

Объявление переменной как одномерного массива (массива с размерностью 1) имеет вид:

тип_данных идентификатор [константное_выражение];

Например, оператор

`int A[10];` // объявляет массив с именем A, содержащий 10 целых чисел

Доступ к элементам этого массива осуществляется выражением $A[i]$, где i - индекс, являющийся в данном примере целым числом в диапазоне 0 - 9. Например, $A[0]$ - значение первого элемента, $A[1]$ - второго, $A[9]$ - последнего.

Напомним, что объявление переменной массива можно совмещать с заданием элементам массива начальных значений. Например, оператор:

`int A[10] = {0};` //присваивает нулевые значения всем элементам массива
`int A[10] = {1,2,3,4,5,6,7,8,9,10};` // $A[0] = 1$, $A[2] = 2$, ..., $A[9] = 10$

Если начальных значений меньше, чем элементов в массиве, оставшиеся элементы автоматически получают нулевые начальные значения.

Можно объявлять многомерные массивы, т.е. массивы, элементами которых являются массивы. Например, двумерный массив можно объявить таким образом:

`int A2[10][3];`

Этот оператор описывает двумерный массив, который можно представить как таблицу, состоящую из 10 строк и 3 столбцов.

Фактически двумерный массив – это одномерный массив одномерных массивов.

Визуально, двумерный массив – это обычная таблица, со строками и столбцами или матрица $A = \{a_{ij}\}_{m \times n}$.

Структура двумерного массива, с именем A, размером m на n показана ниже (см. рис. 8.1).

A[0][0]	A[0][1]	...	A[0][n-1]
A[1][0]	A[1][1]	...	A[1][n-1]
...	...	...	...
A[m-1][0]	A[m-1][1]	...	A[m-1][n-1]

$$A = \begin{pmatrix} a_{00} & a_{01} & \dots & a_{0n-1} \\ \dots & \dots & \dots & \dots \\ a_{m-1\ 0} & a_{m-1\ 1} & \dots & a_{m-1n-1} \end{pmatrix}$$

Рисунок 8.1 – Двумерный массив в виде таблицы (слева) или матрицы (справа)

На рисунке 8.1 m – количество строк двумерного массива;

n – количество столбцов двумерного массива;

$m * n$ – количество элементов массива.

Таким образом, **двумерный массив** – это массив, размерность которого равна 2.

Общий вид описания двумерного массива:

тип_данных имя_массива [количество_строк][количество_столбцов];

В объявлении двумерного массива, также как и в объявлении одномерного массива, первым делом, нужно указать:

- тип данных;
- имя массива.

Доступ к значениям элементов многомерного массива обеспечивается через индексы, каждый из которых заключается в квадратные скобки. Например, $A2[3][2]$ – значение элемента, лежащего на пересечении четвертой строки и третьего столбца (помните, что индексы начинаются с 0).

Если многомерный массив инициализируется при его объявлении, список значений по каждой размерности заключается в фигурные скобки. Приведенный ниже оператор объявляет трехмерный массив $A3$ размерностью 4 на 3 на 2.

```
int A3[4][3][2] = {{{0,1},{2,3},{4,5}},
                  {{6,7},{8,9},{10,11}},
                  {{12,13},{14,15},{16,17}},
                  {{18,19},{20,21},{22,23}}};
```

Этот оператор создает массив $A3$, четыре строки которого являются матрицами вида:

0	1	6	7	12	13	18	19
2	3	8	9	14	15	20	21
4	5	10	11	16	17	22	23

Например, элемент $A3[0][1][0]$ равен 2, элемент $A3[3][0][1]$ равен 19 и т.д.

Если в списке инициализации в какой-то из размерностей не хватает данных, то все дальнейшие не перечисленные элементы считаются равными нулям.

Если массив при его объявлении не инициализирован, то его элементы имеют случайные значения. Элементы такого массива нельзя использовать в выражениях, пока им не будут присвоены какие-нибудь значения.

Все размерности массива должны быть константами или константными выражениями, поскольку инструкции по выделению памяти формируются компилятором до выполнения программы.

В памяти многомерный массив располагается по строкам.

Строки массива ничем не отделены одна от другой, то есть двумерный массив является прямоугольной матрицей только в нашем воображении. При просмотре массива от начала в первую очередь изменяется правый индекс (номер столбца).

К элементу двумерного массива обращаются, указывая номер строки и номер столбца, на пересечении которых он расположен, например:

`a[1][4]` `a[i][j]`

Компилятор воспринимает как номер строки первый индекс, как бы он ни был обозначен в программе.

1. **Инициализация** двумерных массивов аналогична инициализации одномерных массивов:

а) инициализация:

```
const int n = 3; // число строк
const int m = 4; // число столбцов
// выделяем память под матрицу с одновременной
// инициализацией элементов
double a[n][m] =
{
    {3, -3.1, 4.6, 5.5},
    {2, 1.2, -10, -2.5},
    {7.6, 3.1, 1.6, 0.5}
};
```

При инициализации двумерного массива в ходе работы программы, необходимо организовать вложенные циклы:

б) ввод данных с клавиатуры:

```
for(i = 0; i < n; i++)
    for(j = 0; j < m; j++)
        cin >> a[i][j];
```

В соответствии с приведенным здесь порядком следования циклов элементы массива должны вводиться по строкам.

в) формирование массива с помощью генератора псевдослучайных чисел;

г) непосредственное присваивание (вычисление по формуле):

2. **Вывод** матрицы надо реализовать в удобном для чтения виде, т.е. чтобы на одной строке экрана располагалась одна строка матрицы. С этой целью в тело внешнего цикла, помимо внутреннего, включается еще оператор `cout << endl;`, который переводит курсор к началу следующей строки экрана после вывода текущей строки матрицы.

```
for(i = 0; i < n; i++)
{
    for(j = 0; j < m; j++)
        cout << a[i][j] << "\t ";
    cout << endl;
}
```

Пример. Дана прямоугольная матрица действительных чисел размером $n \times m$. Пронормировать эту матрицу, т.е. поделить значение всех элементов матрицы на максимальный по модулю элемент.

Возможный текст программы:

```
#include <iostream>
#include <cmath>

using namespace std;

int main()
{
    const int n = 3; // число строк
    const int m = 4; // число столбцов
    double a[n][m]; // выделяем память под матрицу
    int i, j;
    double max;

    // Ввод матрицы с клавиатуры
    cout << "Matriza A("<< n << "*" << m << "):" << endl;
    for(i = 0; i < n; i++)
        for(j = 0; j < m; j++)
            cin >> a[i][j];

    // Поиск в матрице максимального по модулю значения
    max = fabs(a[0][0]);
    for(i = 0; i < n; i++)
        for(j = 0; j < m; j++)
            if(fabs(a[i][j]) > max)
                max = fabs(a[i][j]);
    cout << "max=" << max << endl;

    // Нормируем матрицу
    for(i = 0; i < n; i++)
        for(j = 0; j < m; j++)
            a[i][j] /= max;

    // Вывод матрицы в виде таблицы на экран монитора
    for(i = 0; i < n; i++)
    {
        for(j = 0; j < m; j++)
            cout << a[i][j] << "\t ";
        cout << endl;
    }
    return 0;
}
```

Как видно из текста программы, для работы с матрицей почти всегда требуются двойные циклы. Особое внимание следует обратить на вывод двумерного массива в виде таблицы.

Практическая часть

ВЫПОЛНЕНИЕ РАБОТЫ: в соответствии с вариантом составить и реализовать программу (задание I- II).

Задание I

1. Составить программу формирования матрицы, число строк и столбцов которой определяется как случайное натуральное число в промежутке $[3, 8]$, а элементы - случайные целые числа от -50 до 50 . Заменить неотрицательные числа полученной матрицы единицей, а отрицательные - нулем.
2. В целочисленной прямоугольной матрице числа, кратные заданному числу m , вводимому с клавиатуры, заменить частным от деления на m , а остальные числа заменить их остатками от деления на m .
3. В квадратной матрице порядка n найти сумму всех тех элементов, сумма индексов которых равна n .
4. Определить, является ли заданная квадратная матрица симметричной относительно главной диагонали. Найти след данной матрицы. Элементы матрицы вводить с клавиатуры.
5. Найти число элементов в каждой из нечетных строк матрицы, равных заданному числу, вводимому с клавиатуры.
6. Преобразовать данную прямоугольную матрицу так, чтобы все элементы в ее строках были записаны в обратном порядке.
7. Вычислить количество и сумму положительных элементов в нижнем левом треугольнике данной квадратной матрицы, включая диагональные элементы.
8. Даны вещественные числа $a_1, a_2, \dots, a_n$ и вещественная квадратная матрица порядка n ($n > 5$). Сформировать вещественную матрицу размером $n \times (n + 1)$, вставив в исходную матрицу между пятым и шестым столбцами новый столбец с элементами $a_1, a_2, \dots, a_n$.
9. Дана вещественная квадратная матрица порядка n . Преобразовать эту матрицу по правилу: строку с номером k сделать столбцом с номером k , а столбец с номером k сделать строкой с тем же номером.
10. В данной вещественной матрице размером $n \times m$ ($n > 2, m > 2$) поменять местами строки с номерами $n - 2$ и $n - 1$.
11. Положительные элементы первой строки прямоугольной матрицы умножить на первый элемент этой же строки, а отрицательные - на последний ее элемент; то же самое проделать с остальными строками.
12. Заменить все элементы строки с номером s и столбца с номером k прямоугольной матрицы на противоположные по знаку (элемент, стоящий на пересечении выбранной строки и выбранного столбца, не изменять).

13. Из квадратной матрицы получить другую матрицу путем исключения диагональных элементов и вычитания из каждого элемента следа исходной матрицы.
14. Дан целочисленный одномерный массив: $a_1, a_2, \dots, a_n$. Сформировать квадратную матрицу порядка n , у которой элементы, расположенные на главной диагонали, равны соответственно $a_1, a_2, \dots, a_n$; элементы, расположенные выше главной диагонали, равны единице, а элементы, расположенные ниже главной диагонали, равны минус единице.
15. Даны две прямоугольные целочисленные матрицы A и B одинакового размера $m \times n$. Создать матрицу того же размера, в которой элементы равны 1, если соответствующие элементы матриц A и B имеют одинаковый знак; -1, если соответствующие элементы имеют разные знаки, и нулю, если хотя бы один из соответствующих элементов равен нулю.
16. Составить программу вывода на экран арифметического квадрата: в нем первый столбец и первая строка заполнены единицами, а каждый из остальных элементов равен сумме своих соседей сверху и слева.
17. Из прямоугольной матрицы размером $m \times n$ удалить строку с номером s и столбец с номером k ($0 \leq s < m, 0 \leq k < n$).
18. Сформировать одномерный массив, каждый элемент которого равен сумме отрицательных элементов соответствующей строки данной целочисленной прямоугольной матрицы.
19. Найти количество элементов в каждом столбце прямоугольной вещественной матрицы, меньших среднего арифметического элементов рассматриваемого столбца.
20. Вывести на экран максимальные элементы каждой строки двумерного массива с указанием их индексов.
21. Задать два двумерных массива, содержащих положительные и отрицательные элементы. Определить, в каком из них больше положительных элементов.
22. Задан массив целых случайных чисел, принадлежащих промежутку $[0, 200]$. Определить, каких элементов в нём больше: превышающих k_1 или не превышающих k_2 .
23. Задан массив целых случайных чисел, принадлежащих промежутку $[-50, 50]$. Найти сумму тех элементов массива, которые больше a , но меньше b , а также вычислить количество этих элементов.
24. Поменять местами максимальный и минимальный элемент заданного двумерного массива. Результат вывести на экран в виде матрицы.
25. Дана целочисленная матрица. Сформировать матрицу, получающуюся из данной перестановкой первого столбца с последним, второго - с предпоследним.

26. Из данной матрицы получить другую следующим образом: на место первой строки поместить вторую, на место второй - третью и т.д., на место последней поместить первую.

Задание II

Квадратная матрица A третьего порядка сформирована из вещественных чисел, принадлежащих диапазону $[-10, 10]$. Найти с точностью до $\varepsilon = 10^{-3}$ [6]:

- 1) матрицу B по заданной формуле вычисления ее элементов b_{ij} через индексы i и j ($i = 1, 2, 3; j = 1, 2, 3$);
- 2) сумму матриц A и B ;
- 3) матрицу, транспонированную матрице $A + B$;
- 4) матрицу, обратную матрице $A + B$;
- 5) произведение матриц $A + B$ и $(A + B)^T$.

Вывести, используя вспомогательную функцию, на экран матрицы A , B , $A + B$, $(A + B)^{-1}$, $(A + B)^T$, $(A + B)(A + B)^{-1}$.

$$1. \quad b_{ij} = \frac{\sqrt{i+j+2}}{2i+1};$$

$$2. \quad b_{ij} = \frac{i+2j}{3+\ln(i+j+5)};$$

$$3. \quad b_{ij} = \frac{25 \ln(i+0.5)}{i+j+1};$$

$$4. \quad b_{ij} = \frac{5(i+1)+2j^2}{3j+1};$$

$$5. \quad b_{ij} = \frac{\cos(i+2j)}{1+\sin i};$$

$$6. \quad b_{ij} = \frac{i+\ln(4+j)}{2i+3};$$

$$7. \quad b_{ij} = 0.5i + \frac{10}{j+1};$$

$$8. \quad b_{ij} = \frac{\cos i+j}{11i+5};$$

$$9. \quad b_{ij} = \frac{10+\tan(i+2j)}{\sqrt{j+4}};$$

$$10. \quad b_{ij} = \frac{e^{2i+j}}{i+2+3j};$$

$$11. \quad b_{ij} = \frac{\sin(i+2j)+10}{\sqrt{1+2j}};$$

$$12. \quad b_{ij} = \frac{i+6+7j}{\ln(i+5)+10};$$

$$13. \quad b_{ij} = \frac{\ln(i^2+4)}{\cos i+2};$$

$$14. \quad b_{ij} = \frac{\cos(i+j/2)}{\ln(7j+7)+i};$$

$$15. \quad b_{ij} = \frac{\tan i+5}{\ln(5+j)+20}$$

$$16. \quad b_{ij} = \frac{11i+2j+7}{e^{-j}};$$

$$17. \quad b_{ij} = \frac{e^{\frac{i+j}{2}}}{i+j+1};$$

$$18. \quad b_{ij} = \frac{\ln(i+0.5)+2j}{\sin(i+j)+3};$$

$$19. \quad b_{ij} = \frac{3+\sqrt{2j}}{i+1} + 4;$$

$$20. \quad b_{ij} = \frac{i+4j+10}{\sin i + \cos j + 5};$$

$$21. \quad b_{ij} = \frac{\sqrt{i^2+4j+1}}{5+7j};$$

$$22. \quad b_{ij} = \frac{\sin i}{9+\ln(i+j+10)};$$

$$23. \quad b_{ij} = \frac{\tan(2i+3)+i^2}{\sqrt{i+1}};$$

$$24. \quad b_{ij} = \frac{i+2j^4}{5+\sqrt{i^3+j}};$$

$$25. \quad b_{ij} = \frac{5(i^2+2j)}{\sin j+2};$$

$$26. \quad b_{ij} = \frac{2i^2+\ln(j+3)}{2(i+j)+1}.$$

Замечание. Обратную матрицу рекомендуется вычислять с помощью присоединенной матрицы, а определитель матрицы – по правилу Саррюса.

Контрольные вопросы

1. Дайте определение массива.
2. Какие типы данных не допустимы для компонентов массива?
3. Может ли левая граница индексов массива быть отрицательной?
4. Какой массив называется двумерным?
5. Правила описания двумерного массива.
6. Как осуществляется доступ к элементам двумерного массива?
7. Как осуществляется ввод двумерного массива?
8. Какие способы ввода двумерного массива вы знаете?
9. Размещение массива в памяти ЭВМ.
10. Вывод двумерного массива.
11. Может ли инструкция `cout<<x`; ввести массив `x` целиком?

Пример выполнения лабораторной работы

Задание. Программа, которая для целочисленной матрицы 3×4 определяет среднее арифметическое ее элементов и количество положительных элементов в каждой строке.

Решение

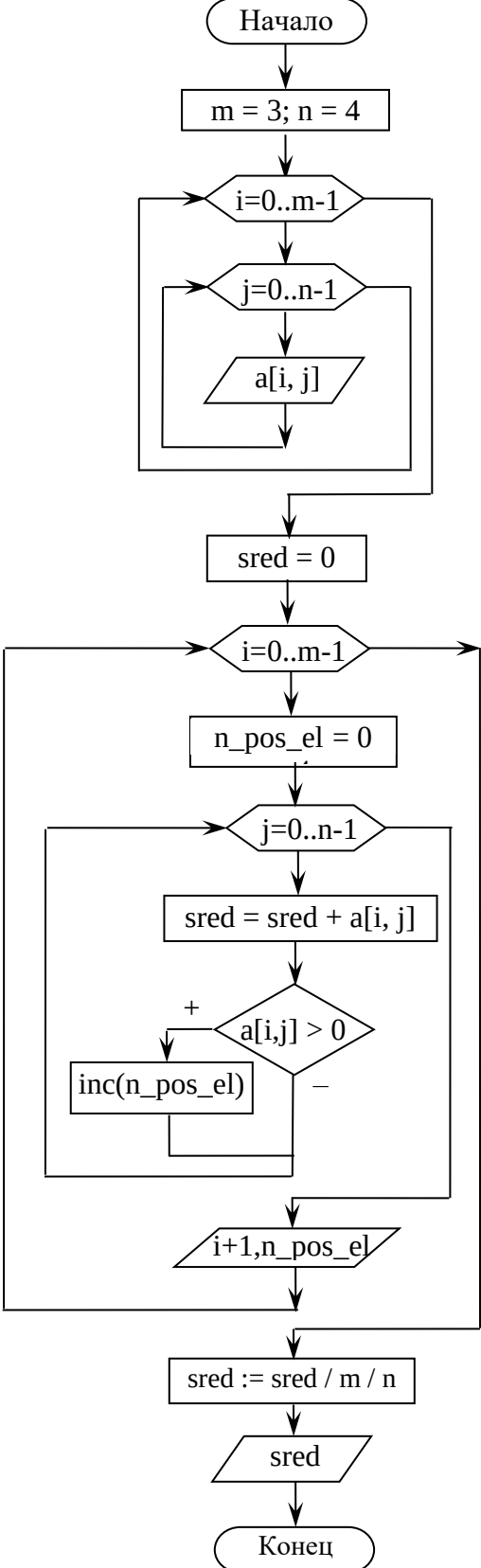
1. Математическая модель

Для нахождения среднего арифметического элементов массива требуется найти их общую сумму, после чего разделить ее на количество элементов. Порядок перебора элементов массива (по строкам или по столбцам) роли не играет. Нахождение количества положительных элементов каждой строки требует просмотра матрицы по строкам.

Аргументы: целочисленный массив `a`; целочисленные `m` и `n` – количество строк и столбцов.

Результаты: среднее арифметическое элементов `sred` (вещественного типа); количество положительных элементов в каждой строке `n_pos_el` (целого типа).

Промежуточные величины: счетчики `i`, `j` (целого типа).

2. Алгоритм	3. Программа
 <pre>graph TD Start([Начало]) --> Init[m = 3; n = 4] Init --> Loop1{i=0..m-1} Loop1 --> Loop2{j=0..n-1} Loop2 --> Read[/a[i, j]/] Read --> Loop2 Loop2 --> Loop1 Loop1 --> Sred0[sred = 0] Sred0 --> Loop3{i=0..m-1} Loop3 --> Npos0[n_pos_el = 0] Npos0 --> Loop4{j=0..n-1} Loop4 --> Sum[sred = sred + a[i, j]] Sum --> Cond{a[i, j] > 0} Cond -- + --> Inc[inc(n_pos_el)] Inc --> Loop4 Cond -- - --> Loop4 Loop4 --> Next[/i+1, n_pos_el/] Next --> Loop3 Loop3 --> Avg[sred := sred / m / n] Avg --> Out[/sred/] Out --> End([Конец])</pre>	<pre>#include <iostream> #include <conio.h> #include <math.h> const int m = 3, n = 4; using namespace std; int main() { int a[m][n]; cout<<"Введите элементы массива "<<endl; for(int i=0; i<m; i++) for(int j=0; j<n; j++) cin>>a[i][j]; double sred = 0; int n_pos_el; for(int i=0; i<m; i++) { n_pos_el = 0; for(int j=0; j<n; j++) { sred += a[i][j]; if(a[i][j]>0) n_pos_el++; } cout<<"В "<<i+1<<"-ой строке " <<n_pos_el<<" положительных элементов" <<endl; } sred = sred/m/n; cout<<"Среднее арифметическое: " <<sred<<endl; _getch(); return 0; }</pre>
4. Результат работы программы:	
Введите элементы массива	
1 2 3 4 -2 0 -3 -1 1 1 1 0	
В 1-й строке 4 положительных элементов	
В 2-й строке 0 положительных элементов	
В 3-й строке 3 положительных элементов	
Среднее арифметическое: 0.58	

СПИСОК ЛИТЕРАТУРЫ

1. Конспект лекций.
2. [2] стр. 61-63.

Лабораторная работа № 9

Задачи сортировки и поиска

ЦЕЛЬ РАБОТЫ: Закрепление теоретических знаний и приобретение практических навыков по составлению алгоритмов и программ различных методов сортировки.

Формируемые компетенции или их части: вырабатывается навык разработки математических и имитационных моделей, а также формируется способность к разработке и применению алгоритмических и программных решений в области прикладного программирования (формируется часть компетенций ОПК-3 и ПК-7).

Теоретическая часть

Алгоритмы решения задач внутренней сортировки и алгоритмы поиска информации

9.1. Сложность алгоритмов

Теория сложности обеспечивает методологию анализа **вычислительной сложности** различных криптографических методов и алгоритмов.

Алгоритм – это формально описанная вычислительная процедура, получающая исходные данные, называемые также входом алгоритма или его аргументом, и выдающая результат вычислений на выход [4].

Чаше всего, говоря о сложности, имеют в виду *временную сложность алгоритма*: T . Временем работы алгоритма называется число элементарных шагов, которые он выполняет. Под элементарными шагами понимают такие базисные операции, как сложение, умножение, замены, безусловные передачи управления и вызовы подпрограмм. T обычно представляется в виде функции от n , где n – это размер входных данных. *Функция времени вычислений* – $T_A(n)$.

Обычно вычислительная сложность алгоритма выражается с помощью нотации "О большого", т.е. описывается порядком величины вычислительной сложности. Это просто член разложения функции сложности, быстрее всего растущий с ростом n , все члены низшего порядка игнорируются. Например, если временная сложность данного алгоритма равна $4n^2 + 7n + 2$, то вычислительная сложность порядка n^2 , записываемая как $O(n^2)$ [4].

Эта нотация позволяет увидеть, как объем входных данных влияет на требования к времени и объему памяти. Например, если $T = O(n)$, то удвоение входных данных удвоит и время выполнения алгоритма. Если $T = O(2^n)$, то добавление одного бита к входным данным удвоит время выполнения алгоритма.

Обычно алгоритмы классифицируются в соответствии с их временной или пространственной сложностью. Алгоритм называют **постоянным**, если его сложность не зависит от n : $O(1)$. Алгоритм является **линейным**, если его временная сложность $O(n)$. Алгоритмы могут быть **квадратичными**, **кубическими** и т.д. Все эти алгоритмы – **полиномиальные**, их сложность $O(n^m)$, где m – константа.

Алгоритмы с полиномиальной временной сложностью называются алгоритмами с **полиномиальным временем**.

Алгоритмы, сложность которых равна $O(t^{f(n)})$, где t – константа, большая, чем 1, а $f(n)$ – некоторая полиномиальная функция от n , называются **экспоненциальными**. Подмножество экспоненциальных алгоритмов, сложность которых равна $O(c^{f(n)})$, где c – константа, а $f(n)$ возрастает быстрее, чем постоянная, но медленнее, чем линейная функция, называется **суперполиномиальным**. Алгоритмы, сложность которых равна $O(n!)$, называются алгоритмами с факториальной сложностью. Таким образом, справедливо следующее соотношение:

$$O(\log n) < O(n) < O(n \log n) < O(n^2) < O(n^3) < O(2^n) < O(n!).$$

В настоящее время алгоритм считается практически полезным только в том случае, если его временная функция растет полиномиально относительно размеров входных данных, причем ограниченной степени. Практичными являются алгоритмы сложности $O(n)$, $O(n^2)$. Не практичны экспоненциальные и факториальные алгоритмы. Их сложность превосходит любую полиномиальную оценку.

9.2. Постановка задачи поиска

В практической деятельности приходится решать такие прикладные задачи, в которых необходимо локально или автономно произвести поиск нужных объектов. Рассмотрим задачу поиска элемента в массиве.

Пусть задан массив, состоящий из одного или нескольких элементов любого типа, отличного от файлового. Предположим, что в некотором массиве хранится множество из n ($n \times m$) элементов и необходимо определить положение некоторого элемента в этом массиве. Каждый элемент массива имеет индекс (индексы), причем индексы различны и однозначно идентифицируют элементы массива.

Задача поиска в этом случае состоит в отыскании индекса (индексов) элемента по заданному свойству элемента.

Существуют два возможных результата поиска:

- поиск оказался удачным, т.е. позволил определить положение элемента в массиве;
- поиск оказался неудачным, т.е. достигнут конец массива, но элемент с заданным свойством отсутствует.

9.3. Последовательный (линейный) поиск

Простейшим методом поиска является последовательный просмотр таблицы, одномерного массива или строки до нахождения элемента с требуемыми свойствами или установления факта, что такого элемента нет. Для массива, данные в котором не упорядочены сортировкой, единственный путь поиска заданного элемента состоит в сравнении каждого элемента массива с заданным. При совпадении некоторого элемента массива с заданным, его позиция в массиве фиксируется. Этот алгоритм называется **последовательным**, или **линейным**, поиском. Ядром такого алгоритма является цикл по индексу массива. Так, если в массиве `int a[n]` нужно найти введенное с клавиатуры целое число x , о котором никакой

дополнительной информации нет, то соответствующий цикл можно записать следующим образом:

```
for (i = 0; i < n; i++) if (a[i] == x) k = i;
```

Этот способ решения обладает двумя существенными недостатками:

- если значение x встречается в массиве несколько раз, то будет найдено последнее из них;
- после того, как нужное значение уже найдено, массив просматривается до конца, то есть всегда выполняется n сравнений.

Для устранения этих недостатков необходимо прервать просмотр массива сразу после обнаружения заданного числа. Так как в этом случае число повторений не известно, необходимо использовать цикл с предусловием:

```
while ((i < n) && (a[i] != x)) i++;
```

9.4. Бинарный поиск

Наиболее эффективным и распространенным методом непоследовательного поиска в одномерных *упорядоченных* массивах является **бинарный** поиск, который называют также методом *половинного деления*, методом *дихотомии*, методом *логарифмического* поиска или методом *деления пополам*. Основная идея бинарного поиска состоит в том, что сначала искомый элемент x сравнивается со средним элементом массива $a[n]$. Результат сравнения либо приводит к решению задачи, либо позволяет определить, в какой части массива – левой или правой – продолжать поиск. После каждой такой итерации область поиска сокращается вдвое и не более чем через $\lceil \log_2 n \rceil$ сравнений мы закончим поиск, то есть либо попадем на элемент x , либо покажем, что x не принадлежит массиву $a[n]$. Например, при $n = 1000$ бинарный поиск в 100 раз быстрее последовательного, а при $n = 1000000$ – в 50000 раз. Разумеется, если массив неупорядоченный, то применить к нему бинарный поиск нельзя. Таким образом, факт упорядоченности элементов в массиве играет ключевую роль при бинарном поиске.

9.5. Постановка задачи сортировки данных

От порядка расположения данных в памяти ЭВМ, во многом зависит скорость и простота выполнения алгоритмов, предназначенных для обработки этих данных. Поэтому в программировании часто возникает задача перегруппировки данных в невозрастающем или неубывающем, порядке. Такая задача называется упорядочением или сортировкой элементов данной структуры, в простейшем случае – элементов одномерного массива.

Задача сортировки связана со многими важными приложениями, кроме того, служат хорошей иллюстрацией анализа сложности алгоритмов и тем самым позволяют разумно выбирать лучшие среди, казалось бы, равноценных методов. Каждый из алгоритмов сортировки имеет свои достоинства и недостатки, и выбирать алгоритмы нужно, исходя из конкретной постановки задачи. Выбор алгоритма сортировки существенно зависит от структуры обрабатываемых данных, поэтому все методы сортировки подразделяют на два класса: внутренний (сортировка массивов) и внешний (сортировка последовательных файлов). Различие: при внутренней сортировке все данные хранятся в оперативной памяти ЭВМ, а при внешней - во

внешней памяти. При внутренней сортировке имеются более гибкие возможности для построения алгоритмов, так как все данные выстроены в виде массива и как бы лежат перед пользователем, он видит каждый элемент и имеет к нему прямой доступ. При внешней сортировке доступ к данным ограничен, поскольку они из-за своего размера в оперативной памяти не помещаются.

Уточним математическую постановку задачи сортировки данных [6].

Пусть надо упорядочить n элементов $m_1, m_2, \dots, m_n$, которые назовем записями. Каждой записи m_j поставим в соответствие свой ключ k_j , который и будет управлять процессом сортировки. Помимо ключа запись может содержать и дополнительную информацию, которая не влияет на сортировку, но всегда остается в этой записи. Задача заключается в *нахождении такой перестановки $m_{p_1}, m_{p_2}, \dots, m_{p_n}$, записей $m_1, m_2, \dots, m_n$, после которой ключи будут располагаться в неубывающем (невозрастающем) порядке:*

$$k_{p_1} \leq k_{p_2} \leq \dots \leq k_{p_n} \quad (k_{p_1} \geq k_{p_2} \geq \dots \geq k_{p_n})$$

Сортировка называется **устойчивой**, если она удовлетворяет дополнительному условию: записи с одинаковыми ключами остаются в прежнем порядке, т.е. $p_i < p_j$, если $k_{p_i} = k_{p_j}$ и $i < j$.

Так как каждый ключ идентифицирует соответствующую запись, то эти записи можно и не определять при сортировке, рассматривая лишь их ключи (в простейшем случае, когда запись состоит лишь из сортируемых элементов, понятия записи и ключ совпадают).

При внутренней сортировке выбранный метод должен экономно использовать время работы процессора и память. Хорошие алгоритмы затрачивают на сортировку n записей время порядка $n \log n$, а мерой эффективности может служить число необходимых сравнений ключей и число перестановок записей. Эти числа являются функциями от n – числа сортируемых записей.

При сортировке массивов будем предполагать, что перестановки, переводящие элементы массива в нужный порядок, должны выполняться *на том же месте*, т.е. без использования промежуточного массива. Методы, в которых элементы из массива A передаются в результирующий массив B , представляют значительно меньший интерес.

9.6. Прямые и быстрые методы внутренней сортировки

Все методы внутренней сортировки делятся на прямые и быстрые (или улучшенные) [6].

Для прямых методов сортировки требуется порядка n^2 сравнений ключей, они более просты и удобны для объяснения характерных черт основных принципов большинства сортировок. Программы этих методов легко понимать и они короче программ быстрых алгоритмов.

Улучшенные методы сортировки требуют небольшого числа операций, но эти операции обычно сами более сложны, и поэтому для достаточно малых n прямые методы оказываются быстрее, хотя при больших значениях n их использовать не следует.

Методы внутренней сортировки можно разбить в соответствии с определяющими их принципами на три класса:

1. Сортировка вставками: элементы просматриваются по одному, и каждый новый элемент вставляется на подходящую позицию среди ранее упорядоченных элементов.

2. Сортировка с помощью выбора: сначала выделяется наименьший (или наибольший) элемент и каким-либо образом отделяется от остальных, затем выбирается наименьший (наибольший) из оставшихся элементов и т.д.

3. Сортировка с помощью обмена: последовательно просматриваются пары соседних элементов; если элементы в паре образуют инверсию (т.е. неупорядочены), то они меняются местами.

Все прямые методы сортировки фактически передвигают каждый элемент массива на каждом элементарном шаге на одну позицию. Поэтому требуют $O(n^2)$ таких шагов. В основу любых улучшенных методов положен принцип перемещения элементов на каждом шаге сортировки на большие расстояния. Такой подход позволяет существенно уменьшить время сортировки, но усложнить алгоритмы выполнения.

Улучшенный метод обмена – Шейкерная сортировка, метод Хоара (самый быстрый из известных на сегодняшний день методов внутренней сортировки), усовершенствование сортировки прямого включения – метод Шелла, прямого выбора – сортировка с помощью бинарного дерева.

9.7. Сортировка вставками

Рассмотрим процесс сортировки методом вставки (прямого включения) массива $a[1], a[2], \dots, a[n]$. Вначале в исходном массиве выделяется уже отсортированная, например по неубыванию, часть $a[1] \leq a[2] \leq \dots \leq a[i-1]$ и не отсортированная $a[i], a[i+1], \dots, a[n]$. Затем с шагом 1, начиная с $i = 2$ чередуя сравнения и перемещения элемента $a[i]$, в отсортированной части массива определяется такой индекс (ключ) j ($1 \leq j \leq i-1$), чтобы выполнялись неравенства $a[j-1] \leq a[i] \leq a[j]$. Если такой индекс найден, то элемент $a[i]$ вставляется на соответствующее место. Для этого выполняется сдвиг на одну позицию вправо элементов отсортированной части: $a[j], \dots, a[i-1]$. После этого осуществляется переход к следующему значению i . В противном случае просто происходит переход к следующему значению i .

Покажем процесс сортировки вставками на примере целочисленного массива $a[1..12]$, состоящего из чисел 22, -14, 13, 28, 6, 41, -5, 18, 0, 35, 6, -15 (значком * обозначены вставленные элементы, выделены элементы отсортированной части)

Исходный массив:	22	-14	13	28	6	41	-5	18	0	35	6	-15
1-й шаг	-14*	22	13	28	6	41	-5	18	0	35	6	-15
2-й шаг:	-14	13*	22	28	6	41	-5	18	0	35	6	-15
3-й шаг:	-14	13	22	28*	6	41	-5	18	0	35	6	-15
4-й шаг:	-14	6*	13	22	28	41	-5	18	0	35	6	-15
5-й шаг:	-14	6	13	22	28	41*	-5	18	0	35	6	-15
6-й шаг:	-14	-5*	6	13	22	28	41	18	0	35	6	-15
7-й шаг:	-14	-5	6	13	18*	22	28	41	0	35	6	-15

8-й шаг:	-14	-5	0*	6	13	18	22	28	41	35	6	-15
9-й шаг:	-14	-5	0	6	13	18	22	28	35*	41	6	-15
10-й шаг:	-14	-5	0	6	6*	13	18	22	28	35	41	-15
11-й шаг:	-15*	-14	-5	0	6	6	13	18	22	28	35	41

С каждым шагом отсортированная часть массива увеличивается на один элемент. Очевидно, что для выполнения полной сортировки потребуется $n - 1$ шаг, где n - число элементов исходного массива.

Как видно из примера, в процессе сортировки на первом шаге первый элемент исходного массива считается отсортированным, а следующий за ним элемент сравнивается с ним. Если этот элемент больше, он остается на своем месте, в противном случае первый элемент сдвигается на вторую позицию, а второй помещается на первую (как в нашем примере). Далее все выбираемые элементы из неотсортированной части массива последовательно сравниваются с элементами отсортированной части, начиная с первого, до тех пор, пока не встретится больший или равный ему. Этот элемент и все последующие элементы отсортированной части перемещаются на одну позицию вправо (или вниз, если массив расположен в столбик), освобождая место для нового элемента. Этот процесс закончится при выполнении одного из двух условий:

- найден первый слева элемент $a[j] \geq a[i]$, что говорит о необходимости вставки $a[i]$ между $a[j]$ и $a[j-1]$, в частности, если $j=1$, то $a[i]$ помещается в первую позицию, что влечет за собой необходимость увеличения диапазона индекса массива на единицу;
- достигнут правый конец отсортированной части массива (правый барьер), следовательно, элемент $a[i]$ нужно оставить на прежнем месте.

Заметим, что сравнение и перемещение элемента $a[i]$ в отсортированной части массива можно проводить и справа налево, начиная с $(i-1)$ -го элемента.

9.8. Сортировка с помощью прямого выбора

Сортировка массива размером n в неубывающем (невозрастающем) порядке методом прямого выбора выполняется по следующему алгоритму.

1. Выбирается элемент данного массива с минимальным (максимальным) значением (или ключом).
2. Выбранный элемент и первый элемент меняются местами.
3. Этот процесс повторяется с оставшимися $n - 1$ элементами, затем с $n - 2$ элементами и т.д. до тех пор, пока не останется один, самый большой/меньший элемент.

Всего потребуется $n - 1$ раз выполнить указанную последовательность действий. В процессе сортировки будет увеличиваться отсортированная часть массива, а неотсортированная соответственно уменьшаться.

В отличие от сортировки вставками, когда на каждом шаге рассматривается только один очередной элемент исходного массива и **все** элементы отсортированной части, среди которых ищется место для вставки, при прямом выборе для поиска одного элемента с наименьшим (наибольшим) ключом просматриваются все элементы исходного массива и найденный элемент помещается как очередной в отсортированную часть массива.

Напомним, что во всех рассматриваемых методах сортировки мы не используем промежуточных массивов.

Схему сортировки методом прямого выбора проиллюстрируем на примере массива, рассмотренного в предыдущем пункте. Элементы, которые меняются местами на очередном шаге, выделены полужирным шрифтом, а минимальные элементы неотсортированной части, кроме того, отмечены звездочкой. Отсортированная часть подчеркнута.

Исходный массив:	22	-14	13	28	6	41	-5	18	0	35	6	-15*
1-й шаг	<u>-15</u>	-14*	13	28	6	41	-5	18	0	35	6	22
2-й шаг:	<u>-15</u>	<u>-14</u>	13	28	6	41	-5*	18	0	35	6	22
3-й шаг:	<u>-15</u>	<u>-14</u>	<u>-5</u>	28	6	41	13	18	0*	35	6	22
4-й шаг:	<u>-15</u>	<u>-14</u>	<u>-5</u>	<u>0</u>	6*	41	13	18	28	35	6	22
5-й шаг:	<u>-15</u>	<u>-14</u>	<u>-5</u>	<u>0</u>	<u>6</u>	41	13	18	28	35	6*	22
6-й шаг:	<u>-15</u>	<u>-14</u>	<u>-5</u>	<u>0</u>	<u>6</u>	<u>6</u>	13*	18	28	35	41	22
7-й шаг:	<u>-15</u>	<u>-14</u>	<u>-5</u>	<u>0</u>	<u>6</u>	<u>6</u>	<u>13</u>	18*	28	35	41	22
8-й шаг:	<u>-15</u>	<u>-14</u>	<u>-5</u>	<u>0</u>	<u>6</u>	<u>6</u>	<u>13</u>	<u>18</u>	28	35	41	22*
9-й шаг:	<u>-15</u>	<u>-14</u>	<u>-5</u>	<u>0</u>	<u>6</u>	<u>6</u>	<u>13</u>	<u>18</u>	<u>22</u>	35	41	28*
10-й шаг:	<u>-15</u>	<u>-14</u>	<u>-5</u>	<u>0</u>	<u>6</u>	<u>6</u>	<u>13</u>	<u>18</u>	<u>22</u>	<u>28</u>	41	35
11-й шаг:	<u>-15</u>	<u>-14</u>	<u>-5</u>	<u>0</u>	<u>6</u>	<u>6</u>	<u>13</u>	<u>18</u>	<u>22</u>	<u>28</u>	<u>35</u>	<u>41</u>

9.9. Сортировка с помощью прямого обмена

Алгоритм данного метода сортировки заключается в последовательных просмотрах массива от начала к концу и обмене местами *соседних* элементов, если они образуют *инверсию*.

Опишем алгоритм сортировки по **неубыванию** подробнее.

Пусть задан массив a размером n .

Начинаем просмотр массива с первой пары элементов $a[1]$ и $a[2]$. Если первый элемент этой пары больше второго, то меняем их местами, иначе оставляем их без изменения и сравниваем вторую пару элементов $a[2]$ и $a[3]$. Если $a[2] > a[3]$, то меняем их местами и т.д. На первом шаге будут просмотрены все пары элементов массива: $a[i]$ и $a[i+1]$ для i от 1 до $n-1$. В результате такого просмотра и необходимых обменов *максимальный* элемент переместится в конец массива и будет являться отсортированной частью.

На втором шаге аналогичная процедура проводится с первого до $(n-1)$ -го элемента. Тем самым второй по величине элемент массива переместится на предпоследнее место. Отсортированная часть будет содержать два элемента.

Эти действия продолжаются до тех пор, пока количество элементов в неотсортированной части массива не уменьшится до двух.

На последнем шаге выполняется упорядочение оставшихся двух элементов. После $(n-1)$ шагов массив окажется отсортированным по **неубыванию**.

Сортировку методом прямого обмена называют еще методом "пузырька". Это название происходит от образной интерпретации, при которой элементы данного массива расположены вертикально и в **процессе** выполнения сортировки на каждом шаге более легкие элементы, как пузырьки в ванне с водой, поднимаются до уровня, соответствующего их весу.

Для иллюстрации отсортируем массив по неубыванию методом простого обмена. Отсортированная часть выделена полужирным шрифтом.

Исходный массив:	22	-14	13	28	6	41	-5	18	0	35	6	-15
1-й шаг:	-14	13	22	6	28	-5	18	0	35	6	-15	41
2-й шаг:	-14	13	6	22	-5	18	0	28	6	-15	35	41
3-й шаг:	-14	6	13	-5	18	0	22	6	-15	28	35	41
4-й шаг:	-14	6	-5	13	0	18	6	-15	22	28	35	41
5-й шаг:	-14	-5	6	0	13	6	-15	18	22	28	35	41
6-й шаг:	-14	-5	0	6	6	-15	13	18	22	28	35	41
7-й шаг:	-14	-5	0	6	-15	6	13	18	22	28	35	41
8-й шаг:	-14	-5	0	-15	6	6	13	18	22	28	35	41
9-й шаг:	-14	-5	-15	0	6	6	13	18	22	28	35	41
10-й шаг:	-14	-15	-5	0	6	6	13	18	22	28	35	41
11-й шаг:	-15	-14	-5	0	6	6	13	18	22	28	35	41

Практическая часть

ВЫПОЛНЕНИЕ РАБОТЫ: в соответствии с заданиями пунктов 1-4, учитывая вариант (п. 3), составить и реализовать программу.

Задание

1. Сформировать при помощи генератора псевдослучайных чисел линейный целочисленный массив $A[20]$, таким образом, что бы элементы массива принадлежали отрезку $[-50, 50]$.
2. Выполнить в нём линейный поиск заданного элемента.
3. Отсортировать массив определённым методом в соответствии с вариантом (см. таблицу).

Метод сортировки	Вариант
Вставкой	1, 4, 7, 10, 13, 16, 19, 22
Обменом	2, 5, 8, 11, 14, 17, 20, 23
Выбором	3, 6, 9, 12, 15, 18, 21, 25

4. Выполнить бинарный поиск элемента.

Формирование массива, вывод его на экран, линейный поиск, сортировку и бинарный поиск оформить в виде соответствующих функций.

Контрольные вопросы

1. Понятие сложности алгоритмов.
2. Постановка задачи сортировки данных.
3. Прямые и быстрые методы внутренней сортировки.
4. Алгоритм сортировки массива методом вставки.
5. Алгоритм сортировки массива методом прямого выбора.
6. Алгоритм сортировки массива методом прямого обмена.
7. Понятие инверсии.

8. Постановка задачи поиска элемента в массиве.
9. Алгоритм последовательного (линейного) поиска.
10. Алгоритм бинарного поиска.

СПИСОК ЛИТЕРАТУРЫ

1. Конспект лекций.
2. [2] стр. 44-51, 73-82.

Лабораторная работа №10

Применение массивов и указателей для решения прикладных задач

ЦЕЛЬ РАБОТЫ: приобретение навыков организации ввода-вывода и обработки массива с использованием указателей и функций.

Формируемые компетенции или их части: вырабатывается навык разработки математических и информационных моделей, а также формируется способность к разработке и применению алгоритмических и программных решений в области прикладного программирования (формируется часть компетенций ОПК-3 и ПК-7).

Теоретическая часть

Указатели и массивы

10.1. Понятие статической и динамической переменной

Ранее были рассмотрены простые и структурированные типы данных, которые определяются в начале программы или подпрограммы и их структура остается неизменной в течение всей работы программы. Компилятор после анализа раздела переменных отводит каждой переменной соответствующее число ячеек и закрепляет их на все время работы блока. Размер выделяемого места зависит от типа переменной. Такие переменные создаются при входе в блок и прекращают свое существование, когда вычислительный процесс выходит из блока. Они не меняют своих размеров и называются статическими.

Однако существует класс задач, в которых память должна выделяться динамически, в процессе решения. Наиболее часто встречаются списки с заранее неизвестным числом объектов. Для таких задач используются динамические переменные. Они хранятся в некоторой области памяти, не обозначенные идентификатором, а со ссылкой на эту область. Такой вид доступа позволяет динамически занимать и освобождать память в процессе работы программы (блока). Динамические переменные образуются при помощи ссылочного типа данных. При образовании динамической переменной создается значение ссылочного типа – адрес места памяти, где находится вновь созданная переменная.

10.2. Указатели

Указатель – это переменная, значение которой равно значению адреса памяти, по которому лежит значение некоторой другой переменной. В этом смысле имя этой другой переменной отправляет к ее значению прямо, а указатель – косвенно. Ссылка на значение посредством указателя называется косвенной адресацией.

Указатели, подобно любым другим переменным, перед своим использованием должны быть объявлены. Объявление указателя имеет вид:

```
type *ptr;
```

где type - один из предопределенных или определенных пользователем типов, а ptr - указатель. Например,

```
int *countPtr, count;
```

объявляет переменную `countPtr` типа `int *` (т.е. указатель на целое число) и переменную `count` целого типа. Символ `*` в объявлении относится только к `countPtr`. Каждая переменная, объявляемая как указатель, должна иметь перед собой знак «звездочка» (`*`). Если в приведенном примере желательно, чтобы и переменная `count` была указателем, надо записать:

```
int *countPtr, *count;
```

Символ `*` в этих записях обозначает операцию косвенной адресации.

Может быть объявлен и указатель на `void`:

```
void *Pv;
```

Это универсальный указатель на любой тип данных. Но прежде, чем его использовать, ему надо в процессе работы присвоить значение указателя на какой-то конкретный тип данных. Например:

```
Pv = countPtr;
```

В Си указатели используются широко. Они должны инициализироваться либо при своем объявлении, либо с помощью оператора присваивания. Указатель может получить в качестве начального значения `0`, `NULL` или адрес. Указатель с начальным значением `0` или `NULL` ни на что не указывает. `NULL` – это символическая константа, определенная с целью показать, что данный указатель ни на что не указывает. Пример объявления указателя с его инициализацией:

```
int *countPtr = NULL;
```

Для присваивания указателю адреса некоторой переменной используется операция адресации `&`, которая возвращает адрес своего операнда. Например, если имеются объявления

```
int Step = 5;  
int *ptr, Num;
```

то оператор

```
ptr = &Step;
```

присваивает адрес переменной `Step` указателю `ptr`.

Для того чтобы получить значение, на которое указывает указатель, используется операция `*`, обычно называемая операцией косвенной адресации или операцией разыменования. Она возвращает значение объекта, на который указывает ее операнд (т.е. указатель). Например, если продолжить приведенный выше пример, то оператор

```
Num = *ptr;
```

присвоит переменной `Num` значение `5`, т.е. значение переменной `Step`, на которую указывает `ptr`.

Операцию разыменования нельзя применять к указателю на `void`, поскольку для него неизвестно, какой размер памяти надо разыменовывать.

Здесь необходимо обратить внимание на различие символа «`*`». Если символ «`*`» стоит перед идентификатором переменной при ее объявлении, то смысл этой «звездочки» – сказать, что объявляется не обычная переменная, а переменная-указатель. Во всех остальных случаях символ «`*`» читается и

воспринимается программой как «взятие значения по адресу», то есть является операцией косвенной адресации.

10.3. Взаимосвязь между массивами и указателями

Массивы и указатели в Си тесно связаны и могут быть использованы почти эквивалентно. Имя массива можно понимать как константный указатель на первый элемент массива. Его отличие от обычного указателя только в том, что его нельзя модифицировать. Например, задан целочисленный массив А, состоящий из 5 элементов:

```
int A[5]={-2,5,4,-17,3};
```

Посмотрим, как этот массив разместится в оперативной памяти (рис. 10.1).

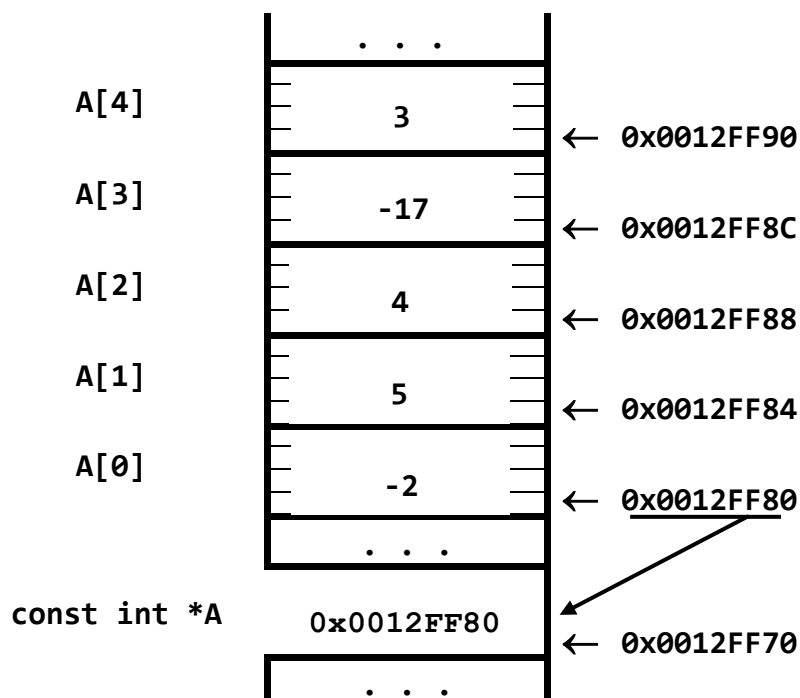


Рисунок 10.1 – Представление массива элементов в памяти ЭВМ

Стоит обратить внимание на то, что под хранение значения каждого элемента массива выделяется четыре 8 битных ячейки памяти. На рисунке 10.1 справа видна последовательность адресов ячеек памяти, содержащая значения, различающиеся на четыре байта. Это связано с тем, что тип элементов данного массива – `int`. Причем, под хранение адреса начала массива выделяется тоже четыре байта, так как все указатели для 32-разрядных процессоров 4-х байтные.

Указатели можно использовать для выполнения любой операции, включая индексирование массива. Допустим, сделано следующее объявление:

```
int b[5] = {1,2,3,4,5}, *Pt;
```

Здесь объявлены массив целых чисел `b[5]` и указатель на целое `Pt`. Поскольку имя массива является указателем на первый элемент массива, то можно задать указателю `Pt` адрес первого элемента массива `b` с помощью оператора `Pt = b;`

Это эквивалентно присваиванию адреса первого элемента массива следующим образом: `Pt = &b[0];`

Теперь можно сослаться на элемент массива `b[3]` с помощью выражения `*(Pt + 3)`.

Указатели можно индексировать точно так же, как и массивы. Например, выражение `Pt[3]` ссылается на элемент массива `b[3]`.

Или, например, определение `int a[5];` задает массив из пяти элементов `a[0]`, `a[1]`, `a[2]`, `a[3]`, `a[4]`. Если объект `*y` определен как

```
int *y;    то оператор    y = &a[0];
```

присваивает переменной `y` адрес элемента `a[0]`. Если переменная `y` указывает на очередной элемент массива `a`, то `y+1` указывает на следующий элемент, причем здесь выполняется соответствующее масштабирование для приращения адреса с учетом длины объекта (для типа `int` – 4 байта, `long` – 4 байта, `double` – 8 байт и т.д.). Так как само имя массива есть адрес его нулевого элемента, то оператор `y = &a[0];` можно записать и в другом виде: `y = a`. Тогда элемент `a[1]` можно представить как `*(a+1)`. С другой стороны, если `y` – указатель на массив `a`, то следующие две записи: `a[i]` и `*(y+i)` эквивалентны. Рассмотрим пример:

```
int main()
{
    int a[5]={-5,0,34,12,-17};
    cout<<endl<<(a+2); // оператор 1
    cout<<endl<<&a[2]; // оператор 2
    cout<<endl<<*(a+2); // оператор 3
    cout<<endl<<a[2];   // оператор 4
    cout<<endl;
    return 0;
}
```

Так, операторы 1 и 2 выведут одно и то же значение – адрес элемента массива с индексом 2. А операторы 3 и 4 – одно и то же значение 34.

Между именем массива и соответствующим указателем есть одно важное различие. Указатель – это переменная и `y = a` или `y++` – допустимые операции. Имя же массива – константа, поэтому конструкции вида `a = y`, `a++` использовать нельзя, так как значение константы постоянно и не может быть изменено.

Указатели могут применяться как операнды в арифметических выражениях, выражениях присваивания и выражениях сравнения. Однако, не все операции, обычно используемые в этих выражениях, разрешены применительно к переменным указателям.

С указателями может выполняться ограниченное количество арифметических операций. Указатель можно увеличивать (`++`), уменьшать (`--`), складывать с указателем целые числа (`+` или `+=`), вычитать из него целые числа (`-` или `-=`) или вычитать один указатель из другого.

Сложение указателей с целыми числами отличается от обычной арифметики. Прибавить к указателю 1 означает сдвинуть его на число байтов, содержащихся в переменной, на которую он указывал. Обычно подобные операции применяются к указателям на массивы. Если продолжить приведенный выше пример, в котором указателю `Pt` было присвоено значение `b` – указателя на первый элемент массива, то после выполнения оператора

```
Pt += 2;
```

`Pt` будет указывать на третий элемент массива `b`. Истинное же значение указателя `Pt` изменится на число байтов, занимаемых одним элементом массива, умноженное на 2. Например, если каждый элемент массива `b` занимает 2 байта, то значение `Pt` (т.е. адрес в памяти, на который указывает `Pt`) увеличится на 4.

Аналогичные правила действуют и при вычитании из указателя целого значения.

Переменные указатели можно вычитать один из другого. Например, если `Pt` указывает на первый элемент массива `b`, а указатель `Pt1` – на третий, то результат выражения `Pt1 - Pt` будет равен 2 – разности индексов элементов, на которые указывают эти указатели. И так будет, несмотря на то, что адреса, содержащиеся в этих указателях, различаются на 4 (если элемент массива занимает 2 байта).

Арифметика указателей теряет всякий смысл, если она выполняется не над указателями на массив. Сравнение указателей операциями `>`, `<`, `>=`, `<=` также имеют смысл только для указателей на один и тот же массив. Однако, операции отношения `==` и `!=` имеют смысл для любых указателей. При этом указатели равны, если они указывают на один и тот же адрес в памяти.

10.4. Порядок объявления динамических массивов

Динамические массивы создают с использованием операции `new`, при этом необходимо указать тип и размерность, например:

```
int a=10;
int *ptr=new int [a];
```

В этой строке создается переменная-указатель на `int`, в динамической памяти отводится непрерывная область, достаточная для размещения 10 элементов целого типа, и адрес ее начала записывается в указатель `ptr`. Динамические массивы нельзя при создании инициализировать, и они не обнуляются.

Доступ к элементам динамического массива осуществляется точно так же, как к статическим, например, к элементу номер 5 можно обратиться как `ptr[5]` или `*(ptr+5)`.

Альтернативный способ создания динамического массива – использование функции `malloc` библиотеки Си:

```
int a=10;
int *ptr=(int*)malloc(a*sizeof(int));
```

Операция преобразования типа, записанная перед обращением к функции `malloc`, требуется потому, что функция возвращает значение указателя типа `void*`.

Память, зарезервированная под динамический массив с помощью `new[]`, должна освобождаться оператором `delete []`, а память, выделенная функцией `malloc` – посредством функции `free`, например:

```
delete [] ptr;
free(ptr);
```

Размерность массива в операции `delete` не указывается, но квадратные скобки обязательны.

10.5. Передача массивов в качестве параметров функции

При использовании массива в качестве параметра в функцию передается указатель на его первый элемент, иными словами, массив всегда передается по адресу. При этом информация о количестве элементов массива теряется, и следует передавать его размерность через отдельный параметр (в случае массива символов, то есть строки, ее фактическую длину можно определить по значению нуля-символа):

```
#include <iostream>

int sum(const int *mas, const int n);
int const n = 10;

int main()
{
    int marks[n]= {3, 4, 5, 4, 4};
    cout<<"сумма элементов массива: "<<sum(marks,n);
    return 0;
}

int sum(const int *mas, const int n) //(величина n должна быть константой)
{
    int s =0;
    for (int i=0; i<n; i++) s += mas[i];
    return s;
}
```

При передаче многомерных массивов все размеры, если они не известны на этапе компиляции, должны передаваться в качестве параметров. Внутри функции массив интерпретируется как одномерный, а его индекс пересчитывается в программе. В приведенном ниже примере с помощью функции подсчитывается сумма элементов двух двумерных массивов. Размерность массива b известна на этапе компиляции, под массив a память выделяется динамически.

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

using namespace std;
int sum(const int *a, const int nstr, const int nstb);

int main()
{
    int b[2][2] = {{2,2},{4,3}};
    printf(сумма элементов b: %d\n",sum(&b[0][0],2,2));
    //имя массива передавать в sum нельзя из-за несоответствия типов

    int i,j, nstr, nstb,*a;
    print f("введите количество строк и столбцов: \n");
    scanf ("%d%d",&nstr, &nstb);

    // формирование матрицы: выделение памяти
    a = (int *)malloc(nstr * nstb * sizeof(int));
    // ввод элементов массива
```

```

        for (i=0; i<nstr; i++)
            for (j=0; j<nstb; j++) scanf("%d",&a[i * nstb+j]);

        // вывод на экран результата работы функции sum
        printf("сумма элементов a: %d\n", sum(a,nstr,nstb));

        //высвобождение памяти
        free(a);

        _getch();
        return 0;
    }

int sum(const int *a,const int nstr, const int nstb)
{
    int i,j,s=0;
    for (i=0;i<nstr; i++)
        for (j=0;j<nstb; j++) s+=a[i * nstb + j];
    return s;
}

```

Для того чтобы работать с двумерным массивом естественным образом, можно применить альтернативный способ выделения памяти:

```

#include <conio.h>
#include <iostream>
#include <stdio.h>
#include <stdlib.h>

using namespace std;
int sum(int **a, const int nstr, const int nstb);

int main()
{
    int nstr, nstb;
    cin>>nstr>>nstb;
    int **a, i, j;

    // формирование матрицы: выделение памяти
    a = new int* [nstr]; // nstr строк в массиве
    for (i = 0; i < nstr; i++)
        a[i] = new int [nstb]; //и nstb столбцов

    // ввод элементов массива
    for(i=0; i<nstr; i++)
        for(j=0; j<nstb; j++) cin>>a[i][j];

    // вывод на экран результата работы функции sum
    cout<<sum(a, nstr, nstb);

    //высвобождение памяти отводимой под двумерный динамический массив:
    for (i = 0; i < nstr; i++) //где nstr – количество строк в массиве
        delete [] a[i]; // удалили массив указателей
    delete [] a; // удалили указатель на массив указателей

    _getch();
    return 0;
}

int sum(int **a, const int nstr, const int nstb)

```

```

{
int i, j, s=0;
for(i=0; i<nstr; i++)
    for(j=0; j<nstb; j++) s+=a[i][j];
return s;
}

```

В этом случае память выделяется в два этапа: сначала под столбец указателей на строки матрицы, а затем в цикле под каждую строку. Освобождение памяти должно выполняться в обратном порядке.

Практическая часть

ВЫПОЛНЕНИЕ РАБОТЫ: освоить теоретический материал, выполнить общее для всех задание I и в соответствии с вариантом составить программу (задание II, III), при необходимости реализовав пользовательские функции.

Задание I

Изучить порядок описания и использования указателей:

- набрать и отладить следующую программу:

```

int main()
{
    int a=7, b=2;
    int *ptr=NULL;
    cout<<"\na="<<a<<", &a="<<&a;
    cout<<"\n b="<<b<<", &b="<<&b;
    ptr=&a;
    cout<<"\nДействие: ptr=&a. Результат: ptr="<<ptr<<", *ptr="<<*ptr;
    ptr=&b;
    cout<<"\nДействие: ptr=&b. Результат: ptr="<<ptr<<", *ptr="<<*ptr;
    *ptr=137;
    cout<<"\nДействие: *ptr=137. Результат: b="<<b;
    ptr++;//или ptr+=2;
    *ptr=-105;
    cout<<"\nДействие: ptr++, *ptr=-105. Результат: ptr="
        <<ptr<<", *ptr="<<*ptr<<endl;
    return 0;
}

```

- предъявить результат и уметь объяснить работу программы.

Задание II

Реализовать с использованием указателей и механизма функций в соответствии с номером варианта:

1. В одномерном массиве, состоящем из n вещественных элементов, вычислить:
 - сумму отрицательных элементов массива;
 - произведение элементов массива, расположенных между максимальным и минимальным элементами.
2. В одномерном массиве, состоящем из n вещественных элементов, вычислить:

- сумму положительных элементов массива;
 - произведение элементов массива, расположенных между максимальным по модулю и минимальным по модулю элементами.
3. В одномерном массиве, состоящем из n целых элементов, вычислить:
 - сумму элементов массива, расположенных между первым и последним нулевыми элементами;
 - преобразовать массив таким образом, чтобы сначала располагались все положительные элементы, а потом – все отрицательные (элементы, равные 0, считать положительными); порядок следования элементов массива в выделенных группах не менять.
 4. В одномерном массиве, состоящем из n вещественных элементов, вычислить:
 - сумму элементов массива с нечетными номерами;
 - сжать массив, удалив из него все элементы, модуль которых не превышает 1.
 5. В одномерном массиве, состоящем из n вещественных элементов, вычислить:
 - максимальный элемент массива;
 - сумму элементов массива, расположенных до последнего положительного элемента.
 6. В одномерном массиве, состоящем из n вещественных элементов, вычислить:
 - минимальный элемент массива;
 - сумму элементов массива, расположенных между первым и последним положительными элементами.
 7. В одномерном массиве, состоящем из n целых элементов, вычислить:
 - номер максимального элемента массива;
 - преобразовать массив таким образом, чтобы в первой его половине располагались элементы, стоявшие в нечетных позициях, а во второй половине – элементы, стоявшие в четных позициях.
 8. В одномерном массиве, состоящем из n вещественных элементов, вычислить:
 - сумму элементов массива, расположенных между первым и вторым отрицательными элементами.
 - преобразовать массив таким образом, чтобы сначала располагались все элементы, модуль которых не превышает 1, а потом – все остальные; порядок следования элементов массива в выделенных группах не менять.
 9. В одномерном массиве, состоящем из n вещественных элементов, вычислить:
 - максимальный по модулю элемент массива;
 - преобразовать массив таким образом, чтобы элементы, равные нулю, располагались после всех остальных.
 10. В одномерном массиве, состоящем из n целых элементов, вычислить:
 - минимальный по модулю элемент массива;
 - сумму модулей элементов массива, расположенных после первого элемента, равного нулю.
 11. В одномерном массиве, состоящем из n вещественных элементов, вычислить:
 - номер минимального по модулю элемента массива;
 - сумму модулей элементов массива, расположенных после первого отрицатель-

ного элемента.

12. В одномерном массиве, состоящем из n вещественных элементов, вычислить:
 - номер максимального по модулю элемента массива;
 - сумму элементов массива, расположенных после первого положительного элемента.
13. В одномерном массиве, состоящем из n вещественных элементов, вычислить:
 - количество элементов массива, лежащих в диапазоне от A до B ;
 - сумму элементов массива, расположенных после максимального элемента.
14. В одномерном массиве, состоящем из n вещественных элементов, вычислить:
 - количество элементов массива, равных 0;
 - сумму элементов массива, расположенных после минимального элемента.
15. В одномерном массиве, состоящем из n вещественных элементов, вычислить:
 - количество элементов массива, больших C ;
 - преобразовать массив таким образом, чтобы сначала располагались все отрицательные элементы, а потом – все положительные (элементы, равные 0, считать положительными) ; порядок следования элементов массива в выделенных группах не менять.
17. В одномерном массиве, состоящем из n целых элементов, вычислить:
 - сумму элементов массива, расположенных между первым и последним нулевыми элементами.
 - преобразовать массив таким образом, чтобы сначала располагались все положительные элементы, а потом – все отрицательные (элементы, равные 0, считать положительными); порядок следования элементов массива в выделенных группах не менять.
18. В одномерном массиве, состоящем из n вещественных элементов, вычислить:
 - сумму элементов массива с нечетными номерами;
 - сжать массив, удалив из него все элементы, модуль которых не превышает 1. Освободившиеся в конце массива элементы заполнить нулями.
19. В одномерном массиве, состоящем из n вещественных элементов, вычислить:
 - максимальный элемент массива;
 - сумму элементов массива, расположенных до последнего положительного элемента.
20. В одномерном массиве, состоящем из n вещественных элементов, вычислить:
 - минимальный элемент массива;
 - сумму элементов массива, расположенных между первым и последним положительными элементами.
21. В одномерном массиве, состоящем из n целых элементов, вычислить:
 - номер максимального элемента массива;
 - преобразовать массив таким образом, чтобы в первой его половине располагались элементы, стоявшие в нечетных позициях, а во второй половине – элементы, стоявшие в четных позициях.
22. В одномерном массиве, состоящем из n вещественных элементов, вычислить:
 - сумму элементов массива, расположенных между первым и вторым отрица-

- тельными элементами.
- преобразовать массив таким образом, чтобы сначала располагались все элементы, модуль которых не превышает 1, а потом – все остальные.
23. В одномерном массиве, состоящем из n вещественных элементов, вычислить:
 - максимальный по модулю элемент массива;
 - преобразовать массив таким образом, чтобы элементы, равные нулю, располагались перед всеми остальными элементами.
 24. В одномерном массиве, состоящем из n целых элементов, вычислить:
 - минимальный по модулю элемент массива;
 - сумму модулей элементов массива, расположенных после первого элемента, равного нулю.
 25. В одномерном массиве, состоящем из n вещественных элементов, вычислить:
 - номер минимального по модулю элемента массива;
 - сумму модулей элементов массива, расположенных после первого отрицательного элемента.
 26. В одномерном массиве, состоящем из n вещественных элементов, вычислить:
 - номер максимального по модулю элемента массива;
 - сумму элементов массива, расположенных после первого положительного элемента.

Задание III

В соответствии с вариантом, используя указатели и пользовательские функции, составить и реализовать программу:

Элементы вещественного одномерного динамического массива x размером $n > 20$ сгенерированы датчиком псевдослучайных чисел в диапазоне $[-10, 10]$. Сформировать массив y по формуле и выполнить указанные вычисления. Формулу записать с помощью математических символов в привычном виде. Вывести на экран исходный массив и результаты решения задачи. Все числа округлять до сотых. Анализ результатов выполнить с использованием известных математических пакетов.

1. $y[i] = \exp(x[i] - i)$, $i = 0, 1, \dots, n-1$, $n = 2k$. Вычислить сумму произведений $P = x[0]y[2k-1] + x[1]y[2k-2] + \dots + x[2k-1]y[0]$.
2. $y[i] = x[i] / \ln(\text{abs}(x[i] + 1.23))$, $i = 0, 1, \dots, n-1$. Вычислить сумму отрицательных значений элементов массива x из первых 15 элементов массива y .
3. $y[i] = \exp(x[i]) / x[i]$, $i = 1, 2, \dots, n$. Вычислить сумму всех произведений соответствующих элементов массивов x и y .
4. $y[i] = \ln(x[i] + 10.4) / (x[i] + 1)$, $i = 0, 1, \dots, n-1$. Вычислить сумму положительных значений элементов массива x из последних 12 элементов массива y .
5. $y[i] = \cos(x[i]) + \sin(x[i]) + 2.6$, $i = 0, 1, \dots, n-1$. Вычислить сумму квадратов элементов массива y , стоящих на нечетных местах.

6. $n = 2k$, $y[0] = x[1]$, $y[1] = x[1] x[3]$, $y[2] = x[1] x[3] x[5]$, ..., $y[k-1] = x[1] x[3] x[5] \dots x[n-1]$, $y[k] = x[0]$, $y[k+1] = x[0] x[2]$, $y[k+2] = x[0] x[2] x[4]$, ..., $y[n-1] = x[0] x[1] \dots x[n-2]$.
Вычесть из суммы первых k элементов массива y сумму последних k его элементов.
7. $y[i] = \sqrt{\text{abs}(x[i])}$, $i = 0, 1, \dots, n-1$. Вычислить количество элементов массива y , больших среднего арифметического элементов массива x .
8. $y[0] = 5.2$, $y[1] = 4.3$, $y[2] = 0.09$, $y[3] = -0.5$, $y[4] = 7.15$;
 $y[i] = \cos(y[i-1] + y[i-2] + y[i-3]) / (y[i-4] \cdot y[i-5])$, $i = 5, 6, \dots, n-1$.
Вычислить произведение всех отрицательных чисел массива y (в этой задаче массива x нет).
9. $n = 2k$, $y[0] = x[0] x[1]$, $y[1] = x[2] x[3]$, ..., $y[k-1] = x[n-2] x[n-1]$. Вычислить сумму квадратов элементов массива y .
10. $y[i] = x[i]^{2/3}$, $i = 0, 1, \dots, n-1$. Просуммировать элементы массива y с шагом 3, начиная с первого элемента.
11. $n = 2k$, $y[i] = \sqrt{\text{abs}(\cos(x[i]) + 6.7)}$, $i = 0, 1, \dots, n-1$. Вычислить сумму произведений $S = x[0] y[1] + x[2] y[3] + \dots + x[n-2] y[n-1]$.
12. $y[i] = i \sin(x[i])$, $i = 0, 1, \dots, n-1$. Вычислить произведение положительных значений из последних 14 элементов массива x .
13. $y[i] = \sin(\exp(x[i]))$, $i = 0, 1, \dots, n-1$. Вычислить количество отрицательных элементов массива y с нечетными индексами.
14. $y[0] = \sqrt{\text{abs}(x[0] - x[1])}$, $y[1] = \sqrt{\text{abs}(x[1] - x[2])}$, ..., $y[n-3] = \sqrt{\text{abs}(x[n-3] - x[n-2])}$, $y[n-2] = \sqrt{\text{abs}(x[n-2] - x[n-1])}$.
Просуммировать элементы массива y , имеющие индекс, кратный 3.
15. $y[0] = x[0]$, $y[1] = x[0] + x[1]$, ..., $y[n-1] = x[0] + x[1] + \dots + x[n-1]$. Вычислить сумму кубов отрицательных элементов массива y .
16. $n = 2k$, $y[0] = x[0] - x[n-1]$, $y[1] = x[1] - x[n-2]$, ..., $y[k-1] = x[k-1] - x[k]$. Из каждого элемента массива y вычесть среднее арифметическое элементов массива x .
17. $y[i] = \exp(\cos(x[i]))$, $i = 0, 1, \dots, n-1$. Вычислить количество элементов массива y , попадающих в интервал $[0, 1]$.
18. $y[i] = 1/8 \text{tg}(x[i])$, $i = 0, 1, \dots, n-1$. Вычислить количество отрицательных элементов массива x и сумму обратных элементов массива y .
19. $y[0] = x[0]$, $y[n-1] = x[n-1]$, $y[i] = (x[i-1] + x[i] + x[i+1]) / 3$. Вычислить количество положительных элементов массива y с четными индексами и найти их сумму.
20. $y[i] = \exp(x[i]) \sin(x[i])$, $i = 0, 1, \dots, n-1$. Вычислить сумму элементов массива y , стоящих на четных местах.
21. $y[i] = \text{tg}(2^{x[i]})$, $i = 0, 1, \dots, n-1$. Вычислить количество элементов массива y , меньших по своим значениям соответствующих элементов массива x .

22. $y[i] = 3^{\cos(x[i])}$, $i = 0, 1, \dots, n-1$. Вычислить количество элементов массива y , не попадающих в промежуток $[0, 1]$.
23. $y[i] = x[i]^2 \sin(1/x[i])$, $i = 0, 1, \dots, n-1$. Вычислить сумму положительных значений из первых 14 элементов массива y .
24. $y[i] = \cos(x[i]) - \text{abs}(x[i])$, $i = 0, 1, \dots, n-1$. Из суммы квадратов элементов массива x вычесть сумму квадратов элементов массива y .
25. $y[0] = 4.5$, $y[1] = -3.6$, $y[2] = -8.2$, $y[3] = 6$;
 $y[i] = \ln(\text{abs}(y[i - 1])) / \sin(y[i - 2] / (y[i - 4] y[i - 3]))$, $i = 4, 5, \dots, n - 1$ (здесь массива x нет, а элементы массива y вычисляются с помощью рекуррентных соотношений). Вычислить сумму отрицательных элементов массива y .

Контрольные вопросы

1. Что такое указатель?
2. Каков общий вид объявления указателя? Приведите пример.
3. Как описать несколько указателей на данные одного и того же типа?
4. Каковы особенности инициализации указателей?
5. Что означает NULL?
6. Приведите пример использования операции взятия адреса (адресации). Поясните принцип проиллюстрированного действия.
7. Приведите пример использования операции «взятия значения по адресу» (косвенной адресации или разыменования). Поясните принцип проиллюстрированного действия.
8. Особенности использования символа «*».
9. Какова взаимосвязь между именем массива и указателем?
10. Как задать указателю адрес первого элемента массива.
11. Как получить значение i -го элемента массива при использовании указателя?
12. Каково различие между именем массива и соответствующим указателем?
13. Каков порядок объявления динамических массивов с использованием операции new? С использованием функции malloc?
14. Как освободить память, отведенную под динамический массив?
15. Каковы основные правила передачи массивов в качестве параметров функции?

Пример выполнения задания III лабораторной работы

Задание. В соответствии с вариантом, используя указатели и пользовательские функции, составить и реализовать программу:

Элементы вещественного одномерного динамического массива x размером $n > 20$ сгенерированы датчиком псевдослучайных чисел в диапазоне $[-10, 10]$. Сформировать массив y по формуле и выполнить указанные вычисления. Формулу записать с помощью математических символов в привычном виде. Вывести на экран исходный массив и результаты решения задачи. Все числа округлять до сотых. Анализ результатов выполнить с использованием известных математических пакетов.

$$y[i] = (15 + \exp(x[i] - i^2)) / (i^2 + 1), i = 0, 1, \dots, n - 1.$$

Вычислить сумму произведений элементов заданных массивов:

$$S = x[0] x[1] \dots x[n - 1] + y[0] y[1] \dots y[n - 1].$$

Решение

1. Математическая модель

Элементы вещественного одномерного динамического массива x размером $n > 20$ сгенерируем датчиком псевдослучайных чисел в диапазоне $[-10, 10]$.

Для нахождения элементов массива y воспользуемся формулой:

$$y_i = \frac{15 + e^{(x_i - i^2)}}{i^2 + 1}.$$

Вывод массива организуем при помощи вспомогательной функции, использующей форматированный вывод данных с округлением до сотых.

Требуется найти сумму произведений элементов заданных массивов:

$$S = \prod_{i=0}^{n-1} x_i + \prod_{i=0}^{n-1} y_i.$$

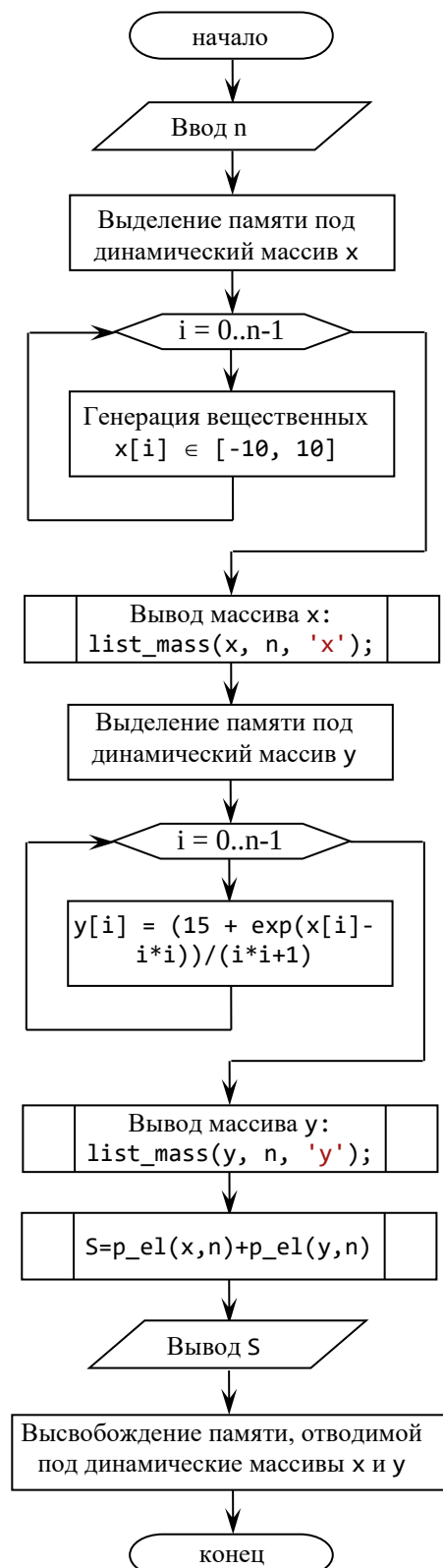
Нахождение произведения элементов массива оформим в виде вспомогательной функции.

Аргументы: вещественный одномерный динамический массив x ; целочисленное n – количество элементов массива.

Результаты: вещественный одномерный динамический массив y ; сумма произведений элементов заданных массивов S (вещественного типа двойной точности).

Промежуточные величины: счетчик i (целого типа), имя массива lit (символьного типа), произведение элементов заданных массивов p (вещественного типа двойной точности).

2. Алгоритм



Функция `double p_el(float *a, int np)` вычисляет произведение элементов массива.

Функция `void list_mass(float *a, int nl, char lit)` осуществляет форматированный вывод одномерного массива на экран

3. Программа

```

#include <conio.h>
#include <iomanip>
#include <iostream>
#include <stdio.h>
#include <time.h> // для time(0)
#include <stdlib.h> // для srand() и rand()

using namespace std;

double p_el(float *a, int ns);
void list_mass(float *a, int nl, char lit);

int main()
{
    srand(time(0));
    cout<<"Введите количество элементов массивов n = ";
    int n;
    cin>>n;
    //формирование массива x:
    float *x = new float [n];
    for (int i = 0; i < n; i++)
        x[i] = (float)(rand()%2100-1000)/100;
    list_mass(x, n, 'x');//вывод массива x
    //формирование массива y:
    float *y = new float [n];
    for (int i=0; i<n; i++)
        y[i] = (15 + exp(x[i]-i*i))/(i*i+1);
    list_mass(y, n, 'y');//вывод массива y
    double S = p_el(x,n) + p_el(y,n);
    cout<< "\n S = "<< fixed << setprecision(2) <<S;
    //высвобождение памяти, отводимой под динамические массивы:
    delete [] y;
    delete [] x;
    _getch();
    return 0;
}

double p_el(float *a, int np)
{
    double p = 1;
    for(int i=0; i<np; i++)
        p*=a[i];
    return p;
}

void list_mass(float *a, int nl, char lit)
{
    cout<<"\n"<<lit<<":\n "<<endl;
    for(int i=0; i<nl; i++)
        printf ("%5.2f\t",a[i]);
    cout<<"\n";
}
  
```

4. Результат работы программы (алгоритм и листинг программы см. на следующей странице):

Введите количество элементов массивов $n = 21$

x:

-8.42	9.06	7.00	10.45	4.15	-6.26	8.49	9.48	0.98	4.11
-6.55	-5.25	0.95	1.95	-6.64	-2.19	6.58	6.67	-2.11	-7.30
5.56									

y:

15.00	1590.15	7.02	1.93	0.88	0.58	0.41	0.30	0.23	0.18
0.15	0.12	0.10	0.09	0.08	0.07	0.06	0.05	0.05	0.04
0.04									

$S = 163630350633731.37$

5. Анализ результатов с использованием известных математических пакетов выполнить самостоятельно.

СПИСОК ЛИТЕРАТУРЫ

1. Конспект лекций.
2. [2] стр. 44-51.

Лабораторная работа № 11

Исследование способов работы с функциями

ЦЕЛЬ РАБОТЫ: совершенствование навыков в программировании с использованием указателей, исследование способов передачи параметров в функцию и порядка возвращения параметров из функций.

Формируемые компетенции или их части: вырабатывается навык разработки математических и информационных моделей, а также формируется способность к разработке и применению алгоритмических и программных решений в области прикладного программирования (формируется часть компетенций ОПК-3 и ПК-7).

Теоретическая часть

Особенности работы с функциями

11.1. Способы передачи параметров в функцию

Все переменные, объявленные в теле функции без указания класса памяти, имеют класс памяти **auto**, т.е. они являются локальными. При вызове функции локальным переменным отводится память в стеке и производится их инициализация. Управление передается первому оператору тела функции и начинается выполнение функции, которое продолжается до тех пор, пока не встретится оператор **return** или последний оператор тела функции. Управление при этом возвращается в точку, следующую за точкой вызова, а локальные переменные становятся недоступными. При новом вызове функции для локальных переменных память распределяется вновь, и поэтому старые значения локальных переменных теряются.

Параметры функции передаются по значению и могут рассматриваться как локальные переменные, для которых выделяется память при вызове функции и производится инициализация значениями фактических параметров. При выходе из функции значения этих переменных теряются. Поскольку передача параметров происходит по значению, в теле функции нельзя изменить значения переменных, являющихся фактическими параметрами в вызывающей функции. Однако, если в качестве параметра передать указатель на некоторую переменную, то используя операцию разадресации можно изменить значение этой переменной.

Пример.

```
/*      использование параметров по значению      */
void change (int x, int y);
void main()
{
    int a=5,b=15;
    cout<<"\na="<<a<<"", b="<<b;
    change(a,b);
    cout<<"\na="<<a<<"", b="<<b;
    return;
}
void change (int x, int y)
```

```

{
    int k=x;
    x=y;
    y=k;
}

```

В данной функции значения переменных *x* и *y*, являющихся формальными параметрами, меняются местами, но поскольку эти переменные существуют только внутри функции *change*, значения фактических параметров, используемых при вызове функции, останутся неизменными, то есть значения переменных *a* и *b* как были 5 и 15 соответственно, так и остались. Для того чтобы менялись местами значения фактических аргументов можно использовать функцию приведенную в следующем примере.

Пример.

```

/*    использование параметров по адресу    */
void change (int *x, int *y)
{   int k=*x;
    *x=*y;
    *y=k;}

```

При вызове такой функции в качестве фактических параметров должны быть использованы не значения переменных, а их адреса *change (&a,&b);*. Тогда функция будет оперировать не с локальными значениями, а со значениями по адресам, переданным в функцию в качестве параметров, что приведет к обмену значениями переменных *a* и *b*.

Существует еще один способ передачи параметров в функцию – по ссылке. Механизм использования ссылок очень удобен и прост в использовании. Рассмотрим тот же самый пример, но с использованием механизма ссылок.

Пример.

```

/*    использование параметров по ссылке    */
void change (int &x, int &y)
{   int k=x;
    x=y;
    y=k;}

```

При вызове такой функции в качестве фактических параметров должны быть использованы имена переменных: *change (a,b);*. При этом, в функции мы будем работать с самими переменными *a* и *b*, но через альтернативные их имена: *x* и *y*.

Любая функция по результатам своей работы может возвращать какое-либо значение. Функция не может возвращать массив или функцию, но может возвращать указатель на любой тип, в том числе и на массив и на функцию. Тип возвращаемого значения, задаваемый в определении функции, должен соответствовать типу в объявлении этой функции.

Функция возвращает значение, если ее выполнение заканчивается оператором *return*, содержащим некоторое выражение. Указанное выражение вычисляется, преобразуется, если необходимо, к типу возвращаемого значения и возвращается в точку вызова функции в качестве результата. Если оператор *return* не содержит выражения или выполнение функции завершается после выполнения последнего ее оператора (без выполнения оператора *return*), то возвращаемое зна-

чение не определено. Для функций, не использующих возвращаемое значение, должен быть использован тип `void`, указывающий на отсутствие возвращаемого значения. Если функция определена как функция, возвращающая некоторое значение, а в операторе `return` при выходе из нее отсутствует выражение, то поведение вызывающей функции после передачи ей управления может быть непредсказуемым.

11.2. Передача имен функций в качестве параметров

Функцию можно вызвать через указатель на нее. Для этого объявляется указатель соответствующего типа и ему с помощью операции взятия адреса присваивается адрес функции:

```
void f(int a){ /*...*/ }    // определение функции
void (*pf)(int);           // указатель на функцию

...
pf=&f;                      //указателю присваивается адрес функции
                             //(можно написать pf=f;)
pf(10);                     //функция f вызывается через указатель pf
                             //(можно написать (*pf)(10) )
```

Для того чтобы сделать программу легко читаемой, при описании указателей на функции используют переименование типов (**typedef**). Можно объявлять массивы указателей на функции (это может быть полезно, например, при реализации меню):

```
// Описание типа PF как указателя
// на функцию с одним параметром типа int:
typedef void (*PF)(int);

// Описание и инициализация массива указателей:
PF menu[] = {&new, &open, save};

// Вызов функции open
menu[1](10);
```

Здесь **new**, **open** и **save** - имена функций, которые должны быть объявлены ранее. Указатели на функции передаются в программу таким же образом, как и параметры других типов:

```
#include<iostream.h>
typedef void (*PF)(int);
void f1(PF pf)
{
    //функция f1 получает в качестве параметра указатель типа PF
    pf(5) //вызов функции, переданной через указатель
}
void f(int i ) {cout <<i}
int main()
{
    f1(f);
    return 0;
}
```

Тип указателя и тип функции, которая вызывается посредством этого указателя, должны совпадать в точности.

11.3. Перегрузка функций

Часто бывает удобно, чтобы функции, реализующие один и тот же алгоритм для различных типов данных, имели одно и то же имя. Если это имя мнемонично, то есть несет нужную информацию, это делает программу более понятной, поскольку для каждого действия требуется помнить только одно имя.

Использование нескольких функций с одним и тем же именем, но с различными типами параметров, называется **перегрузкой функций**.

Компилятор определяет, какую именно функцию требуется вызвать, по типу фактических параметров. Этот процесс называется **разрешением перегрузки** (перевод английского слова *resolution* в смысле «уточнение»). Тип возвращаемого функцией значения в разрешении не участвует. Механизм разрешения основан на достаточно сложном наборе правил, смысл которых сводится к тому, чтобы использовать функцию с наиболее подходящими аргументами и выдать сообщение, если такой не найдется. Допустим, имеется четыре варианта функции, определяющей наибольшее значение:

```
// Возвращает наибольшее из двух целых:
int max(int, int);
// Возвращает подстроку наибольшей длины:
char* max(char*, char*);
// Возвращает наибольшее из первого параметра и длины второго:
int max (int, char*);
// Возвращает наибольшее из второго параметра и длины первого:
int max (char*, int);
void f(int a, int b, char* c, char* d)
{
    cout << max (a,b) << max (c,d) << max (a,c) << max (c,b);
}
```

При вызове функции `max` компилятор выбирает соответствующий типу фактических параметров вариант функции (в приведенном примере будут последовательно вызваны все четыре варианта функции).

Если точного соответствия не найдено, выполняются продвижения порядковых типов в соответствии с общими правилами (см. [2] с. 38 и приложение 3), например, `bool` и `char` в `int`, `float` и `double` и т.п. Далее выполняются стандартные преобразования типов, например, `int` в `double` или указателей в `void*`. Следующим шагом является выполнение преобразований типа, заданных пользователем (об этих преобразованиях рассказывается во второй части [2], с. 195), а также поиск соответствий за счет переменного числа аргументов функций. Если соответствие на одном и том же этапе может быть получено более чем одним способом, вызов считается неоднозначным и выдается сообщение об ошибке.

Неоднозначность может появиться при:

- преобразовании типа;
- использовании параметров-ссылок;
- использовании аргументов по умолчанию.

Ниже приведены правила описания перегруженных функций.

- Перегруженные функции должны находиться в одной области видимости, иначе произойдет сокрытие аналогично одинаковым именам переменных во вложенных блоках.
- Перегруженные функции могут иметь параметры по умолчанию, при этом значения одного и того же параметра в разных функциях должны совпадать. В различных вариантах перегруженных функций может быть различное количество параметров по умолчанию.

Функции не могут быть перегружены, если описание их параметров отличается только модификатором **const** или использованием ссылки (например, **int** и **const int** или **int&**).

Практическая часть

ВЫПОЛНЕНИЕ РАБОТЫ: освоить теоретический материал, выполнить общее для всех задание I и в соответствии с вариантом составить программу (задание II).

Задание I

Исследовать механизм перегрузки и способы передачи параметров при реализации пользовательских функций

- Реализовать в среде программирования листинги 1-3.
- С использованием режима отладки исследовать процесс исполнения программ в микропроцессоре. Обратить внимание на способ и место хранения значений локальных переменных **a**, **b** и **c**.
- Сделать выводы по рассмотренным методам передачи параметров в функцию 1-3.

1. Исследование способа передачи параметров в функцию по значению

```
void Swap(int a, int b);  
  
int main(int argc, char* argv[]){  
    int x,y;  
    cout<<"\nEnter two numbers: ";  
    cin>>x>>y;  
    cout<<"\nx="<<x<<", y="<<y<<endl;  
    Swap(x,y);  
    cout<<"\nx="<<x<<", y="<<y<<endl;  
    return 0;  
}  
  
void Swap(int a, int b){  
    int c;  
    c=a;  
    a=b;  
    b=c;  
    return;  
}
```

2. Исследование способа передачи параметров в функцию по адресу

```
void Swap(int *a, int *b);  
int main(){  
    int x,y;  
    cout<<"\nEnter two numbers: ";  
    cin>>x>>y;  
    cout<<"\nx="<<x<<", y="<<y<<endl;  
    Swap(&x,&y);  
    cout<<"\nx="<<x<<", y="<<y<<endl;  
    return 0;  
}  
void Swap(int *a, int *b){  
    int c;  
    c=*a;  
    *a=*b;  
    *b=c;  
    return;  
}
```

3. Исследование способа передачи параметров в функцию по ссылке

```
void Swap(int &a, int &b);  
int main(){  
    int x,y;  
    cout<<"\nEnter two numbers: ";  
    cin>>x>>y;  
    cout<<"\nx="<<x<<", y="<<y;  
    Swap(x,y);  
    cout<<"\nx="<<x<<", y="<<y<<endl;  
    return 0;  
}  
void Swap(int &a, int &b){  
    int c;  
    c=a;  
    a=b;  
    b=c;  
    return;  
}
```

4. Исследование порядка возвращения параметров из функции

- реализовать в среде программирования следующий листинг:

```
int* GetMem();  
int main(){  
    int *a=GetMem();  
    cout<<"\n a is "<<a;  
    cout<<"\n*a is "<<*a;  
    cout<<endl;  
    return 0;  
}  
int* GetMem(){  
    int *c = new int(5);
```

```

        return c;
    }

```

- исследовать процесс исполнения программы; сделать выводы.

5. Использование механизма перегрузки функций

- реализовать в среде программирования следующий листинг:

```

void PrintData(int *s, int N);
void PrintData(char *w);

int main() {
    int *a,n; char str[50];
    cout<<"\nEnter the array size: "; cin>>n;
    a = new int[n];
    for (int i=0;i<n;i++) a[i] = rand()%101-50;
    PrintData(a,n);
    cout<<"\nEnter a string: "; cin>>str;
    PrintData(str);
    return 0;
}

void PrintData(int *s, int N) {
    cout<<endl;
    for (int i=0;i<N;i++) cout<<s[i]<<" ";
    return;
}

void PrintData(char *w) {
    cout<<endl<<w;
    return;
}

```

- сделать выводы об использовании механизма перегрузки функций в данной программе.

Задание II

В соответствии с вариантом составить и реализовать программу; пояснить какой способ передачи параметров в функцию был использован и почему:

1. Даны 4 натуральных числа. Вычислить их наибольший общий делитель, используя подпрограмму для вычисления наибольшего общего делителя двух чисел.
2. Даны координаты вершин двух треугольников. Определить, какой из них имеет большую площадь.
3. Найти наименьшее общее кратное четырех заданных натуральных чисел, используя подпрограмму для вычисления наименьшего общего кратного двух чисел.
4. Два натуральных числа называются "дружественными", если каждое из них равно сумме всех делителей другого, за исключением его самого (например, 284 и 220). Найти все пары "дружественных" чисел, не превосходящих заданного натурального числа.

5. Даны длины сторон треугольника. Вычислить длины медиан треугольника, сторонами которого являются медианы исходного треугольника.
6. Вычислить площадь произвольного выпуклого шестиугольника, заданного координатами вершин.
7. Даны длины сторон треугольника. Вычислить среднее арифметическое длин биссектрис этого треугольника.
8. Вычислить площадь полной поверхности правильной пятиугольной пирамиды, заданной длиной ребра.
9. Составить программу, которая будет вычислять сумму нужного числа n членов данной арифметической прогрессии по любым двум ее членам, номера которых известны. *Примечание:* $a_n = a_1 + d(n-1)$, $s_n = \frac{(a_1 + a_n) \cdot n}{2}$ a_1 – первый член прогрессии; d – разность прогрессии; n – номер взятого члена, s_n – сумма первых n членов арифметической прогрессии.
10. Составить программу, которая будет вычислять сумму нужного числа n членов данной арифметической прогрессии по любому члену прогрессии, номер которого известен, и разности прогрессии. См. примечание (задача 9).
11. Три точки заданы своими координатами на плоскости. Найти площадь и длины диагоналей параллелограмма, вершинами которого являются эти точки.
12. Вычислить периметры трех треугольников и их среднее арифметическое. Если каждый из треугольников задается координатами своих вершин.
13. Найти площадь полной поверхности правильной n - угольной пирамиды по стороне основания при $n = 4$.
14. Сгенерировать члены ряда Фибоначчи, не превосходящие заранее заданного числа n . Проверить, что если число Фибоначчи $f(k > 4)$ - простое, то и k - простое.
15. Вычислить площадь произвольного выпуклого пятиугольника, заданного координатами вершин.
16. Составить программу, которая будет вычислять сумму нужного числа n членов данной геометрической прогрессии по любому члену прогрессии, номер которого известен, и знаменателю прогрессии.
Примечание: $b_n = b_1 q^{n-1}$, $s_n = \frac{b_n \cdot q - b_1}{q - 1}$, где $q \neq 1$, b_1 – первый член прогрессии; q – знаменатель прогрессии; n – номер взятого члена, s_n – сумма первых n членов геометрической прогрессии.
17. Составить программу, которая будет вычислять сумму нужного числа n членов данной геометрической прогрессии по любым двум ее членам, номера которых известны (см. примечание к 16 варианту)
18. Вычислить члены ряда Фибоначчи, не превосходящие заранее заданного числа n . Проверить для нескольких четверок $f_i, f_{i+1}, f_{i+2}, f_{i+3}$ последователь-

ные члены ряда Фибоначчи справедливость соотношения $f_{i+2}^2 - f_{i+1}^2 = f_i \cdot f_{i+3}$.

19. Запрограммировать игру "Угадай число". Программа с помощью генератора случайных чисел выбирает целое число в диапазоне от 0 до 9. Угадать это число за три попытки. После каждой попытки сообщается больше или меньше названное число задуманного.
20. Найти все простые числа в промежутке между натуральными числами a и b ($a > 2000$, $b - a \geq 20$)
21. Дано целое число a . Вычислить $\frac{1.7t(5) + 2t(1+a)}{6 - t(a^2 - 1)}$, где $t(x) = \sum_{k=0}^{10} \frac{x^{2k+1}}{(2x+1)!}$.
22. Сколькими способами можно отобрать команду в составе m человек из n кандидатов? Провести вычисления при $m=5$, $n=8$; при $m=6$, $n=11$.

$$\left(C_n^m = \frac{n!}{m!(n-m)!} \right)$$
23. n интервалов заданы координатами своих концов на координатной прямой и пронумерованы. Задана также точка x на этой прямой. Выяснить, скольким интервалам принадлежит точка x , указать их номера.
24. Даны три пары комплексных чисел, представленных в алгебраической форме. Организовать в виде подпрограммы выполнение четырех арифметических действий над парой комплексных чисел. Составить программу выполнения этих действий для каждой пары заданных чисел.
25. Даны координаты вершин треугольника и координаты двух точек на плоскости. Определить, лежат ли они внутри треугольника.

Контрольные вопросы

1. Перечислите способы передачи параметров в функцию.
2. Особенности передачи параметров в функцию по значению.
3. Особенности передачи параметров в функцию по адресу.
4. Каковы особенности передачи параметров в функцию по ссылке?
5. Может ли функция возвращать массив или функцию в качестве результата своей работы?
6. Перечислите способы вызова функций.
7. Какое служебное слово используют для переименования типов?
8. Что понимают под перегрузкой функций?
9. Какой процесс называют разрешением «уточнением» перегрузки?
10. На чем основан механизм разрешения перегрузки? Дайте пояснения.
11. Перечислите правила описания перегруженных функций.

СПИСОК ЛИТЕРАТУРЫ

1. Конспект лекций.
2. [2] стр. 83-85.

Лабораторная работа № 12

Обработка символьных массивов

ЦЕЛЬ РАБОТЫ: •получение навыков объявления, инициализации, использования символьных массивов при обработке строк на C/C++.

Формируемые компетенции или их части: вырабатывается навык разработки математических и информационных моделей, а также формируется способность к разработке и применению алгоритмических и программных решений в области прикладного программирования (формируется часть компетенций ОПК-3 и ПК-7).

Теоретическая часть

Строки как массив элементов типа char

12.1. Способы представления строк в СИ; реализация некоторых типовых операций над строками

В Си отсутствует специальный тип строк. Строки рассматриваются как массивы символов, оканчивающиеся нулевым символом ('\0'). Строка доступна через указатель на первый символ в строке. Значением строки является адрес ее первого символа. Таким образом, можно сказать, что в Си строка является указателем – указателем на первый символ строки. В этом смысле строки подобны массивам, потому что массив тоже является указателем на свой первый элемент (рисунок 12.1).

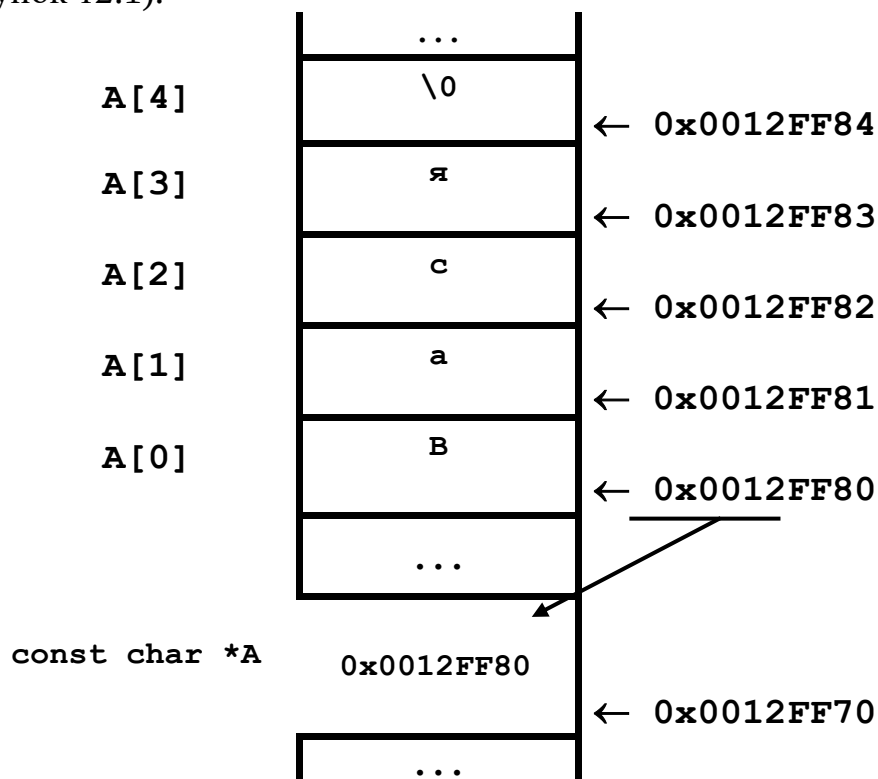


Рисунок 12.1 – Представление массива элементов в памяти ЭВМ

В приведенном примере объявлен символьный массив **A** на 5 элементов:
`char A[]="Вася";`

Обратите внимание на то, что под хранение каждого символа выделяется один байт памяти, а под указатель будет выделено 4 байта. Необходимо отметить, что в приведенном примере размер массива не указан, он определен автоматически по длине строки-инициализатора, причем, автоматически добавлен символ окончания строки.

Строка может быть объявлена либо как массив символов, либо как переменная типа **char***. Каждое из двух приведенных ниже эквивалентных объявлений:

```
char S[] = "строка";  
char *Sp = "строка";
```

присваивает строковой переменной начальное значение "строка". Первое объявление создает массив из 7 элементов S содержащий символы 'с', 'т', 'р', 'о', 'к', 'а' и '\0'. Второе объявление создает переменную-указатель Sp, который указывает на строку с текстом "строка", лежащую где-то в памяти ЭВМ. Но в любом случае число хранимых символов на 1 больше числа значащих символов за счет конечного нулевого символа.

Доступ к отдельным символам строки осуществляется по индексам, начинающимся с нуля. Например, S[0] и Sp[0] - первые символы объявленных выше строк, S[1] и Sp[1] - вторые и т.д.

В приведенных объявлениях длина строк определялась автоматически компилятором. Можно объявлять строковые переменные заданной длины. Например, оператор

```
char buff[100];
```

объявляет переменную buff, которая может содержать строку до 99 значащих символов плюс заключительный нулевой символ.

Теперь приступим к знакомству с наиболее полезным применением символьных указателей: запоминанию массивов строк. На самом деле нельзя создать массив строк, но можно запомнить массив символьных указателей, каждый из которых содержит адрес строки в памяти.

Определением массива символьных указателей создается динамический массив. Этот массив похож на двумерную таблицу, с одним исключением: все строки в этом массиве могут иметь разную длину (разное количество символов в строке).

Например, объявим символьную таблицу с 5 строками и 10 колонками, каждая строка которой содержит одинаковое число символов.

Ее можно определить следующим образом:

```
char names [ 5 ] [ 10 ] = { { "Сергей" },  
                             { "Михаил" },  
                             { "Николай" },  
                             { "Иван" },  
                             { "Алексей" } };
```

Она показана на рисунке 12.2.

Обратите внимание, в таблице много пробелов. Каждая строка занимает 10 символов даже в том случае, когда символов в строке намного меньше. Незаполненные элементы содержат символ ограничитель строки, потому что Си записы-

вает этот символ во все неинициализированные элементы массива. Этот тип таблиц занимает слишком много места в памяти.

		Колонки										
		0	1	2	3	4	5	6	7	8	9	10
Строки	0	С	е	р	г	е	й	\0				
	1	М	и	х	а	и	л	\0				
	2	Н	и	к	о	л	а	й	\0			
	3	И	в	а	н	\0						
	4	А	л	е	к	с	е	й	\0			

Рисунок 12.2 – Пример двумерного символьного массива строк

Для устранения проблемы нерационального использования памяти объявим одномерный массив символьных указателей. Каждый указатель содержит адрес строки в памяти, и эти строки могут иметь разную длину.

Вот примеры определения (инициализации) таких массивов:

<pre>char *names [5] = { { "Сергей" }, { "Михаил" }, { "Николай" }, { "Иван" }, { "Алексей" } };</pre>	или просто	<pre>char *names [] = { { "Сергей" }, { "Михаил" }, { "Николай" }, { "Иван" }, { "Алексей" } };</pre>
--	------------	--

Это одномерные массивы. Звездочка перед `names` говорит о том, что это массив указателей. Указатели имеют символьный тип. Строки не присваиваются элементам массива, но в массиве содержатся адреса этих строк. На рисунке 12.3 показан этот массив указателей. Строки могут находиться в любом месте памяти. Их местоположение не существенно, так как каждый указатель содержит адрес первого элемента соответствующей строки. Строки не занимают лишнюю память; каждая строка использует ровно столько пространства, сколько нужно для размещения ее и символа ограничителя строки. Такая структура содержит данные в форме с рваными краями.

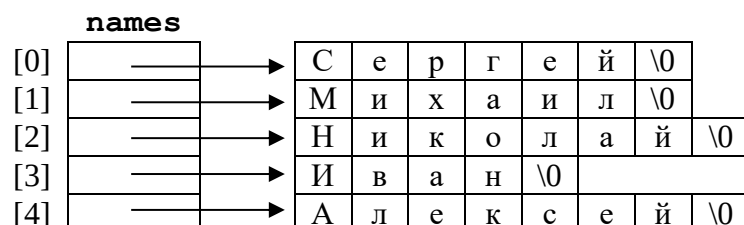


Рисунок 12.3 – Массив указателей

Для вывода первой строки можно использовать оператор `cout`:

```
cout « *names;    // Печать Сергей
```

Для вывода второй строки используется следующий оператор `cout`:

```
cout « *(names+1);    // Печать Михаил
```

При обращении к элементу массива при помощи оператора `*`, получаем доступ к одной из строк массива. Можно использовать возвращаемое этим оператором значение везде, где можно использовать строковую константу или символьный массив.

Перечислим некоторые основные операции, предусмотренные для строки встроенного типа (символьного массива):

- инициализация массивом символов (строкой встроенного типа);
- ввод/вывод;
- доступ к отдельным символам строки для чтения и записи: для этого применяется операция взятия индекса или косвенная адресация.

12.2. Основные функции Си для работы со строками

Очень часто при работе с символьными массивами возникает задача их обработки (сравнение строк, объединение строк, поиск подстроки в строке и так далее). В заголовочном файле `<string.h>` языка Си описано множество функций для работы со строками.

Например, функции:

- копирования строк (**`strcpy`**, **`strncpy`**);
- сравнения строк (**`strcmp`**, **`strncmp`**);
- объединения строк (**`strcat`**, **`strncat`**);
- поиска подстроки (**`strstr`**);
- поиска вхождения символа (**`strchr`**, **`strrchr`**, **`strpbrk`**);
- определения длины строки (**`strlen`**);
- преобразования строки в число (**`atoi`**, **`atoll`**, **`atof`**);

и так далее.

Учитывая, что строки состоят из символов, полезно знать некоторые функции работы с символами. Для работы с символами служит библиотека **`cctype`** (заголовочный файл **`cctype.h`**). Она содержит:

1. Функции проверки на принадлежность символа множеству:

- **`isalnum(c)`** – букв и цифр (A-Z, a-z, 0-9);
- **`isalpha(c)`** – букв (A-Z, a-z);
- **`iscntrl(c)`** – управляющих символов (с кодами 0...31 и 127);
- **`isdigit(c)`** – цифр (0-9);
- **`isgraph(c)`** – печатаемых символов кроме пробела;
- **`islower(c)`** – букв нижнего регистра (a-z);
- **`isprint(c)`** – печатаемых символов;
- **`ispunct(c)`** – знаков пунктуации;
- **`isspace(c)`** – символов-разделителей;
- **`isupper(c)`** – букв верхнего регистра (A-Z);

- `isxdigit(c)` – шестнадцатеричных цифр.
2. Функции преобразования символов:
- `tolower(c)` – к нижнему регистру;
 - `toupper(c)` – к верхнему регистру.

Практическая часть

ВЫПОЛНЕНИЕ РАБОТЫ: освоить теоретический материал, выполнить общее для всех задание I и в соответствии с вариантом составить программу (задание II).

Задание I

1. Изучение способов формирования строк в языке Си

- Набрать и отладить программу:


```
int main(int argc, char* argv[]){
    char Arr1[25];
    char Arr2[]={ "Second string!"};
    char Arr3[]={ 'T','h','i','r','d',' ','s','t','r','i','n','g' };
    char *Arr4="Fouth string";
    cout<<"\nEnter string: \"First string\": ";
    cin>>Arr1;
    cout<<"\nString 1: "<<Arr1; // Оператор 1
    cout<<"\nString 2: "<<Arr2; // Оператор 2
    cout<<"\nString 3: "<<Arr3; // Оператор 3
    cout<<"\nString 4: "<<Arr4; // Оператор 4
    return 0;
}
```
- Проанализировать результат работы программы и ответить на вопрос, почему **Оператор 2** и **Оператор 4** выводят на экран свои фразы полностью, а оператор 1 – только ее часть.
- Почему **Оператор 3** выводит свою фразу своей строки и чужую.
- Что нужно сделать, чтобы **Оператор 3** выводил только свою фразу.

2. Применение функций работы со строками для обработки символьных массивов

- Набрать и отладить программу заполнения числового массива из символьной строки:

```
int main(int argc, char* argv[]) {
    char str[]="2, 75.3, 24.45, 55, 0, 0.37";
    char *ptr=str;                               //Оператор 1
    float Arr[10];
    int i=0;
    printf("\nИсходная строка:\n%s\n",str);
    do {
```

```

        Arr[i] = atof(ptr);                //Оператор 2
        i++;
        if (i>9) break;
    }
    while(ptr==strchr(ptr,','),ptr++);    //Оператор 3
    printf("\nМассив чисел: \n")
    for (int s=0;s<i;s++)
        printf("%5.2f\n", Arr[s]);
    return 0;
}

```

- Проанализировать результат работы программы и ответить на вопрос: что происходит при выполнении **Операторов 1-3**.

Задание II

1. В заданном тексте вставить между словами вместо пробелов запятую и пробел.
2. Задан текст. Вставить заданную последовательность символов после i -го символа каждого слова.
3. Вывести на экран в один столбец слова из заданного текста оканчивающиеся на согласную букву, в другой столбец – слова, оканчивающиеся на гласную букву, в третий - все остальные слова.
4. Из заданного текста удалить его часть, заключенную в круглые скобки вместе со скобками.
5. Каждое слово текста преобразовать так, чтобы оно читалось наоборот.
6. Из заданной последовательности слов, разделенных запятыми, удалить слова, содержащие введенный с клавиатуры символ.
7. Составить программу подсчета числа слов в телеграмме и выдачи на печать квитанции об ее уплате.
8. Дан текст из n символов. Вывести на экран только строчные русские буквы, входящие в этот текст.
9. Дан текст из строчных латинских букв, за которым следует точка. Вывести на экран дисплея этот текст заглавными латинскими буквами.
10. После каждой гласной буквы x в последовательности слов, разделенных пробелами, вставить буквосочетание $pxsx$ (например, слово *порядок* преобразуется в *поросорярядоросок*).
11. Задан текст. Найти сколько удвоенных букв содержится в данном тексте. Заменить каждую пару символом *.
12. Определить, не используя стандартную процедуру, является ли введенная строка правильной записью целого десятичного числа без знака.
13. В тексте слова разделены пробелами. Определить, какой процент слов в тексте заканчивается на русскую букву *а*.
14. Задан текст на русском языке. Удалить из него пробелы, ь, ъ.

15. В тексте слова разделены пробелами. Определить, какой процент слов в тексте заканчивается на русскую букву *а*.
16. Даны две символьные строки, состоящие только из цифр (длина каждой более 10 символов). Считая, что в этих строках находятся очень длинные целые числа, сформировать третью строку – сумму этих чисел.
17. В заданном тексте найти и вывести на экран удвоенные буквы. Посчитать количество удвоенных гласных букв и количество удвоенных согласных.
18. Дана квадратная матрица, элементами которой являются целые положительные числа. Найти корень квадратный из числа, образованного цифрами чисел, стоящих на главной диагонали.
19. Дан произвольный текст. Отредактировать текст так, чтобы между словами был ровно один пробел, а между предложениям ровно два пробела.
20. Исключить из введенной строки символ, введенный с клавиатуры. Вывести на экран количество удаленных символов и полученную строку.
21. В тексте слова разделены пробелами. Определить, какой процент слов в тексте начинается на русскую букву *м*.
22. Вывести на экран самое длинное слово из заданного текста.
23. Текст задан следующим образом: первый символ – цифра указывающая длину первого слова, за первым словом - вновь цифра указывающая длину второго слова (длина каждого слова не более 9 символов) и т.д. Выписать *k*-ое слово из текста (число *k* вводится с клавиатуры).
24. Назовем сложностью предложения сумму количества слов и знаков препинания. Определить сложность заданного предложения.
25. В заданной строке заменить все вхождения одной подстроки на другую. Обе подстроки вводятся с клавиатуры.
26. Выбрать из текста все слова, которые имеют в своем составе по два и более одинаковых символа.
27. Дана квадратная целочисленная матрица, элементами которой являются положительные числа. Найти корень квадратный из числа, образованного цифрами чисел, стоящих на побочной диагонали.

Контрольные вопросы

1. Что представляет из себя строка в программе, написанной на ЯП Си?
2. Что представляет собой строка:
 - а) последовательность английских букв;
 - б) последовательность русских букв;
 - в) последовательность символов ПЭВМ?
3. Как может быть описана строка в Си?
4. Приведите пример объявления строки как массива символов.
5. Приведите пример объявления строки как переменной-указателя на строку.
6. Что является признаком окончания строки?
7. Как осуществляется доступ к элементам строки?

8. Какое максимальное количество символов может содержать строка?
9. Как при описании строки может быть задана ее длина?
10. Может ли в процессе выполнения программы фактическая длина строки стать больше, чем объявлено в разделе описания? Что произойдет в этом случае?
11. Перечислите типовые операции над строками.
12. Перечислите основные функции обработки строк.

Пример выполнения задания II лабораторной работы

Задание. Дано предложение, между словами которого стоит один пробел, в конце предложения – **точка**. Подсчитать количество слов в этом предложении и вывести на экран в столбец те из них, которые начинаются с введенной с клавиатуры буквы или символа, и их количество. Если таких слов во введенном предложении нет, вывести сообщение об этом.

1. Решение (математическая модель). Предложение будем вводить в символьный массив `text[100]`. Разобьем это предложение на отдельные слова и будем хранить их в виде элементов двумерного символьного массива `mas`. Вводимый с клавиатуры символ обозначим через `x`, количество слов в данном предложении – через `m` (`int`), количество слов, начинающихся с введенного символа (результат работы программы) – через `kol_x` (`int`). Формирование массива `mas` выполним с помощью цикла с постусловием до появления точки. Каждое слово будет получаться в цикле при посимвольной обработке одномерного массива `text`, пока не встретится пробел, увеличивая после каждого пробела счетчик слов на единицу.

2. Программа решения данной задачи может иметь следующий вид:

```
#include <conio.h> // для корректной работы с кириллицей необходимо
#include <iostream> // сохранить файл проекта в кодировке DOS 866 файл→
#include <cstdio> // дополнительные параметры сохранения→кириллица (DOS)
using namespace std;

int main() {
    char ch, x, mas[25][100], text[100]; // mas - двумерный массив слов
    cout<<"Введите предложение, оканчивающиеся точкой "<<endl;
    gets(text); // организуем ввод всех символов текста
    int i=0; // № символа слова
    int j=0, m=0; // рассматриваем первое слово
    do {
        ch=text[j];
        if ((ch!=' ') && (ch!=',' ) && (ch!='.')) {
            mas[m][i]=ch; //формируем текущее слово
            i++;
        }
        else
            if(i!=0) {
                mas[m][i]='\0';
                m++; //увеличиваем счетчик слов
                i=0;
            }
        j++;
    }
```

```

    }
    while (ch!='.');
```

cout<<"Количество слов в данном предложении: "<<m<<endl;

cout<<"Введите символ, с которого должны начинаться выводимые на экран слова: ";

cin>>x;

int kol_x=0; // обнуление счетчика искомых слов

cout<<"Слова, начинающиеся с символа '"<<x<<"':"<<endl;

for (i=0; i<m; i++)

 if (mas[i][0]==x)

 {// если введенный символ в текущем слове на первой позиции, то

 cout<<mas[i]<< " ";

 kol_x++;

 }

if (kol_x==0) cout<<"отсутствуют";

else {

 cout<<"Всего в предложении "<<kol_x;

 switch (kol_x)

 {//организация вывода верного окончания

 case 1: case 21:cout<<" слово, ";break;

 case 2: case 3: case 4: case 22: case 23: case 24:

cout<<" слова, ";break;

 default:cout<<" слов, ";

 }

 cout<<"начинающихся на '"<<x<<"'.";

 }

 }

_getch();

return 0;

}

3. Результат работы программы:

1)

Введите предложение

Некоторые люди видят в луже лишь лужу, а некоторые видят в луже звезды.
Количество слов в данном предложении: 13

Введите символ, с которого должны начинаться выводимые на экран слова:
л

Слова, начинающиеся на символ "л"

Люди луже лишь лужу луже

Всего в предложении 5 слов, начинающихся на 'л'.

2)

Введите предложение

А солнце светило ярче.

Количество слов в данном предложении: 4

Введите символ, с которого должны начинаться выводимые на экран слова:
л

Слова, начинающиеся на символ 'л'

Отсутствуют

СПИСОК ЛИТЕРАТУРЫ

1. Конспект лекций.
2. [2] стр. 63-65.

Лабораторная работа № 13

Программирование задач на обработку данных строкового типа

ЦЕЛЬ РАБОТЫ: получение навыков объявления, инициализации и использования объектов класса `string` C++.

Формируемые компетенции или их части: вырабатывается навык разработки математических и информационных моделей, а также формируется способность к разработке и применению алгоритмических и программных решений в области прикладного программирования (формируется часть компетенций ОПК-3 и ПК-7).

Теоретическая часть

Строки как объект специального класса `string`

13.1. Описание строк в C++ и типовые операции над ними

Для использования в программе возможностей класса `string` необходимо в начале программы подключить заголовочный файл `string`:

```
#include <string>
```

В отличие от типа `char`, `string` является классом. Классы, грубо говоря, содержат в себе переменные, константы и функции (методы) для работы с переменными. `string` использует единственный объект `string` для организации работы со строками. Будучи частью стандартной библиотеки C++, этот объект также является частью стандартного пространства имён – `std`. Таким образом, для использования возможностей заголовочного файла `string` необходимо в зоне описания глобальных объектов программы описать пространство стандартных имён `std`.

Теперь для создания строки достаточно написать:

```
string s;
```

Для записи в строку можно использовать оператор присваивания (=):

```
s = "Hello";
```

Пример работы (ввод/вывод) с объектом класса `string`:

```
string name;
cout<<"Enter your name"<<endl;
cin>>name; // строка считается до первого пробела
//поэтому при работе со string лучше использовать getline(cin, name);
//или cin.getline(name[, 50]); в [] указан необязательный параметр
cout<<"Hi "<< name <<"!"<<endl;
```

Рассмотрим некоторые основные операции, реализованные для объекта класса `string`:

- инициализация массивом символов (строкой встроенного типа) или другим объектом типа `string`;

```
string st( "Для работы с кириллицей Dos 866\n" );// или
string s0=st; // или
string s1 = "файл→дополнительные параметры сохранения→кириллица (DOS)";
```

- копирование одной строки в другую;
`st2 = st3;` // С помощью операции присваивания копируем st3 в st2
- доступ к отдельным символам строки для чтения и записи;
- сравнение двух строк на равенство;
`if ( st == st3 )`
// инициализация сработала
- конкатенация двух строк, получая результат либо как третью строку, либо вместо одной из исходных;
`string s1( "hello, " );`
`string s2( "world\n" );`
`string s3 = s1 + s2;` // 3-я строка – результат конкатенации двух первых
`s1 += s2;` // или же возможно добавление s2 в конец s1

Можно переписать пример, приведенный выше, так, чтобы специальные символы и знаки препинания представлялись встроенным типом, а значимые слова – объектами класса `string`:

```
const char *pc = ", "; // Операция сложения может конкатенировать
string s1( "hello" ); //      объекты класса string
string s2( "world" ); //      и со строками встроенного типа
string s3 = s1 + pc + s2 + "\n";
```

Подобные выражения работают потому, что компилятор знает, как автоматически преобразовывать объекты встроенного типа в объекты класса `string`. Возможно и простое присваивание встроенной строки объекту `string`:

```
string s1;
const char *pc = "a character array";
s1 = pc; // правильно
```

13.2. Основные функции класса `string` для работы со строками

Рассмотрим некоторые основные методы (функции), предусмотренные для работы со строками, являющимися экземплярами класса `string`:

- `s.append(str)` – добавляет в конец строки строку `str`. Можно писать как `s.append(переменная)`, так и `s.append("строка")`;
- `s.assign(str)` – присваивает строке `s` значение строки `str`. Аналогично записи `s=str`;
- `int i=s.begin()` – записывает в `i` индекс первого элемента строки;
- `int i=s.end()` – аналогично, но последнего;
- `s.clear()` – как следует из названия, очищает строку, т.е. удаляет все элементы в ней;
- `s.compare(str)` – сравнивает строку `s` со строкой `str` и возвращает 0 в случае совпадения (на самом деле сравнивает коды символов и возвращает их разность);
- `s.copy(str, len (сколько), start (начиная с какого))` – копирует из строки `s` в строку `str` (`str` может быть как типа `string`, так и строка типа `char`). Последние два параметра не обязательные (можно использовать функцию с 1, 2 или 3 параметрами);

- **bool** b=s.empty() – если строка пуста, возвращает **true**, иначе **false**;
 - s.erase(start, len) удаляет len элементов с заданной позиции start;
 - s.find(str, start) – ищет строку str начиная с заданной позиции start;
 - s.insert(start, str, beg, count) – вставляет в строку s, начиная с позиции start часть строки str, начиная с позиции beg count символов;
 - s.replace(start, n, str) – берет n первых символов из str и заменяет символы строки s на них, начиная с позиции start;
- ```
replace(str.begin(), str.end(), '.', '_');//пробегает диапазон от
// begin() до end() и заменяет элементы, равные третьему параметру,
// на четвертый
```
- **int** len=s.length() – записывает в len длину строки;
  - s.push\_back(symbol) – добавляет в конец строки символ;
  - str=s.substr(n, m) – возвращает m символов начиная с позиции n;
  - s.swap(str) – меняет содержимое s и str местами;
  - s.size() – возвращает число элементов в строке.

Пример исследования особенностей ввода/вывода строковых данных.

```
...
string s, s1, ss;
cout<<"\nEnter string: \"1 string\": "<<endl;
cin>>s;
getline(cin,s1);
cout<<"\nEnter string: \"2 string\": "<<endl;
getline(cin,ss);
cout<<"\nString 1: "<<s;
cout<<"\nпродолжение String 1: "<<s1;
cout<<"\nString 2: "<<ss<<endl;
string str = "Hello, world!";
cout << str <<endl;
string name,surname,text,fullname,s2,s3,user;
user = "Petya Petrov";
cout << user <<endl;
cout<<"Enter your name"<<endl;
getline(cin,name);
cout<<"Enter your surname"<<endl;
getline(cin,surname);
fullname=name;
fullname+=" "; // добавляем пробел
fullname.append(surname);
cout <<"fullname = "<<fullname <<endl;
if (fullname.compare(user)==0) // <=> if (!(fullname.compare(user)))
 cout<<"Your are good user"<<endl;
 else cout<<"Bad user"<<endl;
cout<<"enter s1"<<endl;
getline(cin,s1);
cout<<"enter s2"<<endl;
getline(cin,s2);
s1.swap(s2);
cout<<"new s1: "<<s1<<endl<<"new s2: "<<s2<<endl;
cout<<"Enter big text with your name"<<endl;
getline(cin,text);
```

```

cout <<"old text = "<<text <<endl;
int i=text.find(name);
while (i)// <=> while (i!=0) {
 text.replace(0,name.length(),name); //замена первых сим_в text на name
 s3=text.substr(i,name.length()); // помещаем подстроку из text в s3
 cout<<"Replaced: "<<s3<<endl;
 i=text.find(name); //ищем подстроку name в строке text
}
cout<<"New text:"<<endl<<text<<endl;
text.clear(); //очищаем строку
cout<<"text: "<<text<<endl;
...

```

## Практическая часть

**ВЫПОЛНЕНИЕ РАБОТЫ:** освоить теоретический материал, в соответствии с вариантом решить задачу, составить и реализовать программу с использованием функций класса string.

### Задание

1. Ввести предложение, слова в котором разделены пробелам и запятыми. Распечатать это предложение, удалив из него те, которые встретились там более одного раза.
2. Найти множество всех слов, которые встречаются в каждом из двух заданных предложений.
3. Удалить из данной строки те слова, которые уже встречались в ней раньше.
4. Ввести строку, состоящую только из цифр и букв. Распечатать те группы цифр, в которых цифра 7 встречается не более двух раз (группа цифр - это последовательность цифр, обрамленная буквами).
5. Ввести предложение, слова в котором разделены пробелам и запятыми. Распечатать те слова, которые являются обращениями других слов в этом предложении. Если нужных слов нет, сообщить об этом.
6. Распечатать введенное предложение, удалив из него слова, которые состоят менее чем из трех букв.
7. Распечатать введенные слова, отличные от последнего перестроив их следующим образом: перенести последнюю букву каждого слова в его начало и в каждом слове оставить только первые вхождения каждой буквы.
8. Распечатать строку, которая получается из введенной строки следующим образом: каждая цифра заменяется заключенной в круглые скобки последовательностью литер «+», если цифра четная, или «-», если цифра нечетная. Длина последовательности равна числу, изображаемому цифрой.
9. Ученики зашифровывают свои записки, записывая все слова наоборот. Составить программу, расшифровывающую сообщения учеников.

10. Распечатать введенную строку, удалив из нее слова с нечетными номерами и перевернув слова с четными номерами. Например, из строки *'холодный ручей обжигал лицо и руки'* должно получиться *'печур оцил икур'*.
11. Ввести строку, в которую могут входить только цифры и буквы. Распечатать те группы букв, в которых буква *A* встречается не менее двух раз (группа букв - это последовательность букв, обрамленная цифрами).
12. Ввести произвольный текст из нескольких предложений. Вычислить среднее число слов в предложении и среднюю длину предложения.
13. Дан текст, в котором слова разделены пробелами. Создать список слов текста с указанием количества букв в них.
14. Распечатать, начиная с последнего слова, перевернутые слова введенного предложения.
15. Выделить из первой строки все слова, начинающиеся с гласной буквы, а из второй строки - слова, начинающиеся с согласной. Образовать третью строку, состоящую из выделенных слов обеих строк. В новой строке слова должны разделяться двумя пробелами.
16. Из введенного текста распечатать только те слова, которые симметричны (например, *радар, анна, шалаш*) либо в которых буквы расположены в порядке, обратном алфавитному (например, *ток, три, срок*).
17. Ученики зашифровывают свои записки, записывая каждое слово текста наоборот. Составить программу, позволяющую шифровать сообщения описанным способом.
18. Для каждого из слов заданного предложения указать, сколько раз оно встречается в предложении.
19. Заменить окончание *ing* каждого слова, встречающегося в заданном предложении на английском языке, на *ed*.
20. Дана последовательность слов, разделенных пробелами. Напечатать все слова, предварительно удалив из каждого слова все предыдущие вхождения последней буквы.
21. Из заданной последовательности слов, разделенных запятыми, удалить слова, содержащие введенную с клавиатуры последовательность символов.
22. Дана последовательность слов, разделенных пробелами. Напечатать все слова, предварительно удалив из каждого слова все вхождения последней буквы предыдущего слова.
23. Вывести на экран те слова из введенного текста, в которых нет повторяющихся букв.
24. Задан текст, состоящий из нескольких предложений. Для каждого предложения вывести на экран все слова, отличные от последнего слова текущего предложения.
25. Дана последовательность слов, каждое из которых содержит от 2 до 10 латинских букв. Между соседними словами - пробел, за последним словом - точка.

Если слово имеет нечетную длину, то удалить среднюю букву, если четную - перенести первую букву в конец слова. Вывести на экран все преобразованные слова.

26. В заданном тексте слова разделены пробелами или запятыми. Вывести на экран список слов, имеющих в своем составе хотя бы один символ, введенный с клавиатуры.

27. Вывести на экран дисплея список слов, имеющих приставку.

### **Задания для самостоятельной работы**

Следующая группа задач является не обязательной для решения и относится к области криптографии, т.е. к шифровке и расшифровке текстов. Решение задач из этой группы улучшит навыки обработки строковых величин и расширит знания по их применению на практике.

1. Зашифровать текст методом Гронсфельда. Ключом является конечная последовательность цифр, которую записывают подряд над символами шифруемого текста. Цифра, стоящая над литерой, является величиной смещения (т.е. говорит о том, на сколько позиций надо продвинуться вперед по таблице кодировки от текущего символа, чтобы получить для него замену). Протестировать написанную программу.

2. Чтобы зашифровать текст, записанный с помощью русских букв и знаков препинания, его можно переписать, заменив каждую букву непосредственно следующей за ней по алфавиту (буква *я* заменяется буквой *а*). Можно обобщить этот способ шифровки - сдвиг производится не на одну букву, а на  $n$  букв, где  $n$  - данное натуральное число (можно представить себе, что буквы выписаны по кругу, как цифры на циферблате часов). Написать и протестировать программу, которая зашифровывает и расшифровывает данный текст.

3. Пусть по некоторому каналу связи передается сообщение, имеющее вид последовательности нулей и единиц (или аналогично точек и тире). Из-за помех возможен ошибочный прием некоторых сигналов: нуль может быть воспринят как единица и наоборот. Можно передавать каждый сигнал трижды, заменяя, например, последовательность 1, 0, 1 последовательностью 1, 1, 1, 0, 0, 0, 1, 1, 1. Три последовательные цифры при расшифровке заменяются той цифрой, которая встречается среди них по крайней мере дважды. Такое утаивание сигналов повышает вероятность правильного приема сообщения. Написать программу расшифровки некоторого сообщения.

4. Следующий способ предназначен для шифровки последовательностей нулей и единиц (или же, например, точек и тире). Пусть  $a_1, a_2, \dots, a_n$  - такая последовательность. То, что предлагается в качестве ее шифра, - это последовательность  $b_1, b_2, \dots, b_n$ , образованная по следующему правилу:  $b_1 = a_1$ ;  $b_i = 1$ , если  $a_i = a_{i-1}$  и  $b_i = 0$  в противном случае ( $i = 1, 2, \dots, n$ ).

Пользуясь изложенным способом, зашифровать и расшифровать данную последовательность нулей и единиц (или точек и тире).

5. Чтобы зашифровать текст, состоящий из 121 буквы, его можно записать в квадратную матрицу порядка 11 по строкам, а затем прочитать по спирали, начиная с центра (т.е. с элемента, имеющего индексы 6, 6). Зашифровать и расшифровать данный текст этим способом. Выбрав некоторую таблицу, разработать способ табличной замены каждого символа другим символом - его шифром. Зашифровать и расшифровать данный текст указанным способом.

### Контрольные вопросы

1. Как описывается переменная-строка, являющаяся экземпляром класса `string`?
2. Как осуществляется доступ к элементам строки?
3. Перечислите типовые операции над строками.
4. Как выполняется инициализация строк?
5. Как выполняется ввод/вывод строк, являющихся объектом класса `string`?
6. Как выполняется сравнение строк?
7. Как выполняется конкатенация строк?
8. Перечислите основные функции обработки строк.

### Пример выполнения лабораторной работы

**Задание.** Найти сумму цифр введенного натурального числа.

**1. Решение (математическая модель+словесный алгоритм).** Натуральное число будем считывать в строковую переменную `stroka`. Обнулим сумматор `s=0`. Вычислим длину строки `n`, далее в цикле с параметром:

- каждый символ строки будем преобразовывать в число, получая тем самым цифру введенного числа;
- наращивать сумматор на величину полученной цифры числа.

После окончания цикла выведем результат – сумму цифр числа `s`.

**Аргументы:** `stroka (string)` – целое число, введенное с клавиатуры.

**Результаты:** `s (int)` – Сумма цифр числа

**Промежуточные величины:** счетчик `i`, количество элементов в строке `n`, численное представление символа введенной строки `si` – целого типа.

**2. Программа решения данной задачи может иметь следующий вид:**

```
#include <conio.h>
#include <iostream>
#include <string>
using namespace std;
int main()
{
 string stroka;
 cout<<"Введите натуральное число:"<<endl;
 getline(cin,stroka);
 int si, s=0, n=stroka.length();
 for (int i=0; i<=n-1;i++)
 {
```

```

 si=stroka[i]-'0';
 s=s+si;
 }
 cout<<"Сумма цифр числа: "<<s<<endl;
 _getch();
 return 0;
}

```

### 3. Результат работы программы:

Введите натуральное число:  
 12345678  
 Сумма цифр числа: 36

### СПИСОК ЛИТЕРАТУРЫ

1. Конспект лекций.
2. [2] стр. 159-165.

## Лабораторная работа № 14

### Применение структур для решения прикладных задач

**ЦЕЛЬ РАБОТЫ:** получение начальных навыков в объявлении, инициализации и использовании производных типов данных, совершенствование навыков описания и использования функций в программах.

**Формируемые компетенции или их части:** вырабатывается способность к разработке алгоритмических и программных решений в области системного и прикладного программирования, математических, информационных и имитационных моделей, прикладных баз данных, тестов и средств тестирования систем и средств на соответствие стандартам и исходным требованиям (формируется часть компетенций ОПК-3 и ПК-7).

### Теоретическая часть

#### Структуры

#### **14.1. Порядок объявления и инициализации структур**

Структуры в Си применяют для логического объединения связанных между собой данных. В структуру, в отличие от массива, можно объединять данные разных типов.

В языке C++ структура является видом класса и обладает всеми его свойствами, но на данном этапе изучения языка C++ достаточно использовать структуры так, как они определены в языке Си:

```
struct [имя_типа]
{
 тип_1 элемент_1;
 тип_2 элемент_2;
 . . .
 тип_n элемент_n;
} [список описателей];
```

Элементы структуры называются *полями структуры* и могут иметь любой тип, кроме типа этой же структуры, но могут быть указателями на него. Если отсутствует имя типа, должен быть указан список описателей: переменных, указателей или массивов. В этом случае описание структуры служит определением элементов этого списка:

```
struct // Пример структуры, описывающей анкетные данные студентов
{
 char fam[20];
 char name[15];
 int age;
} stud[30], *ptr;
```

В данном примере описывается структура, содержащая три поля: символьный массив на 20 элементов с именем `fam`, символьный массив на 15 элементов с именем `name` и целочисленная переменная с именем `age`, которые предназначены для хранения информации о фамилии, имени и возрасте студента соответственно. В списке описателей задан массив из 30 экземпляров этой структуры и указатель

на структуру (который тоже задает экземпляр структуры, но идентифицируется не по имени экземпляра, а по имени указателя, хранящего адрес начала памяти, выделенной на экземпляр структуры, но без имени экземпляра). Если список описателей отсутствует, описание структуры определяет новый тип, имя которого можно использовать в дальнейшем наряду со стандартными типами, например:

```
struct student
{
 char fam[20];
 char name[15];
 int age;
};
int main()
{ . . .
 student stud[30], *ptr;
 . . .
 return 0;
}
```

В данном примере описана глобальная структура данных типа структуры с заданием нового типа данных `student`. Задание же экземпляров этой структуры производится в рамках функции `main()`.

Переменные структурного типа можно размещать и в динамической области памяти, для этого надо описать указатель на структуру и выделить под нее место:

```
student *ptr=new student; // экземпляр структуры
student *stud=new student[N]; // массив экземпляров
// структур
```

Для инициализации структуры значениями используются те же методы, что и для обычных типов данных, но с небольшим дополнением: для доступа к полю структуры, экземпляр которой создан как обычная переменная или массив – через операцию «точка», а для обращения к полю структуры, экземпляр которой создан как указатель на структуру – через операцию «стрелка»:

```
stud[1].fam="Сидоров"; // Доступ по имени экземпляра
stud[1].name="Александр";
stud[1].age=23;
prt->fam="Петров"; // Доступ через указатель
ptr->name="Семен"; // на структуру
ptr->age=21;
```

На практике ввод значений полей структуры, описанной массивом экземпляров, осуществляется с использованием циклов с параметром:

```
for (int i=0; i<N; i++)
{
 cout<<"\nВведите фамилию студента: ";
 cin >>stud[i].fam;
 cout<<"\nВведите имя студента: ";
 cin >>stud[i].name;
 cout<<"\nВведите возраст студента: ";
 cin >>stud[i].age;
}
```

```
}
```

В данном примере количество студентов N должно быть определено (введено с клавиатуры) выше по тексту программы, до момента использования.

## **14.2. Программирование с использованием структур**

Для переменных одного и того же структурного типа определена **операция присваивания**, при этом происходит поэлементное копирование.

Сами по себе структуры являются лишь средством описания сложных типов данных, но основное их достоинство – удобство в поиске и анализе описанных в таком виде данных. Например, необходимо вывести на экран данные о всех студентах, чей возраст на данный момент не превышает 22 лет:

```
k=0;
for (j=0; j<N; j++)
 if (stud[j].age<=22)
 {
 cout<<"\nФамилия: "<<stud[j].fam;
 cout<<"\tИмя: "<<stud[j].name;
 cout<<"\tВозраст: "<<stud[j].age;
 k++;
 }
if (k==0) cout<<"\nТаких студентов нет!";
```

или о студентах, чья фамилия **“Петров”**:

```
k=0; // f=true;
for (j=0; j<N; j++)
 if (strcmp(stud[j].fam, "Петров")==0)
 {
 cout<<"\nФамилия: "<<stud[j].fam;
 cout<<"\tИмя: "<<stud[j].name;
 cout<<"\tВозраст: "<<stud[j].age;
 k++; // f=false;
 }
if (k==0) cout<<"\nТаких студентов нет!";
// if (f) cout<<"\nТаких студентов нет!";
```

Предложенная в комментариях альтернатива bool f для рассмотренного в примере int k является более корректной реализацией требуемой операции.

## **14.3. Использование функций для работы с производными типами данных**

Структуру можно передавать в функцию и возвращать в качестве значения функции.

В соответствии с модульным подходом к программированию целесообразно реализовывать вопросы поиска в отдельных функциях, например:

```
void ZaprosFamily(student *k, char *f);

int main()
{ . . .
 char *family;
```

```

. . .
cout<<"\nВведите фамилию для поиска: ";
cin >>family;
ZaprosFamily(kurs, family);
. . .
return 0;
}

void ZaprosFamily(student *k, char *f)
{
 k=0;
 for (j=0; j<N; j++)
 if (strcmp(k[j].fam, f)==0)
 {
 cout<<"\nФамилия: "<<k[j].fam;
 cout<<"\tИмя: "<<k[j].name;
 cout<<"\tВозраст: "<<k[j].age;
 k++;
 }
 if (k==0) cout<<"\nТаких студентов нет!";
 return;
}

```

В этом примере в функцию передаются два указателя: один типа `student`, другой типа `char`. Что же это за указатели, почему они передаются и что значат?

Первый указатель есть указатель на тип данных `student`, то есть на структуру. Почему передается указатель? Потому, что экземпляры структуры описаны в виде массива экземпляров, а, известно, что имя массива есть указатель на нулевой элемент массива. Таким образом, мы передаем в функцию через указатель весь массив экземпляров структуры.

Второй указатель есть указатель на тип данных `char`, то есть на символьный тип данных. Через этот указатель в функцию передается строка для поиска интересующей нас фамилии студента.

В результате работы функции ничего не возвращается (тип возвращаемого значения `void`), но нам ничего и не надо возвращать, так как функция и так выводит на экран список интересующих нас студентов.

## Практическая часть

**ВЫПОЛНЕНИЕ РАБОТЫ:** в соответствии с вариантом составить и реализовать программы.

### Задание I

Используя структурированный тип – `struct` (структуру), составить программу ввода с клавиатуры списка объектов (не менее 10) на тему, заданную в соответствующем варианте задания, 4-6 характеристик этих объектов и вывода данных на экран в виде прямоугольной таблицы. Организовать поиск данных по одной из заданных характеристик.

Информация должна быть достоверной. Ее можно почерпнуть из справочников, энциклопедий, реклам, книг и т.п.

- |                               |                                            |
|-------------------------------|--------------------------------------------|
| 1. Знаки зодиака              | 14. Материалы для ремонта квартиры         |
| 2. Промышленные товары        | 15. Вузы России                            |
| 3. Продукты питания           | 16. Сведения о студентах группы            |
| 4. Государства Европы         | 17. Живопись эпохи Возрождения             |
| 5. Русские писатели           | 18. Химические элементы таблицы Менделеева |
| 6. Домашние и дикие животные  | 19. Медицина                               |
| 7. Цветы                      | 20. Туризм и отдых                         |
| 8. Бытовая техника            | 21. Транспорт                              |
| 9. Компьютеры и комплектующие | 22. Цитрусовые растения                    |
| 10. Драгоценные камни         | 23. Горы Северного Кавказа                 |
| 11. Реки России               | 24. Парфюмерия                             |
| 12. Виды спорта               | 25. Канцелярские товары                    |
| 13. Музыка                    |                                            |

## **Задание II**

### **Вариант 1**

1. Описать структуру STUDENT, содержащую следующие поля: фамилия и инициалы, номер группы, успеваемость (массив из пяти элементов).
2. Написать программу, выполняющую следующие действия:
  - ввод с клавиатуры данных в массив экземпляров структур, состоящий из пяти записей;
  - поиск записей, в которых средний бал студента больше 4,0;
  - поиск записей, в которых совпадают номера групп студентов.

### **Вариант 2**

1. Описать структуру AEROFLOT, содержащую следующие поля: название пункта назначения рейса, номер рейса, вместимость, тип самолета.
2. Написать программу, выполняющую следующие действия:
  - ввод с клавиатуры данных в массив экземпляров структур, состоящий из пяти записей;
  - поиск записей, в которых пункт назначения совпадает с введенным с клавиатуры;
  - поиск записей, в которых вместимость самолета не менее введенной с клавиатуры.

### **Вариант 3**

1. Описать структуру TRAIN, содержащую следующие поля: номер поезда, название пункта назначения рейса, время отправления, время в пути.
2. Написать программу, выполняющую следующие действия:
  - ввод с клавиатуры данных в массив экземпляров структур, состоящий из пяти записей;
  - поиск поездов, отправляющихся после времени, введенного с клавиатуры в течении ближайших двух часов;
  - поиск поездов, отправляющихся в пункт назначения, введенный с клавиатуры.

ры.

#### **Вариант 4**

1. Описать структуру TOURIST, содержащую следующие поля: номер маршрута, наименование начального пункта маршрута, наименование конечного пункта маршрута, время в пути.
2. Написать программу, выполняющую следующие действия:
  - ввод с клавиатуры данных в массив экземпляров структур, состоящий из пяти записей;
  - поиск маршрутов, начинающихся в пункте, наименование которого введено с клавиатуры;
  - поиск маршрутов, общее время прохождения которых не превышает времени, введенного с клавиатуры.

#### **Вариант 5**

1. Описать структуру NOTE, содержащую следующие поля: фамилия, имя, номер телефона.
2. Написать программу, выполняющую следующие действия:
  - ввод с клавиатуры данных в массив экземпляров структур, состоящий из пяти записей;
  - поиск информации о человеке, чья фамилия введена с клавиатуры;
  - поиск информации о человеке, чей номер телефона введен с клавиатуры.

#### **Вариант 6**

1. Описать структуру ZNAK, содержащую следующие поля: фамилия, имя, дата рождения, знак зодиака.
2. Написать программу, выполняющую следующие действия:
  - ввод с клавиатуры данных в массив экземпляров структур, состоящий из пяти записей;
  - поиск информации о людях, родившихся сегодня;
  - поиск информации о людях, родившихся под веденным знаком зодиака.

#### **Вариант 7**

1. Описать структуру PRICE, содержащую следующие поля: наименование товара, название магазина, в который товар отгружен, стоимость товара.
2. Написать программу, выполняющую следующие действия:
  - ввод с клавиатуры данных в массив экземпляров структур, состоящий из пяти записей;
  - поиск информации о магазинах, в которые отгружен товар, название которого введено с клавиатуры;
  - поиск информации о товарах, стоимость которых не превышает стоимость, введенную с клавиатуры.

#### **Вариант 8**

1. Описать структуру ORDER, содержащую следующие поля: фамилия налогоплательщика, инициалы налогоплательщика, ИНН налогоплательщика, сумма налогообложения.
2. Написать программу, выполняющую следующие действия:

- ввод с клавиатуры данных в массив экземпляров структур, состоящий из пяти записей;
- поиск информации о налогоплательщике, фамилия которого введена с клавиатуры;
- поиск информации о налогоплательщиках, чей суммарный налог превышает сумму, введенную с клавиатуры.

### **Вариант 9**

1. Описать структуру CAR, содержащую следующие поля: марка автомобиля, государственный регистрационный номер, фамилия владельца, инициалы владельца, пробег поквартально (массив из четырех элементов).
2. Написать программу, выполняющую следующие действия:
  - ввод с клавиатуры данных в массив экземпляров структур, состоящий из пяти записей;
  - поиск информации о владельцах автомобилей, марка которого введена с клавиатуры;
  - поиск информации об автомобилях, у которых суммарный пробег за 2 и 3 кварталы не превышает значение, введенное с клавиатуры.

### **Вариант 10**

1. Описать структуру STUDENT, содержащую следующие поля: фамилия, имя, отчество, серия паспорта, номер паспорта, дата выдачи.
2. Написать программу, выполняющую следующие действия:
  - ввод с клавиатуры данных в массив экземпляров структур, состоящий из пяти записей;
  - поиск записей, в которых совпадает имя студента;
  - поиск записей, в которых совпадает месяц выдачи паспорта.

### **Вариант 11**

1. Описать структуру AEROFLOT, содержащую следующие поля: номер рейса, время отправления, время начала регистрации, тип самолета.
2. Написать программу, выполняющую следующие действия:
  - ввод с клавиатуры данных в массив экземпляров структур, состоящий из пяти записей;
  - поиск рейсов, регистрация на которые уже началась по отношению ко времени, введенному с клавиатуры, но еще не закончилась (временем окончания регистрации считается время отправления минус 20 минут).

### **Вариант 12**

1. Описать структуру WORKER, содержащую следующие поля: фамилия и инициалы работника, дата рождения, год начала трудовой деятельности, год поступления на работу.
2. Написать программу, выполняющую следующие действия:
  - ввод с клавиатуры данных в массив экземпляров структур, состоящий из пяти записей;
  - поиск сотрудников, чей трудовой стаж не менее введенного с клавиатуры;
  - поиск сотрудников, родившихся в течение месяца, введенного с клавиатуры.

### Вариант 13

1. Описать структуру TRAIN, содержащую следующие поля: номер поезда, название пункта назначения рейса, количество спальных, купейных и плацкартных мест (массив из трех элементов), количество свободных спальных, купейных и плацкартных мест (массив из трех элементов).
2. Написать программу, выполняющую следующие действия:
  - ввод с клавиатуры данных в массив экземпляров структур, состоящий из пяти записей;
  - поиск поездов, отправляющихся в пункт назначения, введенный с клавиатуры и имеющих количество свободных плацкартных мест, заданных пользователем.
  - вывод информации о поезде, номер которого введен с клавиатуры.

### Вариант 14

1. Описать структуру NOTE, содержащую следующие поля: фамилия, имя, дата встречи, место встречи.
2. Написать программу, выполняющую следующие действия:
  - ввод с клавиатуры данных в массив экземпляров структур, состоящий из пяти записей;
  - поиск информации о встречах, назначенных на дату, введенную с клавиатуры;
  - поиск информации о встречах, назначенных в месте, введенном с клавиатуры.

### Вариант 15

1. Описать структуру PRICE, содержащую следующие поля: наименование магазина, наименование товара, количество товара на складе, стоимость товара.
2. Написать программу, выполняющую следующие действия:
  - ввод с клавиатуры данных в массив экземпляров структур, состоящий из пяти записей;
  - поиск информации о магазинах, в которых есть товар, название которого введено с клавиатуры;
  - поиск информации о магазинах, в которых есть товар, название которого введено с клавиатуры, со стоимостью, не превышающей значение, введенное с клавиатуры.

### Контрольные вопросы

1. Какие типы данных относятся к простым, а какие к составным?
2. Что такое структура (**struct**)?
3. Как объявляются структуры?
4. Верно ли, что поля структуры должны иметь один и тот же тип?
5. Чем отличается структура от массива?
6. Какие операции допустимы над структурой в целом?
7. Как осуществляется ссылка на компоненты структуры?
8. Можно ли в качестве поля структуры описать структуру?

## Пример выполнения задания I лабораторной работы

**Задание.** Используя структурированный тип **struct** (структуру), составить программу ввода с клавиатуры списка объектов (не менее 10) на тему, заданную в соответствующем варианте задания, 4-6 характеристик этих объектов и вывода данных на экран в виде прямоугольной таблицы. Организовать поиск данных по одной из заданных характеристик.

Информация должна быть достоверной. Ее можно почерпнуть из справочников, энциклопедий, реклам, книг и т.п.

**Тема: Планеты Солнечной системы.**

**1. Решение (информационная модель+словесный алгоритм).** Итак, в качестве примера примитивной реляционной базы данных приведем оформленные в виде таблицы некоторые сведения о планетах Солнечной системы. Таблица будет состоять из заголовка с именами полей, 9 записей (количество планет Солнечной системы) и 6 полей: номер по порядку, название планеты, ее расстояние от Солнца в астрономических единицах (астрономическая единица – это расстояние от Земли до Солнца, равное 149,6 млн.км), период обращения планеты вокруг Солнца в световых годах, наличие у планеты собственных спутников и основной химический состав атмосферы планеты.

Предварительное составление подобной таблицы вручную называется построением информационной модели решаемой задачи. Именно с этого нужно начинать выполнение индивидуального задания.

Таким образом, таблица должна иметь следующий вид:

| №пп. | Название планеты | Расст. от Солнца (в а.е.) | Период обращения (в св.г.) | Наличие спутника | Основной состав атмосферы                          |
|------|------------------|---------------------------|----------------------------|------------------|----------------------------------------------------|
| 1    | Меркурий         | 0.387                     | 0.21                       | -                | Ar, Ne, He                                         |
| 2    | Венера           | 0.700                     | 0.61                       | -                | CO <sub>2</sub>                                    |
| 3    | Земля            | 1.000                     | 1.00                       | +                | N, O <sub>2</sub>                                  |
| 4    | Марс             | 1.520                     | 1.88                       | -                | CO <sub>2</sub>                                    |
| 5    | Юпитер           | 5.200                     | 11.90                      | +                | H <sub>2</sub> , CH <sub>4</sub> , NH <sub>3</sub> |
| 6    | Сатурн           | 9.500                     | 29.46                      | +                | CH <sub>4</sub> , H <sub>2</sub> , He              |
| 7    | Уран             | 19.180                    | 84.00                      | +                | H <sub>2</sub> , He, CH <sub>4</sub>               |
| 8    | Нептун           | 30.070                    | 164.80                     | +                | CH <sub>4</sub> , H <sub>2</sub> , He              |
| 9    | Плутон           | 39.400                    | 247.70                     | +                | CH <sub>4</sub> – метан                            |

При составлении программы ввода-вывода данных в виде приведенной таблицы будем использовать одномерный массив `mas` размером 20, элементы которого имеют тип `struct`. Количество структур (записей  $\Leftrightarrow$  планет) (фактическое количество элементов массива) – `n` будем вводить с клавиатуры.

Кроме описания переменных опишем еще три процедуры с именами `list` (формирование заголовка таблицы и вывод всех данных в виде таблицы на экран), `vvod` (ввод данных) и `search` (поиск информации по названию планеты).

Ширина каждого столбца таблицы определяется значениями соответствующих полей записи, однако, при этом следует учитывать ширину имен полей в заголовке таблицы и ширину экрана. Поэтому при описании функции list стоит помнить о возможности форматированного вывода данных.

Тогда программу можно представить в следующем виде:

## 2. Программа

```
#include <conio.h>
#include <iomanip>
#include <iostream>
#include <string>

using namespace std;

struct
{
 string planeta;
 string sostav;
 float rasst,period;
 char sputnik;
}mas[20];
int n;
string buff;// переменная, используемая в программе для очистки буфера

void list();
void vvod();
void search();

int main()
{
 cout<<"Введите количество планет: ";
 cin>>n;
 system("CLS");
 vvod();
 list();
 search();
 return 0;
}
void vvod() // Процедура ввода данных
{
 system("color 17");
 system("CLS");
 for (int i=1; i<=n; i++)
 {
 system("CLS");
 cout<<" Введите данные для "<<i<<" записи :"<<endl;
 getline(cin,buff); // очистка буфера
 cout<<" Название планеты - ";
 getline(cin,mas[i].planeta);
 cout<<" Расст.от Солнца - ";
 cin>>mas[i].rasst;
 cout<<" Период обращения - ";
 cin>>mas[i].period;
```



```
}
if (f) cout<<" Такой планеты нет!"<<endl;
system("pause");
}
```

При вычерчивании рамок таблицы можно воспользоваться символами псевдографики (Alt+код соответствующего символа в кодовой таблице ASCII, набранный с использованием клавиш цифрового блока клавиатуры).

В результате работы программы должна получиться таблица, подобная приведенной выше.

## **СПИСОК ЛИТЕРАТУРЫ**

1. Конспект лекций.
2. [2] стр. 67-69.

## Лабораторная работа № 15

### Исследование методов доступа к файлам данных

**ЦЕЛЬ РАБОТЫ:** совершенствование навыков в программировании с использованием указателей, исследование функций последовательного и прямого доступа к файлам.

**Формируемые компетенции или их части:** вырабатывается способность к разработке алгоритмических и программных решений в области системного и прикладного программирования, математических, информационных и имитационных моделей, прикладных баз данных, тестов и средств тестирования систем и средств на соответствие стандартам и исходным требованиям (формируется часть компетенций ОПК-3 и ПК-7).

### Теоретическая часть

#### Файлы

**Файлом** называют способ хранения информации на физическом устройстве. Файл – это понятие, которое применимо ко всему – от файла на диске до терминала.

Языки Си и С++ обладают разными возможностями по работе с файловыми потоками. В языке Си отсутствуют операторы для работы с файлами. Все необходимые действия выполняются с помощью функций, включенных в стандартную библиотеку. Они позволяют работать с различными устройствами, такими, как диски, принтер, коммуникационные каналы и т.д. Эти устройства сильно отличаются друг от друга. Однако файловая система преобразует их в единое абстрактное логическое устройство, *называемое потоком*.

В отличие от Си, в С++ файловые потоки обслуживаются с использованием потоковых классов на основе библиотек `<ifsream>` и `<ofstream>`. Рассмотрим каждый из этих способов подробнее.

#### **15.1. Подход с использованием возможностей языка Си**

В Си существует два типа потоков: *текстовые (text)* и *двоичные (binary)*.

**Текстовый поток** – это последовательность символов. При передаче символов из потока на экран, часть из них не выводится (например, символ возврата каретки, перевода строки).

**Двоичный поток** – это последовательность байтов, которые однозначно соответствуют тому, что находится на внешнем устройстве.

Прежде чем читать или записывать информацию в файл, он должен быть открыт и тем самым связан с потоком. Это можно сделать с помощью библиотечной функции `fopen()`. Она берет внешнее представление файла (например, `c:\my_prog.txt`) и связывает его с внутренним логическим именем, которое используется далее в программе. **Логическое имя** – это указатель на требуемый файл. Его необходимо определить; делается это, например, так:

```
FILE *fp;
```

Здесь `FILE` – имя типа, описанное в стандартном заголовочном файле `stdio.h`, `fp` – указатель на файл. Обращение к функции `fopen()` в программе осуществляется выражением:

```
fp = fopen("спецификация_файла",
 "способ_использования_файла");
```

**Спецификация файла** (т.е. физическое имя файла и путь к нему) может, например, иметь вид: `"c:\\my_prog.txt"` – для файла `my_prog.txt` на диске `c:`.

Способ использования файла задается следующими символами:

- `r` – (от `read`) открыть существующий файл для чтения;
- `w` – (от `write`) создать новый файл для записи (если файл с указанным именем существует, то он будет переписан);
- `a` – (от `add`) дополнить файл (открыть существующий файл для записи информации, начиная с конца файла, или создать файл, если он не существует).

Тип файлового потока задается ключом `t` (текстовый) или `b` (бинарный). Если режим `t` или `b` не задан (например, `r`, `w` или `a`), то он определяется значением глобальной переменной `_fmode`. Если `fmode=0_BINARY`, то файлы открываются в двоичном режиме, а если `_fmode=0_TEXT` – в текстовом режиме. Константы `0_BINARY` и `0_TEXT` определены в файле `fcntl.h`.

Если в результате обращения к функции `fopen()` возникает ошибка, то она возвращает указатель на константу `NULL`.

Рекомендуется использовать следующий способ открытия файла:

```
if ((fp = fopen("c:\\my_prog.txt", "rt")) == NULL)
{
 puts("Открыть файл не удалось\n");
 return 0;
}
```

После окончания работы с файлом он должен быть закрыт. Это делается с помощью библиотечной функции `fclose()`. Она имеет следующий прототип:

```
int fclose(FILE *fp);
```

При успешном завершении операции функция `fclose()` возвращает значение нуль. Любое другое значение говорит об ошибке.

Рассмотрим стандартные функции для работы с **файлами последовательного доступа**:

**1. Функция `putc()`** записывает символ в файл и имеет следующий прототип:

```
int putc(int c, FILE *fp);
```

Здесь `fp` – указатель на файл, возвращенный функцией `fopen()`, `c` – символ для записи (переменная `c` имеет тип `int`, но используется только младший байт). При успешном завершении `putc()` возвращает записанный символ, в противном случае возвращается константа `EOF`. Она определена в файле `stdio.h` и имеет значение `(-1)`.

**2. Функция `getc()`** читает символ из файла и имеет следующий прототип:

```
int getc(FILE *fp);
```

Здесь `fp` – указатель на файл, возвращенный функцией `fopen()`. Эта функция возвращает прочитанный символ. Соответствующее значение имеет тип `int`,

но старший байт равен нулю. Если достигнут конец файла, то `getc` возвращает значение EOF.

**3. Функция `feof()`** определяет конец файла при чтении двоичных данных и имеет следующий прототип:

```
int feof(FILE *fp);
```

Здесь `fp` – указатель на файл, возвращенный функцией `fopen()`. При достижении конца файла возвращается ненулевое значение, в противном случае возвращается 0.

**4. Функция `fputs()`** записывает строку символов в файл. Она отличается от функции `puts()` только тем, что в качестве второго параметра должен быть записан указатель на переменную файлового типа.

Например:

```
fputs("Example", fp);
```

При возникновении ошибки возвращается значение EOF.

**5. Функция `fgets()`** читает строку символов из файла. Она отличается от функции `gets()` тем, что в качестве второго параметра должно быть записано максимальное число вводимых символов плюс единица, а в качестве третьего – указатель на переменную файлового типа. Строка считывается целиком, если ее длина не превышает указанного числа символов, в противном случае функция возвращает только заданное число символов. Рассмотрим пример:

```
fgets(string, n, fp);
```

Функция возвращает указатель на строку `string` при успешном завершении и константу `NULL` в случае ошибки либо достижения конца файла.

Рассмотрим другие библиотечные функции, используемые для работы с файлами прямого доступа (все они описаны в файле `stdio.h`):

**1. Функция `fseek()`** позволяет выполнять чтение и запись с произвольным доступом и имеет следующий прототип:

```
int fseek(FILE *fp, long count, int access);
```

Здесь `fp` – указатель на файл, возвращенный функцией `fopen()`, `count` – номер байта относительно заданной начальной позиции, начиная с которого будет выполняться операция, `access` – способ задания начальной позиции.

Переменная `access` может принимать следующие значения:

0 – начальная позиция задана в начале файла;

1 – начальная позиция считается текущей;

2 – начальная позиция задана в конце файла.

При успешном завершении возвращается нуль, при ошибке – ненулевое значение.

**2. Функция `ferror()`** позволяет проверить правильность выполнения последней операции при работе с файлами. Имеет следующий прототип:

```
int ferror(FILE *fp);
```

В случае ошибки возвращается ненулевое значение, в противном случае возвращается нуль.

**3. Функция `remove()`** удаляет файл и имеет следующий прототип:

```
int remove(char *file_name);
```

Здесь `file_name` – указатель на строку со спецификацией файла. При успешном завершении возвращается нуль, в противном случае возвращается ненулевое значение.

**4. Функция `rewind()`** устанавливает указатель текущей позиции в начало файла и имеет следующий прототип:

```
void rewind(FILE *fp);
```

**5. Функция `fread()`** предназначена для чтения блоков данных из потока. Имеет прототип:

```
unsigned fread(void *ptr, unsigned size,
 unsigned n, FILE *fp);
```

Она читает `n` элементов данных, длиной `size` байт каждый, из заданного входного потока `fp` в блок, на который указывает указатель `ptr`. Общее число прочитанных байтов равно произведению `n*size`. При успешном завершении функция `fread()` возвращает число прочитанных элементов данных, при ошибке – 0.

**6. Функция `fwrite()`** предназначена для записи в файл блоков данных. Имеет прототип:

```
unsigned fwrite(void *ptr, unsigned size,
 unsigned n, FILE *fp);
```

Она добавляет `n` элементов данных, длиной `size` байт каждый, в заданный выходной файл `fp`. Данные записываются с позиции, на которую указывает указатель `ptr`. При успешном завершении операции функция `fwrite()` возвращает число записанных элементов данных, при ошибке – неверное число элементов данных.

## **15.2. Подход с использованием возможностей языка C++**

Поток – это абстрактное понятие, относящееся к любому переносу данных от источника к приемнику. Поток `C++` в отличие от функций ввода/вывода в стиле `Си` основаны на понятии класса, которые мы пока не изучаем. Поэтому рассмотрим только порядок использования файловых потоков `C++` без объяснения теории.

Файловый поток для чтения открывается следующим образом:

```
ifstream идентификатор_потока(имя_файла, ключи_потока);
```

Например:

```
ifstream input("input.txt", ios::in);
```

Здесь открывается файловый поток с идентификатором `input` для чтения из файла. Такой файловый поток должен быть по аналогии с `Си` обязательно закрыт: `input.close();`

Файловый поток для записи открывается так:

```
ofstream идентификатор_потока(имя_файла, ключи_потока);
```

Например:

```
ofstream output("output.txt", ios::out);
```

Этот поток также должен быть закрыт:

```
output.close();
```

Как же работать с такими потоками? Очень просто:

```
int main()
{
 int n,k;
 float f;
 char str[100];
 ifstream input("input.txt",ios::in);
 ofstream output("output.txt",ios::out);
 input>>n;
 input>>k;
 n=n+k;
 output<<n<<endl;
 input>>f;
 f=f+2;
 output<<f<<endl;
 input>>str; output<<str<<" ";
 input>>str; output<<str<<"!!!";
 input.close();
 output.close();
 return 0;
}
```

Если исходным содержимым файла `input.txt` было следующее:

```
5
10
-5.56783345
Hello world!
```

то в файл `output.txt` запишется следующее:

```
15
-3.56783
Hello, world!!!
```

## Практическая часть

**ВЫПОЛНЕНИЕ РАБОТЫ:** в соответствии с вариантом составить и реализовать программы.

### Задание I

#### 1. Работа с неструктурированными данными

Для исследования различных методов доступа к файлам данных необходимо выполнить следующие подготовительные действия:

1. Создать в Блокноте следующий текстовый файл:

```
У меня спросили: сколько будет x Опер у ?
А я не знаю! А n Опер k ? Тоже!
Помогите!
```

Например:

У меня спросили: сколько будет  $7 * 2$  ?

А я не знаю! А  $9 / 4$  ? Тоже!

Помогите!

2. Вам известна структура файла. Вывести содержимое файла на экран, а в выходной файл записать результаты:

x Опер y = Рез1

n Опер k = Рез2

Например:

$7 * 2 = 14$

$9 / 4 = 2.25$

3. Исходные данные берутся из таблицы согласно варианта:

| Вар.     | x  | Опер | y  | n  | Опер | k  | Вар.      | x  | Опер | y  | n  | Опер | k  |
|----------|----|------|----|----|------|----|-----------|----|------|----|----|------|----|
| <b>1</b> | 15 | +    | 4  | 7  | *    | 8  | <b>9</b>  | 23 | +    | 37 | 13 | *    | 5  |
| <b>2</b> | 18 | -    | 19 | 18 | /    | 4  | <b>10</b> | 7  | -    | 42 | 37 | /    | 6  |
| <b>3</b> | 9  | *    | 6  | 56 | -    | 37 | <b>11</b> | 34 | *    | 3  | 14 | -    | 53 |
| <b>4</b> | 23 | /    | 5  | 31 | +    | 29 | <b>12</b> | 21 | /    | 5  | 11 | +    | 77 |
| <b>5</b> | 7  | +    | 23 | 14 | /    | 4  | <b>13</b> | 12 | +    | 25 | 20 | /    | 6  |
| <b>6</b> | 34 | -    | 67 | 11 | *    | 3  | <b>14</b> | 15 | -    | 72 | 54 | *    | 2  |
| <b>7</b> | 21 | *    | 2  | 20 | +    | 11 | <b>15</b> | 18 | *    | 4  | 7  | +    | 55 |
| <b>8</b> | 12 | /    | 5  | 54 | -    | 32 | <b>16</b> | 9  | /    | 18 | 18 | -    | 81 |

4. Исследовать возможность решения этой задачи с использованием трех способов:
- функциями последовательного доступа к файлам;
  - функциями прямого доступа к файлам;
  - файловыми потоками C++.
5. Сделать выводы о целесообразности использования того или иного метода считывания или записи данных в файл.

## 2. Работа со структурированными данными

- 1) Используя полученную при выполнении лабораторной работы 14 программу, реализовать возможность сохранения данных в файл с последующим чтением из файла введенных данных.
- 2) Исследовать возможность решения этой задачи с использованием трех способов:
  - функциями последовательного доступа к файлам;
  - функциями прямого доступа к файлам;
  - файловыми потоками C++.
- 3) Сделать выводы о целесообразности использования того или иного метода считывания или записи структурированных данных в файл.

## Задание II

*В соответствии с вариантом написать и отладить программу:*

1. Дана информация о четырех студентах. Структура имеет вид: фамилия, год рождения, место рождения, факультет. Вывести данные о самом старшем по возрасту студенте.

2. Дана информация о шести вкладчиках сберкассы. Структура имеет вид: фамилия, район, соц. положение и величина вклада. Вывести данные о фермерах. Подсчитать их количество.
3. Дана информация о пяти школах. Структура имеет вид: номер школы, год, количество выпускников, число поступивших в вузы. Вывести данные о школе с самым большим отношением числа поступивших к числу выпускников.
4. Дана информация о пяти студентах. Структура имеет вид: фамилия, имя, отчество и 4 оценки за экзамены. Вывести данные о студентах, сдавших сессию на отлично.
5. Дана информация о пяти рабочих цеха. Структура имеет вид: фамилия, пол, размер зарплаты, стаж работы. Вывести данные о рабочем с наибольшей зарплатой.
6. Дана информация о четырех вузах. Структура имеет вид: название вуза, число студентов, количество факультетов, количество кафедр. Вывести данные об общем числе студентов.
7. Дана информация о пяти студентах. Структура имеет вид: фамилия, имя, отчество и 4 оценки за экзамены. Вывести данные о студентах, сдавших сессию без троек.
8. Дана информация о шести вкладчиках сбербанка. Структура имеет вид: фамилия, район, соц. положение и величина вклада. Вывести данные о том, сколько среди вкладчиков служащих. Вычислить их суммарный вклад.
9. Дана информация о пяти больных. Структура имеет вид: фамилия, возраст, пол, давление. Вывести данные о больных с повышенным давлением (более 140) и подсчитать их количество.
10. Дана информация о пяти квартирах. Структура имеет вид: фамилия владельца, площадь, число комнат, этаж. Вывести данные о квартирах с площадью меньше 30 кв.м и подсчитать их количество.
11. Дана информация о шести студентах. Структура имеет вид: фамилия, год рождения, факультет и курс. Вывести данные о студентах в возрасте от 17 до 19 лет и их количестве.
12. Дана информация о шести вкладчиках банка. Структура имеет вид: фамилия, район, соц. положение и величина вклада. Вывести данные о вкладчиках района введенного с клавиатуры.
13. Дана информация о шести студентах. Структура имеет вид: фамилия, рост, вес, год рождения. Вывести данные о студентах с наименьшим ростом и весом от 65 до 75 кг.
14. Дана информация о пяти комнатах в общежитии. Структура имеет вид: факультет, номер комнаты и ее площадь, количество проживающих в ней студентов. Вывести данные о комнате, в которой наименьшая площадь на одного студента.
15. Дана информация о шести вкладчиках сберкассы. Структура имеет вид: фамилия, район, соц. положение и величина вклада. Вывести данные о вкладчике с самой большой величиной вклада.

16. Дана информация о пяти рабочих цеха. Структура имеет вид: фамилия, пол, размер зарплаты, стаж работы. Вывести данные о рабочем с максимальным стажем работы.
17. Дана информация о пяти спортсменах. Структура имеет вид: фамилия, пол, год рождения и рост. Вывести данные о самом высоком спортсмене старше 20 лет.
18. Дана информация о пяти изданиях. Структура имеет вид: название издания, газеты или журнала, стоимость одного экземпляра, количество экземпляров в год. Вывести на печать информацию о самом дешевом издании.
19. Дана информация о шести инвесторах банка. Структура имеет вид: фамилия, район, соц. положение и величина вклада. Вывести данные о том, каков суммарный вклад инвесторов определенного района.
20. Дана информация о шести студентах. Структура имеет вид: фамилия, рост, вес, цвет волос. Вывести данные о блондинах с максимальным ростом.
21. Дана информация о четырех вузах. Структура имеет вид: название вуза, число студентов, количество факультетов, количество кафедр. Вывести данные о вузе с максимальным числом студентов.
22. Дана информация о пяти музыкальных инструментах. Структура имеет вид: название инструмента, число, месяц и год изготовления. Вывести на печать информацию об инструменте с самым большим сроком использования.
23. Дана информация о пяти экземплярах структуры. Структура имеет вид: газета или журнал, название издания, стоимость одного экземпляра, количество экземпляров в год. Вывести на печать информацию о стоимости годовой подписки на все газетные издания.
24. Дана информация о пяти рабочих цеха. Структура имеет вид: фамилия, пол, размер зарплаты, стаж работы. Вывести данные о мужчине с максимальной зарплатой.
25. Дана информация о пяти книгах. Структура имеет вид: название книги, автор, стоимость, количество листов. Вывести данные о заданной книге (по названию и автору) с минимальной стоимостью.
26. Файл содержит информацию о пяти абонентах телефонного справочника. Структура имеет вид: фамилия, номер телефона, дата рождения. Внести в конец списка информацию о трех новых знакомых.
27. Файл содержит информацию о пяти вузах РФ. Структура имеет вид: название вуза, число студентов, количество факультетов. Добавить в конец файла информацию о нескольких вузах.

### **Контрольные вопросы**

1. Понятие файла в C/C++.
2. Классификация файлов.
3. Какого типа могут быть компоненты файла?
4. Нужно ли при определении файла заранее указывать его длину?

5. Чем отличается файл от массива?
6. Понятие о логическом и физическом имени файла.
7. Что понимается под чтением из файла и что под записью в файл?
8. Типы доступа к файлам.
9. Что такое последовательный файл?

### **СПИСОК ЛИТЕРАТУРЫ**

1. Конспект лекций.
2. [2] стр. 151-159.

## СПИСОК ЛИТЕРАТУРЫ

1. Википедия <http://ru.wikipedia.org/wiki>
2. Павловская Т.А. С/C++. Программирование на языке высокого уровня – СПб.: Питер, 2010. – 461 с.
3. Учебник по C++. Текстовая версия. – [myref.ru/preview/74701.html](http://myref.ru/preview/74701.html)
4. Кормен Т., Лейзерсон Ч., Ривест Р. Алгоритмы: построение и анализ. – М.: МЦНМО, 2000.
5. Макоха А.Н. Компьютерные науки. Введение в язык программирования Турбо Паскаль: В 3 ч. Ч. II.: Учебное пособие. – Ставрополь: Изд-во СГУ, 1997.
6. Макоха А.Н. Компьютерные науки. Введение в язык программирования Турбо Паскаль: В 3 ч. Ч. III.: Учебное пособие. – Ставрополь: Изд-во СГУ, 2001.
7. Непретимова Е.В., Саманчук Л.Ф., Корнеев П.К. Программирование. Учебное пособие. Ч1. – Ставрополь: Изд-во СГУ, 2007. – 124 с.
8. Непретимова Е.В., Банько М.А. Введение в язык программирования Паскаль. Учебное пособие. Ч1. – Ставрополь: Изд-во СИУ, 2005. – 120 с.
9. Непретимова Е.В. Введение в язык программирования Паскаль. Учебное пособие. Ч2. – Ставрополь: Изд-во СИУ, 2005. – 104 с.

## Приложение 1

### Содержание отчета, порядок выполнения лабораторных работ и оборудование и материалы

1. Ознакомиться с теоретическим материалом, соответствующим теме лабораторной работы и подготовить конспект, содержащий ответы на предложенные контрольные вопросы.
2. Из списка задач выбрать соответствующие своему варианту. Для каждой задачи разработать математическую модель, алгоритм и программу решения.
3. Реализовать программу на ЭВМ.
4. Для защиты лабораторной работы знать ответы на контрольные вопросы и предоставить **письменный отчёт, содержащий:**
  - а) название лабораторной работы;
  - б) номер варианта;
  - в) решение каждой задачи, включающее в себя:
    - условие задачи;
    - математическую модель;
    - алгоритм;
    - листинг программы с отметкой преподавателя о её выполнении;
    - исходный набор данных и соответствующий ему результат;
    - анализ результатов работы программы.

#### **Оборудование и материалы:**

ПЭВМ с микропроцессором не ниже Pentium 4 объем свободной памяти в ПЗУ не менее 2-3 гб, объем ОЗУ не менее 512 мб.

## Приложение 2

### Базовые функции

| Функция            | Описание                                                                         | Пример                                                         |
|--------------------|----------------------------------------------------------------------------------|----------------------------------------------------------------|
| <b>abs( x )</b>    | возвращает модуль или абсолютное значение от <b>x</b>                            | $\text{abs}(-3.0) = 3.0$<br>$\text{abs}(5.0) = 5.0$            |
| <b>fabs(x)</b>     | абсолютная величина (числа с плавающей точкой)                                   |                                                                |
| <b>sqrt(x)</b>     | корень квадратный из целого <b>x</b> , причём <b>x</b> не отрицательно           | $\text{sqrt}(9) = 3$                                           |
| <b>sqrtf(x)</b>    | корень квадратный из дробного <b>x</b> , причём <b>x</b> не отрицательно         | $\text{sqrt}(9.0) = 3.0$                                       |
| <b>pow(a, b)</b>   | возведение <b>a</b> в степень <b>b</b>                                           | $\text{pow}(2,3) = 8$                                          |
| <b>ceil( x )</b>   | округление <b>x</b> до наименьшего целого, но не меньше чем <b>x</b>             | $\text{ceil}(2.3) = 3.0$<br>$\text{ceil}(-2.3) = -2.0$         |
| <b>floor(x)</b>    | округление <b>x</b> до ближайшего меньшего целого числа                          | $\text{floor}(12.4) = 12$<br>$\text{floor}(-2.9) = -3$         |
| <b>fmod(a, b)</b>  | вычисление остатка от деления нацело для чисел с плавающей точкой ( <b>a/b</b> ) | $\text{fmod}(4.4, 7.5) = 4.4$<br>$\text{fmod}(7.5, 4.4) = 3.1$ |
| <b>modf(x,p)</b>   | извлекает целую и дробную части (с учетом знака) из числа с плавающей точкой     |                                                                |
| <b>exp(a)</b>      | вычисление экспоненты <b>e<sup>a</sup></b>                                       | $\text{exp}(0)=1$                                              |
| <b>frexp(x)</b>    | разбивает число <b>x</b> с плавающей точкой на мантиссу и показатель степени.    |                                                                |
| <b>ldexp(x)</b>    | умножение числа с плавающей точкой на целую степень двух                         |                                                                |
| <b>acos(x)</b>     | арккосинус <b>x</b>                                                              |                                                                |
| <b>asin(x)</b>     | арксинус <b>x</b> , где $-1.0 < x < 1.0$                                         | $\text{asin}(1)=1.5708$                                        |
| <b>atan(x)</b>     | арктангенс <b>x</b>                                                              |                                                                |
| <b>atan2(a, b)</b> | арктангенс с двумя параметрами                                                   |                                                                |
| <b>cos(x)</b>      | косинус <b>x</b> ( <b>x</b> задаётся в радианах)                                 |                                                                |
| <b>sin(x)</b>      | синус <b>x</b> ( <b>x</b> задаётся в радианах)                                   |                                                                |
| <b>sinh(x)</b>     | гиперболический синус <b>x</b>                                                   |                                                                |
| <b>tan(x)</b>      | тангенс <b>x</b>                                                                 |                                                                |
| <b>tanh(x)</b>     | гиперболический тангенс <b>x</b>                                                 |                                                                |
| <b>log(x)</b>      | натуральный логарифм <b>x</b> (основанием является экспонента)                   | $\text{log}(1.0)=0.0$                                          |
| <b>log10(x)</b>    | десятичный логарифм <b>x</b>                                                     | $\text{Log10}(10)=1$                                           |

В таблице показаны основные математические функций, прототипы которых содержатся в заголовочном файле **<math.h>**.

**math.h** – заголовочный файл стандартной библиотеки языка программирования C, разработанный для выполнения простых математических операций. Большинство функций привлекают использование чисел с плавающей точкой. C++ также реализует данные функции для обеспечения совместимости, все они содержатся в заголовочном файле **cmath**.

Все эти функции принимают **double**, если не определено иначе. Для работы с типами **float** и **long double** используются функции с постфиксами **f** и **l** соответственно. Все функции, принимающие или возвращающие угол, работают с радианами.

### **Приложение 3**

#### **Правила поведения в компьютерном классе**

1. К работе в компьютерном классе допускаются лица, ознакомленные с инструкцией по технике безопасности и охране труда, с правилами поведения и размещения информационных ресурсов.
2. Работа студентов в компьютерном классе разрешается только в присутствии преподавателя (инженера, лаборанта).
3. Во время групповых занятий посторонние лица могут находиться в классе только с разрешения преподавателя.
4. **Перед началом работы необходимо:**
  - убедиться в отсутствии видимых повреждений на рабочем месте;
  - разместить на столе тетради, учебные пособия так, чтобы они не мешали работе на компьютере;
  - принять правильную рабочую позу;
  - если сеанс работы предыдущего пользователя не был завершен, завершить его;
  - ввести регистрационную информацию (при необходимости).
5. **При работе в компьютерном классе категорически запрещается:**
  - находиться в классе в верхней одежде;
  - размещать одежду и сумки на рабочих местах;
  - находиться в классе с едой и напитками;
  - класть книги, тетради и т.п. на клавиатуру;
  - располагаться сбоку или сзади от включенного монитора;
  - присоединять или отсоединять кабели, трогать разъемы, провода и розетки;
  - передвигать компьютеры;
  - открывать системный блок;
  - пытаться самостоятельно устранять неисправности в работе аппаратуры;
  - перекрывать вентиляционные отверстия на системном блоке и мониторе;
  - ударять по клавиатуре, нажимать бесцельно на клавиши;
  - удалять или перемещать чужие файлы;
  - устанавливать и запускать компьютерные игры;
  - использовать Интернет-ресурсы неучебного назначения.
6. **Находясь в компьютерном классе, необходимо:**
  - соблюдать тишину и порядок, выключать мобильные телефоны от громкой связи;
  - выполнять все требования преподавателя, инженера и лаборанта;
  - работать только под своим именем и паролем;
  - соблюдать режим работы (продолжительность непрерывной работы за компьютером не более двух часов с обязательным 10-минутным перерывом и гимнастикой для глаз; продолжительность интенсивной работы с клавиатурой не более 30 минут с последующей гимнастикой для рук; общая продолжительность работы не более 4 часов в день);
  - при появлении рези в глазах, резком ухудшении видимости, невозможности сфокусировать взгляд или навести его на резкость, появлении боли

в пальцах и кистях рук, усилении сердцебиения немедленно покинуть рабочее место, сообщить о происшедшем преподавателю и обратиться к врачу;

- после окончания работы завершить все активные программы и корректно завершить сеанс;
- оставить рабочее место чистым.

**7. Работая за компьютером, необходимо соблюдать правильную позу:**

- расстояние от экрана до глаз 70-80 см (расстояние вытянутой руки);
- вертикально прямая спина;
- плечи опущены и расслаблены;
- ноги на полу и не скрещены;
- локти, запястья и кисти рук на одном уровне;
- локтевые, тазобедренные, коленные, голеностопные суставы под прямым углом.

**8. При появлении программных ошибок или сбоях оборудования студент обязан немедленно обратиться к преподавателю (инженеру, лаборанту).**

**9. В случае порчи или выхода из строя оборудования компьютерного класса по вине пользователя ремонт или замена оборудования производится за счет пользователя.**

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ  
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ  
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

## **АЛГОРИТМИЗАЦИЯ И ПРОГРАММИРОВАНИЕ**

**Методические указания для обучающихся по организации и проведению  
самостоятельной работы**

Направление подготовки 03.03.02 Физика  
Направленность (профиль)  
«Фундаментальная и прикладная физика»

**Ставрополь  
2026**

## СОДЕРЖАНИЕ

|                                                                                                      |   |
|------------------------------------------------------------------------------------------------------|---|
| Введение                                                                                             | 3 |
| Методические рекомендации по изучению теоретического материала,<br>подготовке к лабораторным работам | 7 |

## **ВВЕДЕНИЕ**

Методические рекомендации составлены для организации самостоятельной работы студентов с учетом современных требований к подготовке специалистов ВУЗа в связи с широкой компьютеризацией различных отраслей производства, управления, экономики и образования в соответствии с требованиями Федерального государственного образовательного стандарта высшего профессионального образования по направлению подготовки 03.03.02 - Физика. Методические указания содержат необходимые рекомендации, источники получения исходных данных и перечень индивидуальных заданий.

## **1. Цель и задачи освоения дисциплины**

Целью освоения дисциплины является получение знаний и умений необходимых для разработки алгоритмов и составления программ на языке программирования C++, с использованием концепций структурного, модульного и объектно-ориентированного программирования.

Объектами изучения, прежде всего, являются язык программирования C++, методы разработки алгоритмов и программ, методы трансляции, технологии создания программ.

Одна из главных задач изучения дисциплины – формирование алгоритмического стиля мышления и высокого уровня культуры и стиля программирования при изучении конкретного языка программирования высокого уровня, с акцентом на умении выбрать приемлемую структуру данных для решения поставленной задачи, оптимальный алгоритм ее обработки, технологию создания программы и метод трансляции.

## **2. Связь с последующими дисциплинами**

Знания, полученные в результате изучения данной дисциплины, требуются при изучении дисциплин: «Вычислительная физика (практикум на ЭВМ)», «Физические основы элементной базы современных ЭВМ».

## **5. Вопросы для проверки уровня обученности**

**Список контрольных вопросов к каждой теме смотреть в Непретимова Е.В., Корнеев П.К., Ионисян А.С. Алгоритмизация и программирование. Учебное пособие: лабораторный практикум. – Ставрополь: Издательство Северо-Кавказского федерального университета, 2016.– 202 с..**

### **Критерии оценки для проверки уровня обученности:**

Оценка «отлично» выставляется студенту, если студент показал глубокое, прочное и аргументированное освоение программного учебного материала, при этом поставленный вопрос раскрыт последовательно, четко и логически стройно, в полном исчерпывающем объеме, основные категории, понятия и термины учебного курса формулировались правильно, не допущено при ответе ошибок

Оценка «хорошо» выставляется студенту, если студент показал твердое знание программного учебного материала, при этом поставленный вопрос раскрыт грамотно и по существу, в достаточно полном объеме, основные категории, понятия и термины учебного курса формулировались правильно, допущены при ответе отдельные неточности или одна ошибка

Оценка «удовлетворительно» выставляется студенту, если студент показал знание только основной части учебного материала без его частных деталей, при этом поставленный вопрос раскрыт с нарушением логической последовательности, не в полном объеме; были допущены неточные формулировки основных категорий, понятий и терминов учебного курса, а также ошибки (не бо-

лее двух) или ряд незначительных неточностей, не исказивших существенно суть ответа.

Оценка «неудовлетворительно» выставляется студенту, который не знает значительной части программного материала, допускает существенные ошибки (более двух), существенно исказившие его суть. Оценка неудовлетворительно выставляется также, если отсутствует ответ на вопрос, либо студент отказался его сдавать

#### 6. Задачи для проверки полученных умений и навыков по темам:

**Список индивидуальных задач в заданиях, предложенных по каждой теме, смотреть в Непретимова Е.В., Корнеев П.К., Ионисян А.С. Алгоритмизация и программирование. Учебное пособие: лабораторный практикум. – Ставрополь: Издательство Северо-Кавказского федерального университета, 2016.– 202 с..**

#### Вопросы для собеседования

| Вопросы (задача, задание) для проверки уровня обученности |     |                                                                                  |                                                                              |
|-----------------------------------------------------------|-----|----------------------------------------------------------------------------------|------------------------------------------------------------------------------|
| Знать                                                     | №   | Базовый уровень                                                                  | Повышенный уровень                                                           |
|                                                           |     | Этапы решения задач на ЭВМ                                                       | Жизненный цикл ПО                                                            |
|                                                           |     | Свойства алгоритмов                                                              | Основы алгоритмизации                                                        |
|                                                           |     | Способы записи алгоритма                                                         | Основы алгоритмизации                                                        |
|                                                           |     | Базовые алгоритмические конструкции                                              | Основы алгоритмизации                                                        |
|                                                           | 1.  | Теоретические основы программирования                                            | Принцип программного управления. Структура компилятора.                      |
|                                                           | 2.  | Алфавит и лексемы языка Си                                                       | Представление текста в памяти компьютера. Кодовая страница                   |
|                                                           | 3.  | Понятие идентификатора и ключевого слова. Правила создания идентификаторов в C++ | Место компилятора в программном обеспечении.                                 |
|                                                           | 4.  | Переменные и константы                                                           | Спецификаторы типа данных языка Си, их назначение и порядок использования    |
|                                                           | 5.  | Локальные и глобальные переменные.                                               | Спецификаторы области действия переменных                                    |
|                                                           | 6.  | Логические операции языка Си                                                     | Операции языка Си. Общая характеристика                                      |
|                                                           | 7.  | Арифметические операции языка Си. Математические функции языка Си                | Битовые операции языка Си. Трансляция переменных и арифметических выражений. |
|                                                           | 8.  | Стандартные типы данных языка Си                                                 | Данные и способы их организации                                              |
|                                                           | 9.  | Структура программы на языке высокого уровня                                     | Правила записи текста программы. Этапы создания исполняемой программы        |
|                                                           | 10. | Базовые конструкции языка Си. Теорема структуры                                  | Оптимизация кода.                                                            |
|                                                           | 11. | Понятие оператора. Простые операторы языка Си                                    | Организация ввода/вывода данных                                              |
|                                                           | 12. | Условный оператор. Графическое отображение                                       | Условный оператор множественного выбора. Графическое отображение             |
|                                                           | 13. | Понятие цикла с предусловием. Графическое отображение                            | Вложенные циклы. Операторы передачи управления                               |
|                                                           | 14. | Понятие цикла с постусловием, графическое отображение                            | Вложенные циклы. Операторы передачи управления                               |
|                                                           | 15. | Понятие цикла с параметром, графическое отображение                              | Вложенные циклы. Операторы передачи управления                               |
|                                                           | 16. | Понятие массива. Порядок объявления и инициализации массивов.                    | Особенности обработки массивов (ввод/вывод и др. )                           |

|                |     |                                                                                                                                                                                                                                                                                                                       |                                                                                                                                          |
|----------------|-----|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------|
|                | 17. | Понятие указателя. Порядок объявления и инициализации указателей. Операции с указателями                                                                                                                                                                                                                              | Переименование типов и перечисления в языке Си                                                                                           |
|                | 18. | Связь между массивами и указателями. Динамические массивы. Порядок объявления                                                                                                                                                                                                                                         | Функции и массивы. Порядок передачи массивов в функцию                                                                                   |
|                | 19. | Понятие структурированного типа данных. Строки символов и порядок их хранения.                                                                                                                                                                                                                                        | Символьные и строковые функции языка Си                                                                                                  |
|                | 20. | Понятие структурированного типа данных. Определение структур данных и их инициализация                                                                                                                                                                                                                                | Функции и структуры. Порядок передачи структур в функцию                                                                                 |
|                | 21. | Понятие подпрограммы. Понятие функции. Ее назначение, порядок объявления и описания. Понятие и назначение прототипа пользовательской функции.                                                                                                                                                                         | Перегрузка функций                                                                                                                       |
|                | 22. | Передача параметров в функцию. Возвращаемые параметры функции. Назначение возвращаемого значения                                                                                                                                                                                                                      | Передача параметров в функцию (по значению, по адресу, по ссылке)                                                                        |
|                | 23. | Понятие подпрограммы. Понятие функции. Рекурсивные функции. Понятие итеративного процесса                                                                                                                                                                                                                             | Понятие стека. Особенности инициализации и обработки стека.                                                                              |
|                | 24. | Понятие файлов данных и потоков и их классификация                                                                                                                                                                                                                                                                    | Типы файлов данных и ключи доступа. Функции для работы с файлами последовательного доступа. Функции для работы с файлами прямого доступа |
|                | 25. | Постановка задачи сортировки данных. Внутренняя и внешняя сортировки. Алгоритмы решения задач внутренней сортировки                                                                                                                                                                                                   | Сложность алгоритмов. Ускоренные алгоритмы решения задач внутренней сортировки                                                           |
|                | 26. | Алгоритмы поиска информации                                                                                                                                                                                                                                                                                           | Динамические структуры данных. Особенности обработки связанного списка.                                                                  |
| Уметь, владеть |     | Дана квадратная матрица $X$ размером $n \times n$ . Сформировать одномерный массив $Y$ из ее диагональных элементов. Найти сумму диагональных элементов. Оформить ввод, обработку и вывод в виде подпрограмм.                                                                                                         |                                                                                                                                          |
|                |     | Заданы две матрицы $A$ и $B$ размером $7 \times 5$ . Вычислить элементы матрицы $C$ , являющейся разностью двух матриц $A$ и $B$ , то есть $c_{ij} = a_{ij} - b_{ij}$ . Ввод матриц $A$ и $B$ оформить в виде одной процедуры, вычисление матрицы $C$ – в виде другой, вывод матриц $A$ , $B$ и $C$ – в виде третьей. |                                                                                                                                          |
|                |     | Задана матрица $A$ размером $m \times n$ . Найти сумму элементов каждой строки матрицы и записать эти суммы в соответствующие элементы массива $B$ размером $m$ . Оформить ввод и обработку матрицы в виде подпрограмм.                                                                                               |                                                                                                                                          |
|                |     | Задана матрица $A$ размером $4 \times 5$ . Образовать матрицу $B$ , каждый элемент которой равен произведению соответствующего элемента матрицы $A$ на номер строки. Оформить ввод, обработку и вывод матриц в виде подпрограмм.                                                                                      |                                                                                                                                          |
|                | 1.  | Организовать меню: Создание файла, Просмотр файла, Поиск, Выход. В файле содержатся сведения о рейсах самолетов: № рейса, пункт назначения, время вылета, день недели вылета, цена билета. Вывести информацию о рейсах, вылетающих в Москву в понедельник.                                                            |                                                                                                                                          |
|                | 2.  | Организовать меню: Создание файла, Просмотр файла, Поиск, Выход. В файле содержатся сведения о школах: № школы, район, адрес, количество учащихся. Вывести количество учащихся школ Октябрьского района.                                                                                                              |                                                                                                                                          |
|                | 3.  | Задана матрица $A$ размером $4 \times 5$ . Оформить ввод с клавиатуры номера столбца – $k \in [1, 5]$ . Образовать массив $B$ , каждый элемент которого $b_i = A_{ik} \cdot i$ . Оформить ввод, обработку и вывод матриц в виде подпрограмм.                                                                          |                                                                                                                                          |
|                | 4.  | Организовать меню: Создание файла, Просмотр файла, Поиск, Выход. В файле содержатся сведения о студентах: ФИО, группа, специальность, адрес, дата рождения. Вывести список студентов мужского пола старше 18 лет.                                                                                                     |                                                                                                                                          |

|     |                                                                                                                                                                                                                                                                                                            |
|-----|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 5.  | Задана матрица $A$ размером $5 \times 3$ . Построить новую матрицу $B$ , где $b_{ij} = \frac{a_{ij}}{S}$ , где $S = \sum_{i=1}^5 \sum_{j=1}^3 a_{ij}^2$ . Оформить ввод, обработку и вывод матриц в виде подпрограмм.                                                                                      |
| 6.  | Задана матрица $A$ размером $n \times n$ . Найти сумму элементов $k$ -ой строки матрицы, $k$ вводится с клавиатуры. Оформить ввод и обработку матрицы в виде подпрограмм.                                                                                                                                  |
| 7.  | Организовать меню: Создание файла, Просмотр файла, Поиск, Выход. В файле содержатся сведения о студентах: ФИО, группа, специальность, адрес, дата рождения. Вывести список студентов женского пола старше 18 лет                                                                                           |
| 8.  | Задана квадратная матрица $X$ размером $4 \times 4$ . Построить новую матрицу $Y$ , где $y_{ij} = \frac{x_{ij}}{S}$ , $S$ – след матрицы $X$ , то есть сумма элементов ее главной диагонали. Оформить ввод и обработку матрицы $X$ и вывод матрицы $Y$ в виде подпрограмм.                                 |
| 9.  | Организовать меню: Создание файла, Просмотр файла, Поиск, Выход. В файле содержатся сведения: ФИО, адрес, дата рождения, телефон. Вывести сведения о тех, кто родился летом.                                                                                                                               |
| 10. | Организовать меню: Создание файла, Просмотр файла, Поиск, Выход. В файле содержатся сведения о магазинах: № магазина, категория, название, адрес, № телефона. По заданной категории (например, мебель) вывести информацию обо всех магазинах этой категории.                                               |
| 11. | Дана матрица $A$ размером $3 \times 4$ и вектор $B$ размером 3, где каждый элемент вектора равен сумме элементов соответствующей строки матрицы. Найти наименьшую компоненту вектора $B$ . Оформить ввод, обработку и вывод в виде подпрограмм.                                                            |
| 12. | Организовать меню: Создание файла, Просмотр файла, Поиск, Выход. В файле содержатся сведения о студентах: ФИО, адрес, пол, дата рождения. Вывести сведения о студентах мужского пола, родившихся весной.                                                                                                   |
| 13. | Задана матрица $X$ размером $n \times n$ и вектор $B$ размером $n$ . Поменять местами элементы $k$ -ой строки матрицы с элементами вектора, $k$ вводится с клавиатуры. Оформить ввод, обработку и вывод в виде подпрограмм.                                                                                |
| 14. | Задана матрица $A$ размером $4 \times 5$ . Построить новую матрицу $B$ , где $b_{ij} = a_{ij} / m$ , $m$ – наибольший элемент матрицы $A$ . Оформить ввод, обработку и вывод в виде подпрограмм.                                                                                                           |
| 15. | Организовать меню: Создание файла, Просмотр файла, Поиск, Выход. В файле содержатся сведения о магнитофонах: марка, стоимость, завод-изготовитель, дата изготовления. Вывести информацию о магнитофонах ниже заданной стоимости.                                                                           |
| 16. | Организовать меню: Создание файла, Просмотр файла, Поиск, Выход. В файле содержатся сведения о владельцах автомобилей: ФИО владельца, адрес, № автомобиля, марка автомобиля, цвет автомобиля. Вывести сведения о владельцах, имеющих автомобиль заданной марки и цвета.                                    |
| 17. | Заданы две матрицы $A$ и $B$ размером $7 \times 5$ . Вычислить элементы матрицы $C$ , являющейся разностью двух матриц $A$ и $B$ , то есть $c_{ij} = a_{ij} - b_{ij}$ . Ввод матриц $A$ и $B$ оформить в виде одной процедуры, вычисление матрицы $C$ – в виде другой, вывод матрицы $C$ – в виде третьей. |
| 18. | Организовать меню: Создание файла, Просмотр файла, Поиск, Выход. В файле содержатся сведения о заказанных товарах: название товара, цена единицы товара, количество товара, название заказчика, адрес заказчика. Вывести информацию о товарах, стоимость которых превышает 1000 руб.                       |
| 19. | Задана матрица $A$ размером $m \times n$ . Сформировать массив $B$ из максимальных элементов каждой строки матрицы $A$ . Упорядочить элементы вектора $B$ по убыванию. Оформить ввод и обработку матрицы в виде подпрограмм.                                                                               |
| 20. | Задана матрица $A$ размером $m \times n$ . Сформировать массив $B$ из максимальных элементов каждой строки матрицы $A$ . Оформить ввод и обработку матрицы в виде подпрограмм.                                                                                                                             |
| 21. | В квадратной матрице $A$ размером $4 \times 4$ найти и вывести элементы, удовлетворяющие условию $a_{ij} > b_i$ , где $B$ – одномерный массив размером 4. Оформить ввод и обработку матрицы в виде подпрограмм.                                                                                            |

|  |     |                                                                                                                                                                                                                                                                                                 |
|--|-----|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|  | 22. | Организовать меню: Создание файла, Просмотр файла, Поиск, Выход. В файле содержатся сведения об институтах: название института, адрес института, название специальности, количество студентов на специальности. Вывести названия и адреса институтов, в которых имеется заданная специальность. |
|--|-----|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

## 7. Рекомендуемая литература и интернет-ресурсы

### Основная литература:

1. Окулов С.М. Алгоритмы обработки строк. – М.: "Лаборатория знаний" – 2015 – 258 с.
2. Паронджанов В.Д. Учись писать, читать и понимать алгоритмы. Алгоритмы для правильного мышления. Основы алгоритмизации. – ДМК Пресс. – 2012 – 520 с
3. Арипова О.В., Гушин А.Н., Палехова О.А. Программирование на языке высокого уровня: лабораторный практикум для вузов. – БГТУ "Военмех" им. Д.Ф. Устинова – 2014 – 164 с.
4. Павловская Т.А. С/С++. Программирование на языке высокого уровня – СПб.: Питер, 2010. – 461 с.

### Дополнительная литература:

1. Александров Э. Э., Афонин В. В. Программирование на языке С в Microsoft Visual Studio 2010 - М.: Интернет-Университет Информационных Технологий, 2010
2. Березин Б. И., Березин С. Б. Начальный курс С и С++ - М.: Диалог-МИФИ, 2008
3. Архангельский А.Я. Язык С++ в С++ Builder. –М.: ООО «Бином-Пресс», 2008. – 944 с.
4. Архангельский А.Я. Программирование в С++ Builder 6. –М.: Бином, 2007. – 1152 с.
5. Прата, Стивен. Язык программирования С++. Лекции и упражнения, 5-е изд.: Пер. с англ. – М.: ООО «И.Д.Вильямс», 2007. – 1184 с.
6. Страутструп Б. Язык программирования С++. 2-е издание. –М.: Бином, 2000. –358 с.
7. Шелупанов А.А., Кирнос В.Н. Информатика. Базовый курс. Ч.3. Основы алгоритмизации и программирования в среде Visual C++ 2005: Учеб. пособие. – ТУСУР. – 2008 – 216 с.
8. С/С++. Структурное и объектно-ориентированное программирование: Практикум / Т.А.Павловская, Ю.А.Щупак. – СПб.: Питер, 2011. – 239 с.

### Методическая литература:

1. Непретимова Е.В., Корнеев П.К., Ионисян А.С. Алгоритмизация и программирование. Учебное пособие: лабораторный практикум. – Ставрополь: Издательство Северо-Кавказского федерального университета, 2016.– 202 с.
2. Непретимова Е.В., Корнеев П.К., Саманчук Л.Ф. Программирование. Учеб. пособие. Ч.І. – Ставрополь: Изд-во СГУ, 2007

### Интернет-ресурсы:

- <http://www.computer-museum.ru> (Еженедельник PC-week RE и виртуальный компьютерный музей).
- [www.osp.ru](http://www.osp.ru) (издательство Открытые системы).
- [www.ibxt.ru](http://www.ibxt.ru) (новости вычислительной техники)
- <https://ru.wikipedia.org/wiki> Википедия свободная энциклопедия.
- <http://www.intuit.ru> Национальный открытый университет ИНТУИТ