

ПЕРМСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ  
Кафедра Информационных технологий и  
автоматизированных систем

Викентьева О. Л., Гусин А. Н., Полякова О. А.

Программирование на языке С++  
Лабораторный практикум  
для студентов специальностей АСУ и ЭВТ

Пермь 2006

Методические указания к лабораторным работам по дисциплине "Алгоритмические языки и программирование"

Составители: Викентьева О. Л., к. т. н., доцент,  
Гусин А. Н., старший преподаватель,  
Полякова О. А., к. т. н., доцент.

Приведены методические указания по выполнению лабораторных работ по дисциплине "Алгоритмические языки и программирование", изучаемой в 1 семестре.

Введение.....	6
Среда программирования Visual C++ 6.0.....	6
1.1. Общий вид окна.....	6
1.2. Создание консольного приложения и работа с ним.....	7
1.3. Компиляция и запуск проекта.....	8
1.4. Отладка программы.....	9
1.5. Создание рабочего пространства для нескольких проектов.....	9
Лабораторная работа №1.....	10
Выполнение программы простой структуры. Вычисление выражений с использованием стандартных функций.....	10
1. Цель задания:.....	10
2. Теоретические сведения.....	10
2.1. Структура программы на C++.....	10
2.2. Элементы языка C/C++.....	12
2.3. Константы в C/C++.....	12
2.3. Типы данных в C++.....	13
2.4. Переменные.....	14
2.5. Операции.....	14
2.6. Выражения.....	16
2.7. Ввод и вывод данных.....	16
3. Постановка задачи.....	17
4. Варианты.....	18
5. Методические указания.....	23
6. Содержание отчета.....	23
Лабораторная работа №2.....	24
Использование основных операторов языка C++.....	24
1. Цель задания:.....	24
2. Теоретические сведения.....	24
2.1. Составные операторы.....	24
2.2. Операторы выбора.....	24
2.3. Операторы циклов.....	26
2.4. Операторы перехода.....	27
3. Постановка задачи.....	28
4. Варианты.....	30
5. Методические указания.....	31
6. Содержание отчета.....	31
Лабораторная работа №4 Работа с одномерными массивами.....	32
1. Цель работы:.....	32
2. Краткие теоретические сведения.....	32
2.1. Определение массива в C/C++.....	32
2.2. Понятие указателя.....	32
2.3. Одномерные массивы и указатели.....	34
2.4. Перебор элементов массива.....	35
2.5. Классы задач по обработке массивов.....	35
2.4. Сортировка массивов.....	36
2.5. Поиск в отсортированном массиве.....	37
3. Постановка задачи.....	38
4. Варианты.....	38
5. Методические указания.....	41
6. Содержание отчета:.....	41

Лабораторная работа №4 .....	42
Функции и массивы в C++ .....	42
1. Цель работы: .....	42
2. Теоретические сведения .....	42
2.1. Параметры функции .....	43
2.2. Локальные и глобальные переменные .....	44
2.3. Передача одномерных массивов как параметров функции .....	44
2.4. Передача строк в качестве параметров функций .....	44
2.5. Передача многомерных массивов в функцию .....	45
2.6. Строки .....	45
3. Постановка задачи .....	47
4. Варианты .....	47
5. Методические указания .....	49
6. Содержание отчета .....	50
Лабораторная работа №5 .....	51
Динамические массивы .....	51
1. Цель работы: .....	51
2. Теоретические сведения .....	51
3. Постановка задачи .....	53
4. Варианты .....	53
5. Методические указания .....	54
6. Содержание отчета .....	55
Лабораторная работа №6 .....	55
Массивы структур и массивы строк .....	55
1. Цель работы: .....	55
2. Теоретические сведения .....	55
2.1. Структуры .....	55
3. Постановка задачи .....	56
4. Варианты .....	56
5. Методические указания .....	59
6. Содержание отчета .....	59
Лабораторная работа №7 .....	60
Функции в C++ .....	60
1. Цель работы: .....	60
2. Теоретические сведения .....	60
2.1. Функции с начальными значениями параметров (по-умолчанию) .....	60
2.2. Функции с переменным числом параметров .....	60
2.3. Перегрузка функций .....	62
2.3. Шаблоны функций .....	63
2.4. Указатель на функцию .....	64
2.5. Численные методы решения уравнений .....	65
3. Постановка задачи .....	67
4. Варианты .....	67
5. Методические указания .....	70
6. Содержание отчета .....	71
Лабораторная работа №8 .....	72
Динамические структуры данных .....	72
1. Цель работы: .....	72
2. Краткие теоретические сведения .....	72
2.1. Однонаправленные списки .....	72
2.1. Двухнаправленные списки .....	75
2.3. Очередь и стек .....	76

2.4. Бинарные деревья .....	76
3. Постановка задачи .....	78
4. Варианты .....	79
5. Методические указания .....	82
6. Содержание отчета .....	83
Лабораторная работа №9 .....	84
Хранение данных на внешних носителях .....	84
1. Цель работы: .....	84
2. Краткие теоретические сведения .....	84
2.1. Поточковый ввод-вывод в стиле C .....	84
2.2. Обработка элементов файла .....	87
2.3. Поточковый ввод-вывод в стиле C++ .....	88
3. Постановка задачи .....	90
5. Содержание отчета .....	94

## Введение

Для того, чтобы научиться программировать, в первую очередь, надо научиться строить и записывать алгоритмы решаемых задач. Алгоритм – это точное предписание, определяющее вычислительный процесс, идущий от изменяемых начальных данных к конечному результату, т. е. это рецепт достижения какой-либо цели. Совокупность средств и правил для представления алгоритма в виде пригодном для выполнения вычислительной машиной называется языком программирования, алгоритм, записанный на этом языке – программой. Для записи алгоритмов существуют разные формы:

- 1) словесное описание (псевдокоды),
- 2) графическое описание (блок-схемы),
- 3) алгоритмические языки.

Для того чтобы составить программу желательно выполнить по порядку следующие этапы:

- 1) Определить исходные данные задачи и результаты, которые должны быть получены, а также формулы, связывающие исходные данные и результаты.
- 2) Составить алгоритм в виде блок-схемы, с помощью которого можно от исходных данных перейти к результатам.
- 3) Записать алгоритм на требуемом языке программирования (т. е. каждому блоку блок-схемы надо поставить в соответствие оператор языка программирования).
- 4) Выполнить программу, используя какую-то систему программирования.
- 5) Выполнить отладку и тестирование программы. При выполнении программы могут возникать ошибки трех типов:

Самыми опасными являются именно семантические ошибки, т. к. их достаточно сложно обнаружить. Программа будет работать, но неправильно, причем, ошибки в ее работе могут возникать не все время, а только при каких-то определенных наборах исходных данных. Для обнаружения таких ошибок выполняется тестирование программы. Набор исходных данных, для которых известен результат, называется тестом. Если результаты работы теста не совпадут с известным значением, значит, в программе имеется ошибка. Тест, выявивший ошибку, считается успешным. Отладка программы заканчивается, когда достаточное количество тестов будет выполнено неуспешно. Самым распространенным критерием для определения количества неуспешных тестов является тестирование ветвей: набор тестов в совокупности должен обеспечить прохождение каждой ветви не менее одного раза.

Начинающие программисты должны обязательно выполнять все указанные этапы. В дальнейшем этапы 2-3 можно объединить в один и сразу записывать программу на требуемом языке программирования.

В качестве изучаемого языка программирования выбран C++, т. к. этот язык позволяет выработать алгоритмическое мышление, стоит короткую программу, продемонстрировать основные приемы алгоритмизации.

## Среда программирования Visual C++ 6.0

### 1.1. Общий вид окна

Проект (project) – это набор файлов, которые совместно используются для создания одной программы.

Рабочее пространство (workspace) может включать в себя несколько проектов.

После запуска VC++ 6.0 на экране появится окно (рис. 1).

Рис. 1. Окно VC++ 6.0.

Окно содержит:

- Главное меню (1) – список основных команд VC++;
- Панель инструментов (2) - панель с кнопками команд Visual C++;
- Панель рабочего пространства Workspace (3) - содержит две вкладки:
  - ClassView – отображает список классов в проекте,
  - FileView – отображает список файлов, входящих в проект.
- Окно для редактирования кодов (4) – окно с текстом программы;
- Выходную панель результатов компиляции (5) - окно для вывода сообщений в процессе компиляции или отладки, показывает текущую стадию компиляции, список ошибок и предупреждений и их количество.

### **1.2. Создание консольного приложения и работа с ним**

Консольное приложение – это приложение, которое с точки зрения программиста является программой DOS, но может использовать всю доступную оперативную память (если каждый элемент данных программы не будет превышать 1 Мб). Этот тип приложения запускается в особом окне, которое называется “Окно MS-DOS”. На примере консольных приложений прослеживаются этапы развития VC++ при переходе от одной версии к другой.

Каждое приложение, разрабатываемое как отдельный проект в среде VC++6.0, нуждается в том, чтобы ему соответствовало свое собственное рабочее пространство. Рабочее пространство включает в себя те папки, в которых будут храниться файлы, содержащие информацию о конфигурации проекта. Для того чтобы создать новое пространство для проекта, надо выполнить следующие действия:

1. В линейке меню нажать на меню **File**.
2. Выбрать пункт **New** или нажать **Ctrl+N**.
3. Появится окно **New**. В нем содержится четыре вкладки: Files, Projects, Workspaces, Other Documents. Выбрать вкладку Projects.

4. Из списка возможных проектов выбрать **Win32 Console Application** для создания приложения DOS.

5. В поле **Project name** ввести имя проекта.

6. В поле **Location** ввести путь для размещения каталога проекта, или, нажав на кнопку справа [...], выбрать нужную директорию.

7. Должен быть установлен флажок **Create New Workspace**. Тогда будет создано новое рабочее окно. Нажать кнопку **OK**

8. Установить один из флажков:

- **An empty project** – создается пустой проект, не содержащий заготовок для файлов;

- **A simple application** – создается простейшая заготовка, состоящая из заголовочного файла StdAfx.h, файла StdAfx.cpp и файла реализации;

- **A “Hello World” application** и **An application that supports MFC** являются демонстрационными и разными способами демонстрируют вывод на экран строки символов.

Нажать кнопку **Finish**. Появится информация о созданном проекте содержащая: тип проекта, некоторые особенности и директорию.

После создания проекта в него необходимо записать код программы. При этом можно создать новый файл или добавить в проект существующий файл.

Для создания нового файла надо выполнить следующие действия:

1. Выбрать меню **File > New** или **Project > Add to Project > New**.

2. Открыть вкладку **Files**.

3. Выбрать **C++ Source File**.

4. Чтобы создаваемый файл был автоматически присоединен к проекту, необходимо установить флаг **Add to project**.

5. В поле **Filename** ввести имя файла.

6. В поле **Location** указать путь для создания файла.

7. Нажать **OK**.

Для добавления существующего файла надо:

1. Выбрать в меню **File > Add to Project > Files**

2. Указать полное имя файла, который нужно присоединить

Для открытия существующего проекта надо:

1. Выбрать меню **File > Open Workspace**

2. Указать файл с расширением **.dsw**

Для сохранения текущего проекта надо выбрать в главном меню **File > Save Workspace**.

Для закрытия текущего проекта надо выбрать в главном меню **File > Close Workspace**.

После создания или открытия проекта в окне **Workspace** появится или список классов, или список файлов входящих в проект. В зависимости от типа проекта, он будет или пустой, или содержать изначально некоторые файлы, присущие данному типу. Проект приложения для DOS изначально пустой. В него можно добавить новые файлы или присоединить уже существующие.

### **1.3. Компиляция и запуск проекта**

Для компиляции проекта надо выбрать в главном меню **Build > Build <имя проекта>** или нажать клавишу F7.

Visual C++ 6.0 откомпилирует исходные файлы и создаст соответствующие файлы с расширением **.obj**. Затем эти файлы соединятся в исполняемый файл. Весь процесс компиляции и создания исполняемого файла отображается в окне Output, вкладка Build. После компиляции файла его можно запустить.



Для запуска исполняемого файла надо выбрать в главном меню **Build > Execute <имя файла>.exe** или нажмите клавиши **Ctrl+F5** . Если файл был создан, то он запустится. Для повторного запуска файла не нужно его снова компилировать. Но если в программу были внесены изменения, то перед запуском необходимо выполнить компиляцию. Выполняется именно файл с расширением .exe, а не текущий проект, т.е. в процессе запуска компиляции не происходит.

#### **1.4. Отладка программы**

Для отладки программы используется команда главного меню **Build>Start Debug> Step Into** – отладка с заходом в функции, которая начинается с первой строки функции `main` или **Build>Start Debug> Run to Cursor** – выполнение программы до курсора, т. е. отладка начинается с той строки, в которой установлен курсор. После выполнения этой команды выполнение программы происходит в режиме отладчика. Переход к следующей строке программы можно выполнять с помощью команды **Step Into (F11)** (с заходом во все вызываемые функции) или с помощью команды **Step over (F10)** (без захода в вызываемые функции). Выход из функции нижнего уровня выполняется командой **Step Out (Shift+F11)**. Текущие значения переменных можно просматривать:

- 1) в специальных окнах **Watch** (отображает значения всех используемых переменных) и **Value** (отображает значения заданных пользователем переменных);
- 2) при наведении курсора мышки на переменную отображается текущее значение этой переменной.

#### **1.5. Создание рабочего пространства для нескольких проектов**

Несколько проектов можно объединить в одно рабочее пространство с помощью команды **Project/Insert Project into Workspace**. Активный проект, т. е. тот, который будет выполняться, устанавливается с помощью команды **Project/Set Active Project**. Активный проект надо отметить галочкой.

# Лабораторная работа №1

## Выполнение программы простой структуры. Вычисление выражений с использованием стандартных функций

### 1. Цель задания:

- 1) Выполнение простой программы в системе программирования VC++6.0
- 2) Приобретение навыков в записи выражений на языке C++ и использование стандартных функций.

### 2. Теоретические сведения

#### 2.1. Структура программы на C++

Программа на языке Си имеет следующую структуру:

#директивы препроцессора

.....

#директивы препроцессора

функция а ( )

операторы

функция в ( )

операторы

void main ( ) //функция, с которой начинается выполнение программы

операторы

описания

присваивания

функция

пустой оператор

составной

выбора

циклов

перехода

Директивы препроцессора управляют преобразованием текста программы до ее компиляции. Исходная программа, подготовленная на C++ в виде текстового файла, проходит 3 этапа обработки:

- 1) препроцессорное преобразование текста;
- 2) компиляция;
- 3) компоновка (редактирование связей или сборка).

## Рис. 2. Обработка С++ программы

После этих трех этапов формируется исполняемый код программы. Задача препроцессора – преобразование текста программы до ее компиляции. Правила препроцессорной обработки определяет программист с помощью директив препроцессора. Директива начинается с #.

`#define` – указывает правила замены в тексте.

`#include`<имя заголовочного файла> – директива предназначена для включения в текст программы текста из каталога заголовочных файлов, поставляемых вместе со стандартными библиотеками. Каждая библиотечная функция С имеет соответствующее описание в одном из заголовочных файлов. Список заголовочных файлов определен стандартом языка. Употребление директивы `include` не подключает соответствующую стандартную библиотеку, а только позволяют вставить в текст программы описания из указанного заголовочного файла. Если используется заголовочный файл из стандартной библиотеки, то его имя заключают в угловые скобки. Если используется заголовочный файл, который находится в текущем каталоге проекта (он может быть создан разработчиком программы), то его имя заключается в кавычки. Подключение кодов библиотеки осуществляется на этапе компоновки, т. е. после компиляции. Хотя в заголовочных файлах содержатся все описания стандартных функций, в код программы включаются только те функции, которые используются в программе.

После выполнения препроцессорной обработки в тексте программы не остается ни одной препроцессорной директивы.

Программа представляет собой набор описаний и определений, и состоит из набора функций. Среди этих функций всегда должна быть функция с именем `main`. Без нее программа не может быть выполнена. Перед именем функции помещаются сведения о типе возвращаемого функцией значения (тип результата). Если функция ничего не возвращает, то указывается тип `void`: `void main()`. Каждая функция, в том числе и `main`, должна иметь список параметров. Список может быть пустым, тогда он указывается как `(void)` (слово `void` может быть опущено: `()`).

За заголовком функции размещается тело функции. Тело функции – это последовательность определений, описаний и исполняемых операторов, заключенных в фигурные скобки. Каждое определение, описание или оператор заканчивается точкой с запятой.

Определения – вводят объекты (объект – это именованная область памяти, частный случай объекта – переменная), необходимые для представления в программе обрабатываемых данных. Примерами являются

```
const int y = 10 ; //именованная константа
```

```
float x ; //переменная
```

Описания – уведомляют компилятор о свойствах и именах объектов и функций, описанных в других частях программы.

Операторы – определяют действия программы на каждом шаге ее исполнения.

## 2.2. Элементы языка C/C++

- 1) Алфавит языка который включает
  - прописные и строчные латинские буквы и знак подчеркивания;
  - арабские цифры от 0 до 9;
  - специальные знаки "{ } , | [ ] ( ) + - / % \* . \ ' : ; & ? < > = ! # ^
  - пробельные символы (пробел, символ табуляции, символы перехода на новую строку).
- 2) Из символов формируются лексемы языка:
  - *Идентификаторы* – имена объектов C/C++-программ. В идентификаторе могут быть использованы латинские буквы, цифры и знак подчеркивания. Прописные и строчные буквы различаются, например, PROG1, prog1 и Prog1 – три различных идентификатора. Первым символом должна быть буква или знак подчеркивания (но не цифра). Пробелы в идентификаторах не допускаются.
  - *Ключевые (зарезервированные) слова* – это слова, которые имеют специальное значение для компилятора. Их нельзя использовать в качестве идентификаторов.
  - *Знаки операций* – это один или несколько символов, определяющих действие над операндами. Операции делятся на унарные, бинарные и тернарную по количеству участвующих в этой операции операндов.
  - *Константы* – это неизменяемые величины. Существуют целые, вещественные, символьные и строковые константы. Компилятор выделяет константу в качестве лексемы (элементарной конструкции) и относит ее к одному из типов по ее внешнему виду.
  - *Разделители* – скобки, точка, запятая пробельные символы.

## 2.3. Константы в C/C++

Константа – это лексема, представляющая изображение фиксированного числового, строкового или символьного значения. Константы делятся на 5 групп:

- целые;
- вещественные (с плавающей точкой);
- перечислимые;
- символьные;
- строковые.

Компилятор выделяет лексему и относит ее к той или другой группе, а затем внутри группы к определенному типу по ее форме записи в тексте программы и по числовому значению.

*Целые константы* могут быть десятичными, восьмеричными и шестнадцатеричными.

Название	Определение	Примеры
Десятичная константа	Последовательность десятичных цифр, начинающаяся не с 0, если это число не 0	8, 0, 192345
Восьмеричная константа	Последовательность восьмеричных цифр, которым предшествует 0.	026, 034, 017

Шестнадцатеричная константа	Последовательность шестнадцатеричных цифр, которым предшествуют символы 0x или 0X	0xA, 0X00F, 0x123
-----------------------------	---	-------------------

*Вещественные константы* могут иметь две формы представления: с фиксированной точкой и с плавающей точкой.

Название	Вид	Примеры
Константы с фиксированной точкой	[цифры].[цифры]	5.7, .0001, 41.
Константа с плавающей точкой	[цифры][.][цифры]E e[+ -] [цифры]	0.5e5, .11e-5, 5E3

*Перечислимые константы* вводятся с помощью ключевого слова `enum`. Это обычные целые константы, которым приписаны уникальные и удобные для использования обозначения.

```
enum {one=1, two=2, three=3, four=4};
enum {zero, one, two, three};
enum {ten=10, three=3, four, five, six};
enum {Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday};
```

*Символьные константы* – это один или два символа, заключенные в апострофы. Символьные константы, состоящие из одного символа, имеют тип `char` и занимают в памяти один байт, символьные константы, состоящие из двух символов, имеют тип `int` и занимают два байта. Последовательности, начинающиеся со знака `\`, называются управляющими, они используются:

- для представления символов, не имеющих графического отображения, например:
  - `\a` – звуковой сигнал,
  - `\b` – возврат на один шаг,
  - `\n` – перевод строки,
  - `\t` – горизонтальная табуляция;
- для представления символов: `\, ', ?, " (\, \', \?, \")`;
- для представления символов с помощью шестнадцатеричных или восьмеричных кодов (`\073`, `\0xF5`);

*Строковая константа* – это последовательность символов, заключенная в кавычки. Внутри строк также могут использоваться управляющие символы. Например:

```
"\nНовая строка",
"\n\"Алгоритмические языки программирования\""
```

## 2.3. Типы данных в C++

Типы C++ можно разделить на простые и составные. К простым типам относят типы, которые характеризуются одним значением. В языке C++ определено 6 простых типов данных:

<code>int</code> (целый)	}	целочисленные
<code>char</code> (символьный)		
<code>wchar_t</code> (расширенный символьный) (C++)		
<code>bool</code> (логический) (C++)		
<code>float</code> (вещественный)	}	с плавающей точкой
<code>double</code> (вещественный с двойной точностью)		

Существует 4 спецификатора типа, уточняющих внутреннее представление и диапазон стандартных типов

short (короткий)  
 long (длинный)  
 signed (знаковый)  
 unsigned (беззнаковый)

Тип данных	Определение	Размер	Диапазон
(signed) char	Значениями являются элементы конечного упорядоченного множества символов. Каждому символу ставится в соответствие число, которое называется кодом символа.	1 байт	-128..127
unsigned char			0..255
wchar_t	Значениями являются элементы конечного упорядоченного множества символов в кодировке Unicode	2 байта	0..65535
(signed) int	Значениями являются целые числа.	4 байта (для 32-разрядного МП)	-2147483648 ... +2147483647.
(signed) long (int)			0...+4294967 295.
unsigned int			
unsigned long (int)		2 байта (для 32-разрядного МП)	-32768 ... +32767
(signed) short int			0 ... 65536;
unsigned short int			
bool	Данные этого типа могут принимать значения true и false.	1 байт	false, true
float	Значениями являются вещественные числа	4 байта	3.4E-38..3.4E+38
double		8 байт	1.7E-308 ..1.7E+308
long double		10 байт	3.4E-4932..1E+4932

## 2.4. Переменные

Переменная в C++ – 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. Перед использованием любая переменная должна быть описана.

```
int a; float x;
```

## 2.5. Операции

В соответствии с количеством операндов, которые используются в операциях они делятся на унарные (один операнд), бинарные (два операнда) и тернарную (три операнда).

Операция	Описание
Унарные операции	

++	Увеличение на единицу: префиксная операция - увеличивает операнд до его использования, постфиксная операция увеличивает операнд после его использования.
--	Уменьшение на единицу: префиксная операция - уменьшает операнд до его использования, постфиксная операция уменьшает операнд после его использования.
sizeof	вычисление размера (в байтах) для объекта того типа, который имеет операнд
-	Унарный минус
+	Унарный плюс
!	Логическое отрицание (НЕ). В качестве логических значений используется 0 (false) - ложь и не 0 (true) - истина, отрицанием 0 будет 1, отрицанием любого ненулевого числа будет 0.
&	Получение адреса операнда
*	Получение значения, находящегося по указанному адресу (разыменованное)
new	Выделение памяти
delete	Освобождение памяти
(type)	Преобразование типа
Бинарные операции	
Мультипликативные	
*	умножение операндов арифметического типа
/	деление операндов арифметического типа (если операнды целочисленные, то выполняется целочисленное деление)
%	получение остатка от деления целочисленных операндов
Аддитивные	
+	бинарный плюс (сложение арифметических операндов)
-	бинарный минус (вычитание арифметических операндов)
Операции сравнения	
<	меньше, чем
<=	меньше или равно
>	больше
>=	больше или равно
=	равно
!=	не равно
Логические о	
&&	конъюнкция (И) целочисленных операндов или отношений, целочисленный результат ложь(0) или истина( не 0)
	дизъюнкция (ИЛИ) целочисленных операндов или отношений, целочисленный результат ложь(0) или истина(не 0)
Тернарная	
?:	Условная операция в ней используется три операнда. Выражение1 ? Выражение2 : Выражение3; Первым вычисляется значение выражения1. Если оно истинно, то вычисляется значение выражения2, которое становится результатом. Если при вычислении выражения1 получится 0, то в качестве результата берется значение выражения3. Например: x<0 ? -x : x ; //вычисляется абсолютное значение x.

Присваивание	
=	присваивание
*=	умножение с присваиванием (мультипликативное присваивание)
/=	деление с присваиванием
%=	деление с остатком с присваиванием
+=	сложение с присваиванием
-=	вычитание с присваиванием

Приоритеты операций.

Ранг	Операции
1	( ) [ ] -> .
2	! ~ - ++ -- & * (тип) sizeof тип( )
3	* / % (мультипликативные бинарные)
4	+ - (аддитивные бинарные)
5	< > <= >= (отношения)
6	== != (отношения)
7	&& (конъюнкция «И»)
8	(дизъюнкция «ИЛИ»)
9	?: (условная операция)
10	= *= /= %= -= &= ^=  = <<= >>= (операция присваивания)
11	, (операция запятая)

## 2.6. Выражения

Из констант, переменных, разделителей и знаков операций можно конструировать выражения. Каждое выражение представляет собой правило вычисления нового значения. Каждое выражение состоит из одного или нескольких операндов, символов операций и ограничителей. Если выражение формирует целое или вещественное число, то оно называется арифметическим. Пара арифметических выражений, объединенная операцией сравнения, называется отношением. Если отношение имеет ненулевое значение, то оно – истинно, иначе – ложно.

## 2.7. Ввод и вывод данных

В языке C/C++ нет встроенных средств ввода и вывода – он осуществляется с помощью функций, типов и объектов, которые находятся в стандартных библиотеках. Существует два основных способа: функции C и объекты C++.

Для ввода/вывода данных в стиле C используются функции, которые описываются в библиотечном файле `stdio.h`.

- `printf` (форматная строка, список аргументов);  
форматная строка – строка символов, заключенных в кавычки, которая показывает, как должны быть напечатаны аргументы. Например:  
`printf ("Значение числа Пи равно %f\n", pi);`

Форматная строка может содержать:

- символы печатаемые текстуально;
- спецификации преобразования;
- управляющие символы.

Каждому аргументу соответствует своя спецификация преобразования:

`%d, %i` – десятичное целое число;

`%f` – число с плавающей точкой;

`%e, %E` – число с плавающей точкой в экспоненциальной форме;



%u – десятичное число в беззнаковой форме;  
%c – символ;  
%s – строка.

В форматную строку также могут входить управляющие символы:

\n – управляющий символ новая строка;  
\t – табуляция;  
\a – звуковой сигнал и др.

Также в форматной строке могут использоваться модификаторы формата, которые управляют шириной поля, отводимого для размещения выводимого значения. Модификаторы – это числа, которые указывают минимальное количество позиций для вывода значения и количество позиций для вывода дробной части числа:

%[-]m[.p]C, где

– – задает выравнивание по левому краю,

m – минимальная ширина поля,

p – количество цифр после запятой для чисел с плавающей точкой и минимальное количество выводимых цифр для целых чисел (если цифр в числе меньше, чем значение p, то выводятся начальные нули),

C – спецификация формата вывода.

---

```
printf("\nСпецификации формата:\n%10.5d – целое, \n \\ %10.5f –  
с плавающей точкой\n %10.5e – \\  
в экспоненциальной форме\n%10s – строка", 10, 10.0, 10.0, "10");
```

---

Будет выведено:

Спецификации формата:

00010 – целое

10.00000 – с плавающей точкой

1.00000e+001 – в экспоненциальной форме

10 – строка.

- scanf (форматная строка, список аргументов);  
в качестве аргументов используются адреса переменных. Например:

---

```
scanf(" %d%f ", &x, &y);
```

---

При использовании библиотеки классов C++, используется библиотечный файл `iostream.h`, в котором определены стандартные потоки ввода данных от клавиатуры `cin` и вывода данных на экран `cout`, а также соответствующие операции

<< – операция записи данных в поток;

>> – операция чтения данных из потока.

---

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

```
...
```

```
cout << "\nВведите количество элементов: ";
```

```
cin >> n;
```

---

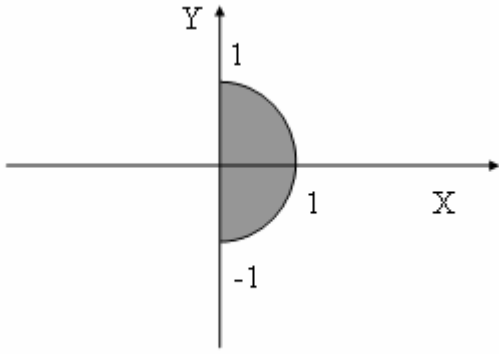
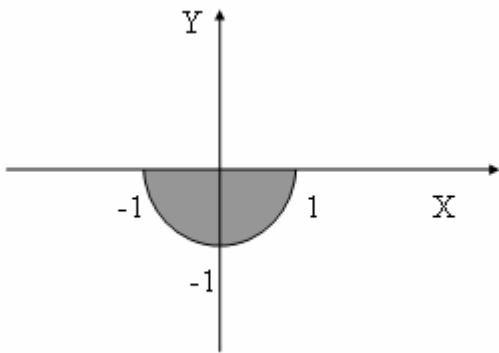
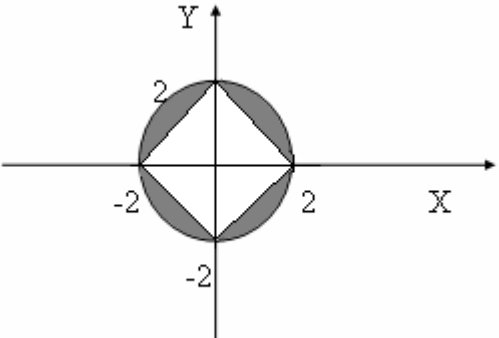
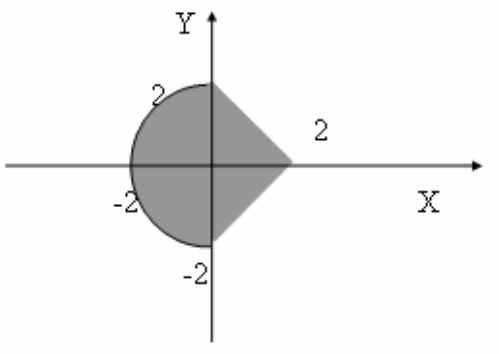
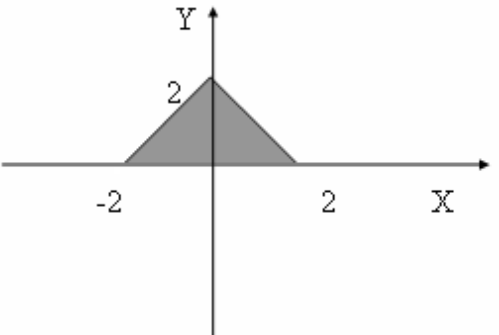
### 3. Постановка задачи

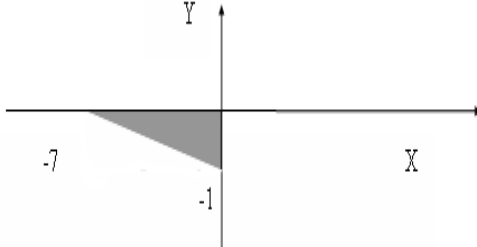
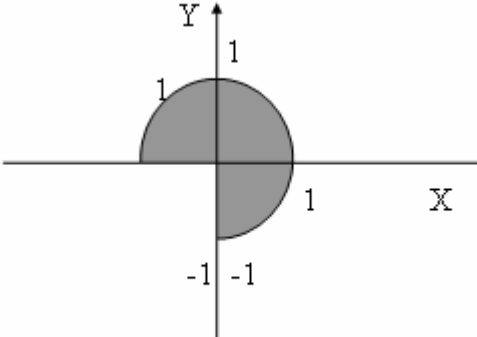
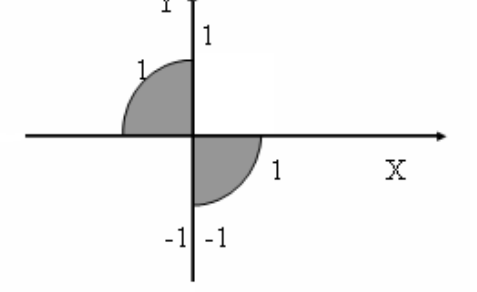
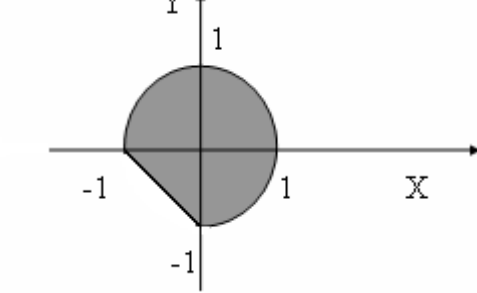
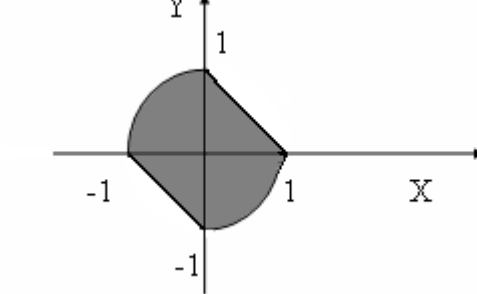
1. Для задачи 1 определить тип заданных выражений и найти их значения.

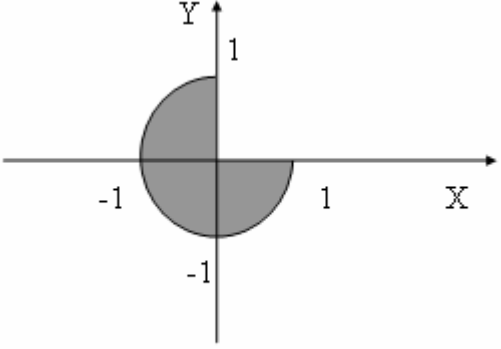
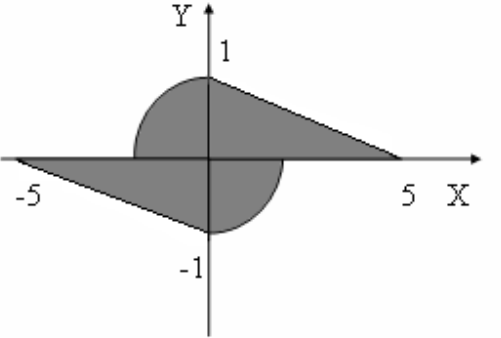
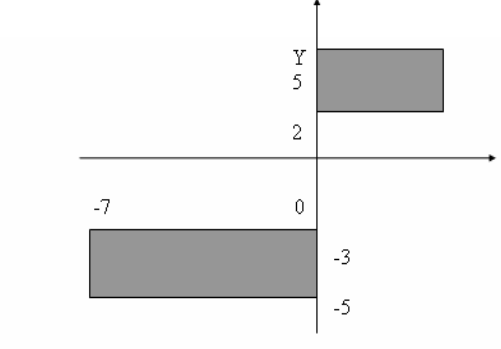
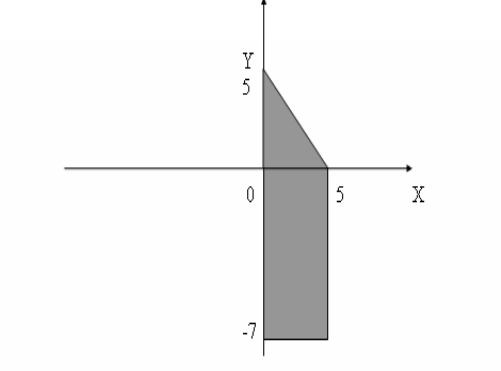
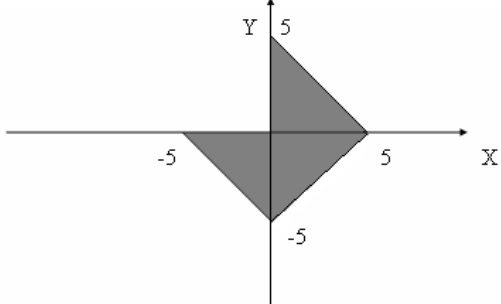
2. Составить систему тестов и вычислить полученное выражение для нескольких значений X, определить при каких X выражение не может быть вычислено.
3. Для задачи 2 записать выражение, зависящее от координат точки X1 и Y1 и принимающее значение TRUE, если точка принадлежит заштрихованной области, и FALSE, если не принадлежит.
4. Составить систему тестов и вычислить полученное выражение для нескольких точек, принадлежащих и не принадлежащих заштрихованной области.
5. Для задачи 3 вычислить значение выражения, используя различные вещественные типы данных (float и double).
6. Объяснить полученные результаты.
7. Результаты всех вычислений вывести на печать.

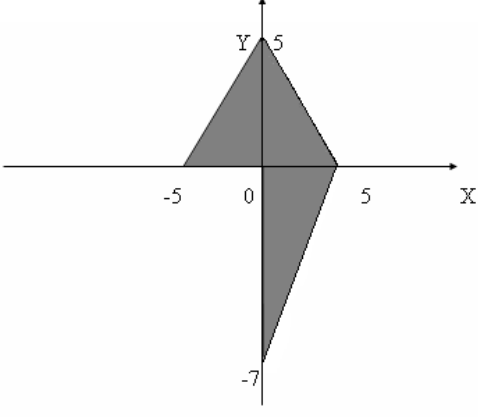
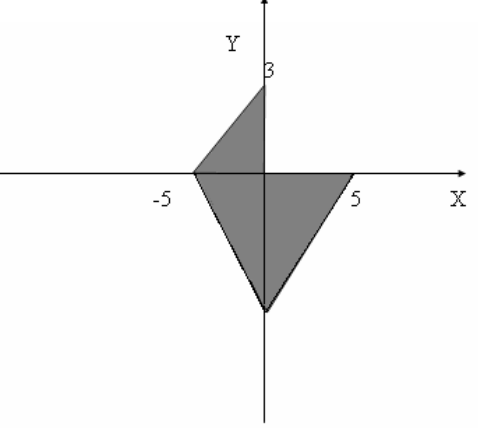
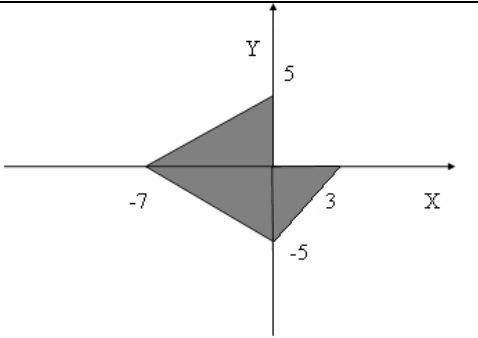
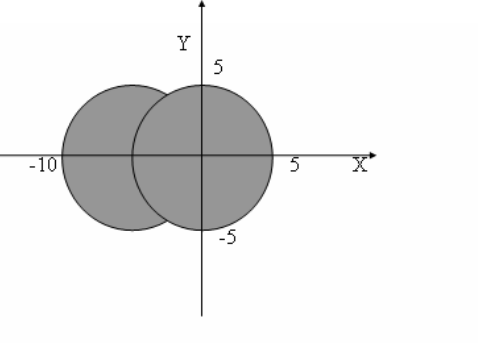
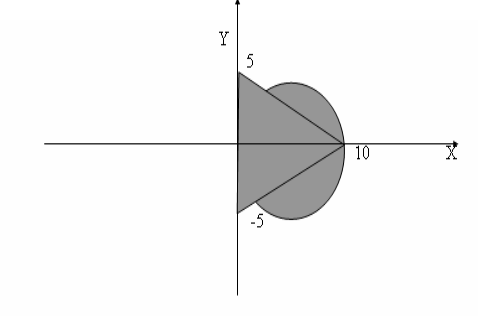
#### 4. Варианты

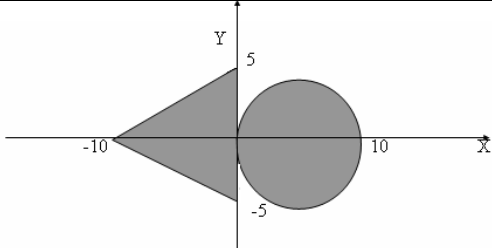
№	Задача 1	Задача 2	Задача 3
1	1) $n+++m$ 2) $m-->n$ 3) $n-->m$ 4) $\sin(x) + x^3 + \frac{1}{x^2 + 1}$		$\frac{(a+b)^2 - (a^2 + 2ab)}{b^2}$ $a=1000, b=0.0001$
2	1) $+++n*m$ 2) $m++<n$ 3) $n++>m$ 4) $x + \frac{1}{x^3 - x} - 2$		$\frac{(a-b)^2 - (a^2 - 2ab)}{b^2}$ $a=1000, b=0.0001$
3	1) $m--n$ 2) $m++<n$ 3) $n++>m$ 4) $x^4 - \cos(\arcsin(x))$		$\frac{(a+b)^3 - (a^3 + 3a^2b)}{3ab^2 + b^2}$ $a=100, b=0.001$
4	1) $n+++m$ 2) $n++<m$ 3) $--m>n$ 4) $\sqrt[3]{x - x^2 + x^5}$		$\frac{(a-b)^3 - (a^3)}{3ab^2 - b^3 - 3a^2b}$ $a=100, b=0.001$

5	1) $--m-n++$ 2) $m*m<n++$ 3) $n-->++m$ 4) $fg(x) - (5-x)^4$		$\frac{(a-b)^3 - (a^3 - 3a^2b)}{3ab^2 - b^3}$ $a=100, b=0.001$
6	1) $m-++n$ 2) $m++>--n$ 3) $m--<++n$ 4) $25x^5 - \sqrt{x^2+x}$		$\frac{(a-b)^3 - (a^3 + 3ab^2)}{-3a^2b - b^3}$ $a=100, b=0.001$
7	1) $m+--n$ 2) $m++<--n$ 3) $--m>n--$ 4) $\sqrt[5]{x^3+x^4} + \text{ctg}(\text{arctg}(x^2))$		$\frac{(a-b)^3 - (a^3)}{-b^3 + 3ab^2 - 3a^2b}$ $a=100, b=0.001$ a) $Y = \sqrt[5]{x^3+x^4} + \text{ctg}(\text{arctg}(x^2))$
8	1) $n/m++$ 2) $m++<--n$ 3) $(m/n)++<n/m$ 4) $\sqrt{ x^3-1 } - 7 \cos \sqrt[3]{x^4+x}$		$\frac{(a+b)^3 - (a^3)}{b^3 + 3ab^2 + 3a^2b}$ $a=100, b=0.001$
9	1) $m++/n--$ 2) $++m<n--$ 3) $n-->m$ 4) $\sin x^3 + x^4 + \sqrt[5]{x^2+x^3}$		$\frac{(a+b)^3 - (a^3 + 3ab^2)}{3a^2b + b^3}$ $a=100, b=0.001$

10	1) m/--n++ 2) m/n<n— 3)m+n++>n+m 4) $x^5 \sqrt{ x-1 } +  25-x^5 $		$\frac{(a+b)^3 - (a^3 + 3a^2b)}{3ab^2 + b^3}$ $a=100, b=0.001$
11	1) n+++m-- 2) n*m<n++ 3) n-->+++m 4) $2^x x \cos(x) + 1$		$\frac{(a+b)^4 - (a^4 + 4a^3b)}{6a^2b^2 + 4ab^3 + b^4}$ $a=10, b=0.01$
12	1) n+++*m 2) m--<n 3) ++m>n 4) $\sqrt[3]{x + \sqrt{ x }} +  x $		$\frac{(a+b)^4 - (a^4)}{6a^2b^2 + 4ab^3 + b^4 + 4a^3b}$ $a=10, b=0.01$
13	1) (n++/--m)++ 2) ++m<n— 3) --m>+++n 4) $\sqrt[3]{e^x + \operatorname{tg} x} + \frac{1}{x}$		$\frac{(a+b)^4 - (a^4 + 6a^2b^2 + 4ab^3)}{b^4 + 4a^3b}$ $a=10, b=0.01$
14	1) n+++*--m 2) n--<m++ 3) --n>--m 4) $\sqrt[4]{ x+1 } + \frac{1}{x^2}$		$\frac{(a+b)^4 - (a^4 + 6a^2b^2 + b^4)}{4ab^3 + 4a^3b}$ $a=10, b=0.01$

15	1) $n++/--m$ 2) $n-->n/m++$ 3) $m<n++$ 4) $1 + x \cos^2(x) + \sin^3(x)$		$\frac{(a-b)^4 - (a^4 - 4a^3b)}{6a^2b^2 - 4ab^3 + b^4}$ $a=10, b=0.01$
16	1) $m/--n++$ 2) $m/n<n--$ 3) $m+n++>n+m$ 4) $\sqrt{\sin(x) +  x^2 + x }$		$\frac{(a-b)^4 - (a^4)}{6a^2b^2 - 4ab^3 + b^4 - 4a^3b}$ $a=10, b=0.01$
17	1) $n+++m--$ 2) $n*m<n++$ 3) $n-->++m$ 4) $\arcsin(x + x^2)$		$\frac{(a-b)^4 - (a^4 + 6a^2b^2 - 4ab^3)}{b^4 - 4a^3b}$ $a=10, b=0.01$
18	1) $n+++*m$ 2) $m--<n$ 3) $++m>n$ 4) $\cos(\arctg(x))$		$\frac{(a-b)^4 - (a^4 + 6a^2b^2 + b^4)}{-4ab^3 - 4a^3b}$ $a=10, b=0.01$
19	1) $(n++/--m)++$ 2) $++m<n--$ 3) $--m>++n$ 4) $7\arctg(x^2)$		$\frac{(a+b)^2 - (a^2 + 2ab)}{b^2}$ $a=1000, b=0.0001$

20	1) $n++*--m$ 2) $n--<m++$ 3) $--n>--m$ 4) $5x^3 \sqrt[5]{\frac{1}{x^2} + \frac{1}{x^3}}$		$\frac{(a-b)^2 - (a^2 - 2ab)}{b^2}$ $a=1000, b=0.0001$
21	1) $n++/--m$ 2) $n-->n/m++$ 3) $m<n++$ 4) $\sqrt[3]{e^x - \sin x}$		$\frac{(a+b)^3 - (a^3 + 3a^2b)}{3ab^2 + b^2}$ $a=100, b=0.001$
22	1) $n++*m$ 2) $n++<m$ 3) $--m>n$ 4) $2^{-x} \sqrt{ x + \sqrt[4]{ x }} $		$\frac{(a-b)^3 - (a^3)}{3ab^2 - b^3 - 3a^2b}$ $a=100, b=0.001$
23	1) $--m-n++$ 2) $m*m<n++$ 3) $n-->++m$ 4) $1 + \frac{1}{x} + \frac{1}{x^2}$		$\frac{(a-b)^3 - (a^3 - 3a^2b)}{3ab^2 - b^3}$ $a=100, b=0.001$
24	1) $m-++n$ 2) $m++>--n$ 3) $m--<++n$ 4) $\arcsin( x+1 )$		$\frac{(a-b)^3 - (a^3 + 3ab^2)}{-3a^2b - b^3}$ $a=100, b=0.001$

25	1) m+--n 2) m++<--n 3) --m>n— 4) arccos( x + x <sup>2</sup> )		$\frac{(a-b)^3 - (a^3)}{-b^3 + 3ab^2 - 3a^2b}$ $a=100, b=0.001$
----	--	--	---

## 5. Методические указания

1. Для ввода и вывода данных использовать операции >> и << и стандартные потоки cin, cout.
2. Ввод данных для заданий А и Б организовать с клавиатуры.
3. При вычислении выражений подключить библиотеку <math.h> для вычисления функций (например, pow(x,y) для вычисления x<sup>y</sup>).
4. Вывод результатов для задания А организовать в виде:

```
n?1
n?2
n=3 n=1 m++n=3
Press any key to continue
```

5. При выполнении задания Б использовать переменную логического типа, а не условный оператор.
6. При выполнении задания В использовать вспомогательные переменные для хранения промежуточных значений. Например:  
`c=pow(a,3); d=3*pow(a,2)*b; e=3*a*pow(b,2); f=pow(b,3);`

## 6. Содержание отчета

- 1) Постановка задачи (общая и конкретного варианта).
- 2) Формулы, используемые при решении задачи (математическая модель).
- 3) Программы для решения задач на языке C++.
- 4) Описание используемых в программе стандартных функций.
- 5) Система тестов для проверки правильности работы программы и результаты выполнения тестов.
- 6) Объяснение результатов работы программы.

## Лабораторная работа №2

### Использование основных операторов языка C++

#### 1. Цель задания:

- 1) Получение практических навыков использования операторов выбора.
- 2) Получение практических навыков выбора и использования операторов циклов.

#### 2. Теоретические сведения

Операторы управления работой программы называют управляющими конструкциями программы. К ним относят:

- составные операторы;
- операторы выбора;
- операторы циклов;
- операторы перехода.

#### 2.1. Составные операторы

К составным операторам относят собственно составные операторы и блоки. В обоих случаях это последовательность операторов, заключенная в фигурные скобки. Блок отличается от составного оператора наличием определений в теле блока.

---

```
{  
n++;                //это составной оператор  
summa+=n;  
}  
  
{  
int n=0;  
n++;                //это блок  
summa+=n;  
}
```

---

#### 2.2. Операторы выбора

Операторы выбора – это условный оператор и переключатель.

1. Условный оператор имеет полную и сокращенную форму.

```
if (выражение-условие) оператор; //сокращенная форма
```

В качестве выражения-условия могут использоваться арифметическое выражение, отношение и логическое выражение. Если значение выражения-условия отлично от нуля (т. е. истинно), то выполняется оператор.

---

```
if (x<y&&x<z) min=x;  
if (выражение-условие) оператор1; //полная форма  
else оператор2;
```

---



Если значение выражения-условия отлично от нуля, то выполняется оператор1, при нулевом значении выражения-условия выполняется оператор2.

---

```
if (d>=0)
{
x1=(-b-sqrt(d))/(2*a);
x2=(-b+sqrt(d))/(2*a);
cout<< "\nx1="<<x1<<"x2="<<x2;
}
else cout<<"\nРешения нет";
```

---

## 2. Переключатель определяет множественный выбор.

```
switch (выражение)
{
case константа1 : оператор1 ;
case константа2 : оператор2 ;
. . . . .
[default: операторы;]
}
```

При выполнении оператора switch, вычисляется выражение, записанное после switch, оно должно быть целочисленным. Полученное значение последовательно сравнивается с константами, которые записаны следом за case. При первом же совпадении выполняются операторы, помеченные данной меткой. Если выполненные операторы не содержат оператора перехода, то далее выполняются операторы всех следующих вариантов, пока не появится оператор перехода или не закончится переключатель. Если значение выражения, записанного после switch, не совпало ни с одной константой, то выполняются операторы, которые следуют за меткой default. Метка default может отсутствовать.

---

```
#include <iostream.h>
void main()
{
    int i;
    cout<<"\nEnter the number";
    cin>>i;
    switch(i)
    {
    case 1:cout<<"\nthe number is one";
    case 2:cout<<"\n2*2="<<i*i;
    case 3: cout<<"\n3*3="<<i*i;break;
    case 4: cout<<"\n"<<i<<" is very beautiful!";
    default:cout<<"\nThe end of work";
    }
}
```

---

Результаты работы программы:

1. При вводе 1 будет выведено:  
The number is one  
2\*2=1  
3\*3=1

2. При вводе 2 будет выведено:  
2\*2=4  
3\*3=4
3. При вводе 3 будет выведено:  
3\*3=9
4. При вводе 4 будет выведено:  
4 is very beautiful!
5. При вводе всех остальных чисел будет выведено:  
The end of work

## 2.3. Операторы циклов

- Цикл с предусловием:

```
while (выражение-условие)  
оператор;
```

В качестве <выражения-условия> чаще всего используется отношение или логическое выражение. Если оно истинно, т. е. не равно 0, то тело цикла выполняется до тех пор, пока выражение-условие не станет ложным.

---

```
while (a!=0)  
{  
cin>>a;  
s+=a;  
}
```

---

- Цикл с постусловием:

```
do  
оператор  
while (выражение-условие);
```

Тело цикла выполняется до тех пор, пока выражение-условие истинно.

---

```
do  
{  
cin>>a;  
s+=a;  
}  
while (a!=0);
```

---

- Цикл с параметром:

```
for (выражение_1; выражение-условие; выражение_3)  
оператор;
```

выражение\_1 и выражение\_3 могут состоять из нескольких выражений, разделенных запятыми. Выражение\_1 – задает начальные условия для цикла (инициализация). Выражение-условие определяет условие выполнения цикла, если оно не равно 0, цикл выполняется, а затем вычисляется значение выражения\_3.

Выражение\_3 – задает изменение параметра цикла или других переменных (коррекция). Цикл продолжается до тех пор, пока выражение-условие не станет равно 0. Любое выражение может отсутствовать, но разделяющие их « ; » должны быть обязательно.

---

```
1.
for ( n=10; n>0; n--)// Уменьшение параметра
{
    оператор;
}
2.
for ( n=2; n>60; n+=13)// Изменение шага корректировки
{
    оператор;
}
3.
for ( num=1; num*num*num<216; num++)//проверка условия отличного
от
//того, которое налагается на число итераций
{
    оператор;
}
4.
for ( d=100.0; d<150.0;d*=1.1)//коррекция с помощью
//умножения
{
    оператор;
}
5.
for      (x=1; y<=75; y=5*(x++)+10) //коррекция      с      помощью
//арифметического выражения
{
    оператор;
}
6.
for      (x=1,   y=0;   x<10; x++; y+=x); //использование      нескольких
корректирующих выражений, тело цикла отсутствует
```

---

## 2.4. Операторы перехода

Операторы перехода выполняют безусловную передачу управления.

- `break` – оператор прерывания цикла.

---

```
{
    оператор;
    if (<выражение_условие>) break;
    оператор;
}
```

---

Т. е. оператор `break` целесообразно использовать, когда условие продолжения итераций надо проверять в середине цикла.

---

```
// Найти сумму чисел, числа вводятся с клавиатуры до тех пор,
пока не будет //введено 100 чисел или 0.
for(s=0, i=1; i<100;i++)
{
    cin>>x;
    if( x==0) break; // если ввели 0, то суммирование
заканчивается
    s+=x;
}
```

---

- `continue` – переход к следующей итерации цикла. Он используется, когда тело цикла содержит ветвления.

---

```
//Найти количество и сумму положительных чисел
for( k=0, s=0, x=1; x!=0;)
{
    cin>>x;
    if (x<=0) continue;
    k++; s+=x;
}
```

---

- `goto <метка>` – передает управление оператору, который содержит метку.  
В теле той же функции должна присутствовать конструкция:  
`<метка>:оператор;`

Метка – это обычный идентификатор, областью видимости которого является функция. Оператор `goto` передает управления оператору, стоящему после метки. Использование оператора `goto` оправдано, если необходимо выполнить переход из нескольких вложенных циклов или переключателей вниз по тексту программы или перейти в одно место функции после выполнения различных действий.

Применение `goto` нарушает принципы структурного и модульного программирования, по которым все блоки, из которых состоит программа, должны иметь только один вход и только один выход.

Нельзя передавать управление внутрь операторов `if`, `switch` и циклов. Нельзя переходить внутрь блоков, содержащих инициализацию, на операторы, которые стоят после инициализации.

- `return` – оператор возврата из функции. Он всегда завершает выполнение функции и передает управление в точку ее вызова. Вид оператора:  
`return [выражение];`

### **3. Постановка задачи**

Решить указанные в варианте задачи, используя основные операторы языка C++. При решении задачи, использовать все типы циклов (`for`, `while`, `do while`).

1. Дана последовательность из  $n$  целых чисел. Найти среднее арифметическое этой последовательности.
2. Дана последовательность из  $n$  целых чисел. Найти сумму четных элементов этой последовательности.
3. Дана последовательность из  $n$  целых чисел. Найти сумму элементов с четными номерами из этой последовательности.
4. Дана последовательность из  $n$  целых чисел. Найти сумму нечетных элементов этой последовательности.

5. Дана последовательность из  $n$  целых чисел. Найти сумму элементов с нечетными номерами из этой последовательности.
6. Дана последовательность из  $n$  целых чисел. Найти минимальный элемент в этой последовательности.
7. Дана последовательность из  $n$  целых чисел. Найти номер максимального элемента в этой последовательности.
8. Дана последовательность из  $n$  целых чисел. Найти номер минимального элемента в этой последовательности.
9. Дана последовательность из  $n$  целых чисел. Найти максимальный элемент в этой последовательности.
10. Дана последовательность из  $n$  целых чисел. Найти сумму минимального и максимального элементов в этой последовательности.
11. Дана последовательность из  $n$  целых чисел. Найти разность минимального и максимального элементов в этой последовательности.
12. Дана последовательность из  $n$  целых чисел. Найти количество нечетных элементов этой последовательности.
13. Дана последовательность из  $n$  целых чисел. Найти количество четных элементов этой последовательности.
14. Дана последовательность из  $n$  целых чисел. Найти количество элементов этой последовательности, кратных числу  $K$ .
15. Дана последовательность из  $n$  целых чисел. Найти количество элементов этой последовательности, кратных ее первому элементу.
16. Дана последовательность из  $n$  целых чисел. Найти количество элементов этой последовательности, кратных числу  $K_1$  и не кратных числу  $K_2$ .
17. Дана последовательность из  $n$  целых чисел. Определить, каких чисел в этой последовательности больше: положительных или отрицательных.
18. Дана последовательность целых чисел, за которой следует 0. Найти среднее арифметическое этой последовательности.
19. Дана последовательность целых чисел, за которой следует 0. Найти сумму четных элементов этой последовательности.
20. Дана последовательность целых чисел, за которой следует 0. Найти сумму элементов с четными номерами из этой последовательности.
21. Дана последовательность целых чисел, за которой следует 0. Найти сумму нечетных элементов этой последовательности.
22. Дана последовательность целых чисел, за которой следует 0. Найти сумму элементов с нечетными номерами из этой последовательности.
23. Дана последовательность целых чисел, за которой следует 0. Найти минимальный элемент в этой последовательности.
24. Дана последовательность целых чисел, за которой следует 0. Найти номер максимального элемента в этой последовательности.
25. Дана последовательность целых чисел, за которой следует 0. Найти номер минимального элемента в этой последовательности.
26. Дана последовательность целых чисел, за которой следует 0. Найти максимальный элемент в этой последовательности.
27. Дана последовательность целых чисел, за которой следует 0. Найти сумму минимального и максимального элементов в этой последовательности.
28. Дана последовательность целых чисел, за которой следует 0. Найти разность минимального и максимального элементов в этой последовательности.
29. Дана последовательность целых чисел, за которой следует 0. Найти количество нечетных элементов этой последовательности.
30. Дана последовательность целых чисел, за которой следует 0. Найти количество четных элементов этой последовательности.

31. Дана последовательность целых чисел, за которой следует 0. Найти количество элементов этой последовательности, кратных числу К.
32. Дана последовательность целых чисел, за которой следует 0. Найти количество элементов этой последовательности, кратных ее первому элементу.
33. Дана последовательность целых чисел, за которой следует 0. Найти количество элементов этой последовательности, кратных числу К1 и не кратных числу К2.
34. Дана последовательность целых чисел, за которой следует 0. Определить, каких чисел в этой последовательности больше: положительных или отрицательных.
35.  $S = 1 - 2 + 3 - 4 + 5 - \dots$ , всего n слагаемых;
36.  $S = 1 + 3 + 5 + 7 + \dots$ , всего n слагаемых;
37.  $S = 1 + 2 - 3 + 4 + 5 - 6 + 7 + 8 - 9 + \dots$ , всего n слагаемых;
38.  $S = 15 + 17 - 19 + 21 + 23 - 25 + \dots$ , всего n слагаемых;
39.  $S = \sin X + \sin X^2 + \sin X^3 + \sin X^4 + \dots + \sin X^n$
40.  $S = \sin X + \sin^2 X + \sin^3 X + \sin^4 X + \dots + \sin^n X$
41.  $S = \sqrt{3} + \sqrt{6} + \sqrt{9} + \dots + \sqrt{99}$
42.  $S = \sin(x + \cos(2x - \sin(3x + \cos(4x + \sin(5x - \cos(6x + \dots))))))$
43. Найти первое отрицательное число последовательности  $u = \cos(\text{ctg}(n))$ , где  $n = 1, 2, 3, \dots$
44. Определить является ли число k степенью 3.
45. Определить является ли число k простым.
46. Дана последовательность из 100 чисел. Найти номер первого отрицательного числа.
47. Найти количество цифр в десятичном числе k.
48. Найти сумму цифр в десятичном числе k.
49. Сформировать n чисел Фибоначчи ( $a_1 = 1, a_2 = 1, a_i = a_{i-1} + a_{i-2}$ ).
50. Сформировать все числа Фибоначчи не превышающие заданное число Q.
51. Дано число k. Определить, является ли оно числом Фибоначчи.
52.  $P = \frac{2}{3} \cdot \frac{4}{5} \cdot \frac{6}{7} \cdot \dots \cdot \frac{2N}{2N+1}$ .
53.  $P = a \cdot (a+1) \cdot \dots \cdot (a+n-1)$ .
54.  $S = \frac{1}{a} + \frac{1}{a^2} + \frac{1}{a^4} + \dots + \frac{1}{a^{2^{n-1}}}$ .
55.  $P = \frac{(x-1)(x-3)(x-7)\dots(x-63)}{(x-2)(x-4)(x-8)\dots(x-64)}$ .
56.  $P = (1 + \sin 0,1)(1 + \sin 0,2)\dots(1 + \sin 10)$ .
57.  $P = (1 - \frac{1}{2^2})(1 - \frac{1}{3^2}) \cdot \dots \cdot (1 - \frac{1}{n^2})$ , где  $n > 2$ .
58.  $P = (1 - \frac{1}{2})(1 - \frac{1}{4})(1 - \frac{1}{6}) \cdot \dots \cdot (1 - \frac{1}{2n})$
59.  $S = \frac{1}{3^2} + \frac{1}{5^2} + \frac{1}{7^2} + \dots + \frac{1}{(2n+1)^2}$

#### 4. Варианты

Вариант	Задача 1	Задача 2	Задача 3
1	1	34	35
2	2	33	36
3	3	32	37
4	4	31	38

5	5	30	39
6	6	29	40
7	7	28	41
8	8	27	42
9	9	26	43
10	10	25	44
11	11	24	45
12	12	23	46
13	13	22	47
14	14	21	48
15	15	20	49
16	16	19	50
17	17	18	51
18	1	23	52
19	2	24	53
20	3	25	54
21	4	26	55
22	5	27	56
23	6	28	57
24	7	29	58
25	8	30	59

### **5. Методические указания**

1. Ввод данных в задачах №1 и №2 осуществляется с клавиатуры.
2. Массивы при решении задач не используются.
3. При решении задачи №1 целесообразно использовать цикл с параметром, т. к. известно количество элементов последовательности.
4. При решении задачи №2 целесообразно использовать цикл с условием, т. к. известно, что признаком окончания последовательности является 0.

### **6. Содержание отчета**

1. Постановка задач для конкретного варианта.
2. Алгоритм решения каждой задачи в виде блок-схемы.
3. Программы для решения задач на языке C++.
4. Результаты решения.

# Лабораторная работа №4

## Работа с одномерными массивами

### 1. Цель работы:

- 1) Получение практических навыков при работе с массивами.
- 2) Получение практических навыков при работе с указателями.

### 2. Краткие теоретические сведения

Массив – это упорядоченная последовательность переменных одного типа. Каждому элементу массива отводится одна ячейка памяти. Элементы одного массива занимают последовательно расположенные ячейки памяти. Все элементы имеют одно имя – имя массива и отличаются индексами – порядковыми номерами в массиве. Количество элементов в массиве называется его размером. Чтобы отвести в памяти нужное количество ячеек для размещения массива, надо заранее знать его размер. Резервирование памяти для массива выполняется на этапе компиляции программы.

#### 2.1. Определение массива в C/C++

Массивы определяются следующим образом:

```
int a[100]; //массив из 100 элементов целого типа
```

Операция `sizeof(a)` даст результат 400, т. е. 100 элементов по 4 байта. Элементы массива всегда нумеруются с 0.

45	352	63		124	значения элементов массива индексы элементов массива
0	1	2	.....	99	

Чтобы обратиться к элементу массива, надо указать имя массива и номер элемента в массиве (индекс):

`a[0]` – индекс задается как константа,  
`a[55]` – индекс задается как константа,  
`a[i]` – индекс задается как переменная,  
`a[2*i]` – индекс задается как выражение.

Элементы массива можно задавать при его определении:

```
int a[10]={1,2,3,4,5,6,7,8,9,10};  
int a[]={1,2,3,4,5};
```

#### 2.2. Понятие указателя

Указатели являются специальными объектами в программах на C/C++. Указатели предназначены для хранения адресов памяти.

Когда компилятор обрабатывает оператор определения переменной, например, `int i=10;`, то в памяти выделяется участок памяти в соответствии с типом переменной (для `int` размер участка памяти составит 4 байта) и записывает в этот участок указанное значение. Все обращения к этой переменной компилятор заменит адресом области памяти, в которой хранится эта переменная.



Рис. 3. Значение переменной и ее адрес

Программист может определить собственные переменные для хранения адресов областей памяти. Такие переменные называются указателями. Указатель не является самостоятельным типом, он всегда связан с каким-то другим типом.

В простейшем случае объявление указателя имеет вид:

```
тип* имя;
```

Знак \*, обозначает указатель и относится к типу переменной, поэтому его рекомендуется ставить рядом с типом, а от имени переменной отделять пробелом, за исключением тех случаев, когда описываются несколько указателей. При описании нескольких указателей знак \* ставится перед именем переменной-указателя, т. к. иначе будет не понятно, что эта переменная также является указателем.

```
int* i;
double *f, *ff; //два указателя
char* c;
```

Размер указателя зависит от модели памяти. Можно определить указатель на указатель: `int** a;`

Указатель может быть константой или переменной, а также указывать на константу или переменную.

```
int i; //целая переменная
const int ci=1; //целая константа
int* pi; //указатель на целую переменную
const int* pci; //указатель на целую константу
```

Указатель можно сразу проинициализировать:

```
//указатель на целую переменную
int* pi=&i;
```

С указателями можно выполнять следующие операции:

- разыменование (\*);
- присваивание;
- арифметические операции (сложение с константой, вычитание, инкремент ++, декремент --);
- сравнение;
- приведение типов.

Операция разыменования предназначена для получения значения переменной или константы, адрес которой хранится в указателе. Если указатель указывает на переменную, то это значение можно изменять, также используя операцию разыменования.

---

```
int a; //переменная типа int
int* pa=new int; //указатель и выделение памяти под
//динамическую переменную
*pa=10;//присвоили значение динамической
//переменной, на которую указывает указатель
a=*pa;//присвоили значение переменной a
```

---

Арифметические операции применимы только к указателям одного типа.

- Инкремент увеличивает значение указателя на величину sizeof (тип) .

---

```
char* pc;
int* pi;
double* pd;
. . . . .
pc++; //значение увеличится на 1
pi++; //значение увеличится на 4
pd++; //значение увеличится на 8
```

---

- Декремент уменьшает значение указателя на величину sizeof (тип) .
- Разность двух указателей – это разность их значений, деленная на размер типа в байтах.  
Суммирование двух указателей не допускается.
- Можно суммировать указатель и константу:

### 2.3. Одномерные массивы и указатели

При определении массива ему выделяется память. После этого имя массива воспринимается как константный указатель того типа, к которому относятся элементы массива. Исключением является использование операции sizeof(имя\_массива) и операции &имя\_массива.

---

```
int a[100];

/*определение количества занимаемой массивом памяти, в нашем
случае это 4*100=400 байт*/
int k=sizeof(a);

/*вычисление количества элементов массива*/
int n=sizeof(a)/sizeof(a[0]);
```

---

Результатом операции & является адрес нулевого элемента массива:

```
имя_массива==&имя_массива=&имя_массива[0]
```

Имя массива является указателем-константой, значением которой служит адрес первого элемента массива, следовательно, к нему применимы все правила адресной арифметики, связанной с указателями. Запись имя\_массива[индекс] это выражение с двумя операндами: имя массива и индекс. Имя\_массива – это указатель-константа, а индекс определяет смещение от начала массива. Используя указатели, обращение по индексу можно записать следующим образом: \*(имя\_массива+индекс) .

---

```
for(int i=0;i<n;i++) //печать массива
cout<<*(a+i)<<" ";
```

```
/*к имени (адресу) массива добавляется константа i и  
полученное значение разыменовывается*/
```

---

Так как имя массива является константным указателем, то его невозможно изменить, следовательно, запись `* (a++)` будет ошибочной, а `* (a+1)` – нет.

Указатели можно использовать и при определении массивов:

---

```
int a[100]={1,2,3,4,5,6,7,8,9,10};  
  
//поставили указатель на уже определенный массив  
int* na=a;  
  
/*выделили в динамической памяти место под массив из 100  
элементов*/  
int b = new int[100];
```

---

## 2.4. Перебор элементов массива

- 1) Элементы массива можно обрабатывать по одному элементу, двигаясь от начала массива к его концу (или в обратном направлении):  
`for(int i=0;i<n;i++) <обработка a[i]>`
- 2) Элементы массива можно обрабатывать по два элемента, двигаясь с обеих сторон массива к его середине:  
`int i=0, j=n-1;  
while (j>i){  
<обработка a[i] и a[j]>;  
i++;j++;}`
- 3) Элементы массива можно обрабатывать по два элемента, двигаясь от начала к концу с шагом 1(т. е. обрабатываются пары элементов `a[0]` и `a[1]`, `a[1]` и `a[2]` и т. д.)  
`for (i=0;i<n-1;i++)  
<обработка a[i] и a[i+1]>`
- 4) Элементы массива можно обрабатывать по два элемента, двигаясь от начала к концу с шагом 2(т. е. обрабатываются пары элементов `a[0]` и `a[1]`, `a[2]` и `a[3]` и т. д.)  
`i=1;  
while (i<n){  
<обработка a[i] и a[i+1]>  
i:=i+2;}`

## 2.5. Классы задач по обработке массивов

- 1) К задачам 1 класса относятся задачи, в которых выполняется однотипная обработка всех или указанных элементов массива. Решение таких задач сводится к установлению того, как обрабатывается каждый элемент массива или указанные элементы, затем подбирается подходящая схема перебора, в которую вставляются операторы обработки элементов массива. Примером такой задачи является нахождение среднего арифметического элементов массива.
- 2) К задачам 2 класса относятся задачи, в которых изменяется порядок следования элементов массива. Обмен элементов внутри массива выполняется с использованием вспомогательной переменной:  
`R=a[i]; a[i]=a[j]; a[j]=R; //обмен a[i] и a[j] элементов массива.`
- 3) К задачам 3 класса относятся задачи, в которых выполняется обработка нескольких массивов или подмассивов одного массива. Массивы могут обрабатываться по одной схеме – синхронная обработка или по разным схемам – асинхронная обработка массивов.

- 4) К задачам 4 класса относятся задачи, в которых требуется отыскать первый элемент массива, совпадающий с заданным значением – поисковые задачи в массиве.

## 2.4. Сортировка массивов

Сортировка – это процесс перегруппировки заданного множества объектов в некотором установленном порядке.

### 2.4.1. Сортировка с помощью включения

Элементы массива делятся на уже готовую последовательность и исходную. При каждом шаге, начиная с  $I=2$ , из исходной последовательности извлекается  $i$ -ый элемент и вставляется на нужное место готовой последовательности, затем  $i$  увеличивается на 1 и т. д.

44	55	12	42	94	18
----	----	----	----	----	----

готовая

исходная

В процессе поиска нужного места осуществляются пересылки элементов больше выбранного на одну позицию вправо, т. е. выбранный элемент сравнивают с очередным элементом отсортированной части, начиная с  $j:=i-1$ . Если выбранный элемент больше  $a[j]$ , то его включают в отсортированную часть, в противном случае  $a[j]$  сдвигают на одну позицию, а выбранный элемент сравнивают со следующим элементом отсортированной последовательности. Процесс поиска подходящего места заканчивается при двух различных условиях:

- если найден элемент  $a[j]>a[i]$ ;
- достигнут левый конец готовой последовательности.

---

```
int i, j, x;
for (i=1; i<n; i++)
{
    x=a[i]; //запомнили элемент, который будем вставлять
    j=i-1;
    while (x<a[j] && j>=0) //поиск подходящего места
    {
        a[j+1]=a[j]; //сдвиг вправо
        j--;
    }
    a[j+1]=x; //вставка элемента
}
```

---

### 2.4.2. Сортировка методом простого выбора

Выбирается минимальный элемент массива и меняется местами с первым элементом массива. Затем процесс повторяется с оставшимися элементами и т. д.

44	55	12	42	94	18
----	----	----	----	----	----

1 мин

---

```
int i, min, n_min, j;
for (i=0; i<n-1; i++)
{
    min=a[i]; n_min=i; //поиск минимального
    for (j=i+1; j<n; j++)
        if (a[j]<min)
        {
```

---

```

        min=a[j];
        n_min=j;
    }
    a[n_min]=a[i]; //обмен
    a[i]=min;
}

```

---

### 2.4.3. Сортировка методом простого обмена

Сравниваются и меняются местами пары элементов, начиная с последнего. В результате самый маленький элемент массива оказывается самым левым элементом массива. Процесс повторяется с оставшимися элементами массива.

44	55	12	42	94	18
----	----	----	----	----	----



```

for(int i=1;i<n;i++)
for(int j=n-1;j>=i;j--)
    if(a[j]<a[j-1])
    {
        int r=a[j];
        a[j]=a[j-1];
        a[j-1]=r;}
}

```

---

### 2.5. Поиск в отсортированном массиве

В отсортированном массиве используется дихотомический (бинарный) поиск. При последовательном поиске требуется в среднем  $n/2$  сравнений, где  $n$  – количество элементов в массиве. При дихотомическом поиске требуется не более  $m$  сравнений, если  $n$  –  $m$ -ая степень 2, если  $n$  не является степенью 2, то  $n < k = 2^m$ .

Массив делится пополам  $S := (L+R) / 2 + 1$  и определяется в какой части массива находится нужный элемент  $X$ . Так как массив упорядочен, то, если  $a[S] < X$  – искомый элемент находится в правой части массива, иначе – находится в левой части. Выбранную часть массива снова надо разделить пополам и т. д., до тех пор, пока границы отрезка  $L$  и  $R$  не станут равны.

1	3	8	10	11	15	19	21	23	37
0	1	2	3	4	5	6	7	8	9

L                      S                                      R

---

```

int b;
cout<<"\nB=?";cin>>b;
int l=0,r=n-1,s;
do
{
    s=(l+r)/2; //средний элемент
    if(a[s]<b)l=s+1; //перенести левую границу
    else r=s; //перенести правую границу
}while(l!=r);
if(a[l]==b) return l;
else return -1;
...

```

---

### 3. Постановка задачи

- 1) Сформировать массив из  $n$  элементов с помощью датчика случайных чисел ( $n$  задается пользователем с клавиатуры).
- 2) Распечатать полученный массив.
- 3) Выполнить удаление указанных элементов из массива.
- 4) Вывести полученный результат.
- 5) Выполнить добавление указанных элементов в массив.
- 6) Вывести полученный результат.
- 7) Выполнить перестановку элементов в массиве.
- 8) Вывести полученный результат.
- 9) Выполнить поиск указанных в массиве элементов и подсчитать количество сравнений, необходимых для поиска нужного элемента.
- 10) Вывести полученный результат.
- 11) Выполнить сортировку массива указанным методом.
- 12) Вывести полученный результат.
- 13) Выполнить поиск указанных элементов в отсортированном массиве и подсчитать количество сравнений, необходимых для поиска нужного элемента.
- 14) Вывести полученный результат.

### 4. Варианты

Вариант	Удаление	Добавление	Перестановка	Поиск	Сортировка
1	Максимальный элемент	$K$ элементов в начало массива	Перевернуть массив	Первый четный	Простой обмен
2	Минимальный элемент	$K$ элементов в конец массива	Сдвинуть циклически на $M$ элементов вправо	Первый отрицательный	Простой выбор
3	Элемент с заданным номером	$N$ элементов, начиная с номера $K$	Сдвинуть циклически на $M$ элементов влево	Элемент с заданным ключом (значением)	Простое включение
4	$N$ элементов, начиная с номера $K$	Элемент с номером $K$	Поменять местами элементы с четными и нечетными номерами	Элемент равный среднему арифметическому элементов массива	Простой обмен
5	Все четные элементы	$K$ элементов в начало массива	Четные элементы переставить в начало массива, нечетные - в конец	Первый четный	Простой выбор
6	Все элементы с четными	$K$ элементов в	Поменять местами	Первый отрицательный	Простое включение

	индексами	конец массива	минимальный и максимальный элементы	й	
7	Все нечетные элементы	N элементов, начиная с номера K	Положительные элементы переставить в начало массива, отрицательные - в конец	Элемент с заданным ключом (значением)	Простой обмен
8	Все элементы с нечетными индексами	Элемент с номером K	Перевернуть массив	Элемент равный среднему арифметическому элементов массива	Простой выбор
9	Все элементы больше среднего арифметического элементов массива	K элементов в начало массива	Сдвинуть циклически на M элементов вправо	Первый четный	Простое включение
10	Максимальный элемент	K элементов в конец массива	Сдвинуть циклически на M элементов влево	Первый отрицательный	Простой обмен
11	Минимальный элемент	N элементов, начиная с номера K	Поменять местами элементы с четными и нечетными номерами	Элемент с заданным ключом (значением)	Простой выбор
12	Элемент с заданным номером	Элемент с номером K	Четные элементы переставить в начало массива, нечетные - в конец	Элемент равный среднему арифметическому элементов массива	Простое включение
13	N элементов, начиная с номера K	K элементов в начало массива	Поменять местами минимальный и максимальный элементы	Первый четный	Простой обмен
14	Все четные элементы	K элементов в конец массива	Положительные элементы переставить в начало массива, отрицательные - в конец	Первый отрицательный	Простой выбор

15	Все элементы с четными индексами	N элементов, начиная с номера K	Перевернуть массив	Элемент с заданным ключом (значением)	Простое включение
16	Все нечетные элементы	Элемент с номером K	Сдвинуть циклически на M элементов вправо	Элемент равный среднему арифметическому элементов массива	Простой обмен
17	Все элементы с нечетными индексами	K элементов в начало массива	Сдвинуть циклически на M элементов влево	Первый четный	Простой выбор
18	Все элементы больше среднего арифметического элементов массива	K элементов в конец массива	Поменять местами элементы с четными и нечетными номерами	Первый отрицательный	Простое включение
19	Максимальный элемент	N элементов, начиная с номера K	Четные элементы переставить в начало массива, нечетные - в конец	Элемент с заданным ключом (значением)	Простой обмен
20	Минимальный элемент	Элемент с номером K	Поменять местами минимальный и максимальный элементы	Элемент равный среднему арифметическому элементов массива	Простой выбор
21	Элемент с заданным номером	K элементов в начало массива	Положительные элементы переставить в начало массива, отрицательные - в конец	Первый четный	Простое включение
22	N элементов, начиная с номера K	K элементов в конец массива	Перевернуть массив	Первый отрицательный	Простой обмен
23	Все четные элементы	N элементов, начиная с номера K	Сдвинуть циклически на M элементов вправо	Элемент с заданным ключом (значением)	Простой выбор
24	Все элементы с четными	Элемент с номером K	Сдвинуть циклически на	Элемент равный	Простое включение



	индексами		М элементов влево	среднему арифметическому элементов массива	
25	Все нечетные элементы	К элементов в начало массива	Поменять местами элементы с четными и нечетными номерами	Первый четный	Простой обмен

## 5. Методические указания

1. При решении задач использовать псевдодинамические массивы. Псевдодинамические массивы реализуются следующим образом:

1) при определении массива выделяется достаточно большое количество памяти:  
`const int MAX_SIZE=100; //именованная константа`  
`int mas[MAX_SIZE];`

2) пользователь вводит реальное количество элементов массива меньше N.  
`int n;`

`cout<<"\nEnter the size of array<<"<<MAX_SIZE<<":";cin>>n;`

3) дальнейшая работа с массивом ограничивается заданной пользователем размерностью n.

2. Формирование массива осуществляется с помощью датчика случайных чисел. Для этого можно использовать функцию `int rand()`, которая возвращает псевдослучайное число из диапазона `0..RAND_MAX=32767`, описание функции находится в файле `<stdlib.h>`. В массиве должны быть записаны и положительные и отрицательные элементы. Например, оператор `a[I]=rand()%100-50;` формирует псевдослучайное число из диапазона `[-50;49]`.

3. Вывод результатов должен выполняться после выполнения каждого задания. Элементы массива рекомендуется выводить в строчку, разделяя их между собой пробелом.

## 6. Содержание отчета:

- 1) Постановка задачи (общая и конкретного варианта).
- 2) Анализ поставленного задания: определить к какому классу задач относится задача и объяснить почему.
- 3) Текст программы.
- 4) Результаты тестов.
- 5) Решение одной из задач с использованием указателей для доступа к элементам массива.

# Лабораторная работа №4

## Функции и массивы в C++

### 1. Цель работы:

- 1) Получение практических навыков при работе со строками, одномерными и двумерными массивами.
- 2) Получение практических навыков при работе с функциями
- 3) Получение практических навыков при передаче массивов и строк в функции.

### 2. Теоретические сведения

Функция – это именованная последовательность описаний и операторов, выполняющая законченное действие, например, формирование массива, печать массива и т. д.

Любая функция должна быть объявлена и определена.

- Объявление функции (прототип, заголовок) задает имя функции, тип возвращаемого значения и список передаваемых параметров.
- Определение функции содержит, кроме объявления, тело функции, которое представляет собой последовательность описаний и операторов.

---

```
тип имя_функции ( [список_формальных_параметров] )  
{ тело_функции }
```

---

- Тело\_функции – это блок или составной оператор. Внутри функции нельзя определить другую функцию.

В теле функции должен быть оператор, который возвращает полученное значение функции в точку вызова. Он может иметь 2 формы:

- 1) return выражение;
- 2) return;

Первая форма используется для возврата результата, поэтому выражение должно иметь тот же тип, что и тип функции в определении. Вторая форма используется, если функция не возвращает значения, т. е. имеет тип void. Программист может не использовать этот оператор в теле функции явно, компилятор добавит его автоматически в конец функции перед }.

- Тип возвращаемого значения может быть любым, кроме массива и функции, но может быть указателем на массив или функцию.
- Список формальных параметров – это те величины, которые требуется передать в функцию. Элементы списка разделяются запятыми. Для каждого параметра указывается тип и имя. В объявлении имени можно не указывать.

Для того, чтобы выполнялись операторы, записанные в теле функции, функцию необходимо вызвать. При вызове указываются: имя функции и фактические параметры. Фактические параметры заменяют формальные параметры при выполнении операторов тела функции. Фактические и формальные параметры должны совпадать по количеству и типу.

Объявление функции должно находиться в тексте раньше вызова функции, чтобы компилятор мог осуществить проверку правильности вызова. Если функция имеет тип не void, то ее вызов может быть операндом выражения.

## 2.1. Параметры функции

Основным способом обмена информацией между вызываемой и вызывающей функциями является механизм параметров. Существует два способа передачи параметров в функцию: по адресу и по значению.

- При передаче по значению выполняются следующие действия:
  - вычисляются значения выражений, стоящие на месте фактических параметров;
  - в стеке выделяется память под формальные параметры функции;
  - каждому фактическому параметру присваивается значение формального параметра, при этом проверяются соответствия типов и при необходимости выполняются их преобразования.

---

```
void Change(int a,int b)//передача по значению
{
    int r=a;
    a=b;
    b=r;
}
int main()
{
    int x=1,y=5;
    Change(x,y);
    cout<<"x="<<x<<" y="<<y; //выведется: x=1 y=5

    return 1;
}
```

---

- При передаче по адресу в стек заносятся копии адресов параметров, следовательно, у функции появляется доступ к ячейке памяти, в которой находится фактический параметр и она может его изменить.

```
void Change(int *a,int *b)//передача по адресу
{
    int r=*a;
    *a=*b;
    *b=r;
}
int main()
{
    int x=1,y=5;
    Change(&x,&y);
    cout<<"x="<<x<<" y="<<y; //выведется: x=5 y=1

    return 1;
}
```

---

Для передачи по адресу также могут использоваться ссылки. Ссылка – это синоним имени объекта, указанного при инициализации ссылки.

Формат объявления ссылки

тип & имя =имя\_объекта;

Ссылка не занимает дополнительного пространства в памяти, она является просто другим именем объекта.

При передаче по ссылке в функцию передается адрес указанного при вызове параметра, а внутри функции все обращения к параметру неявно разыменовываются.

```

void Change(int &a,int &b)
{
    int r=a;
    a=b;
    b=r;
}
int main()
{
    int x=1,y=5;
    Change(x,y);
    cout<<"x="<<x<<" y="<<y; //выведется: x=5 y=1

return 1;
}

```

---

Использование ссылок вместо указателей улучшает читаемость программы, т. к. не надо применять операцию разыменовывания. Использование ссылок вместо передачи по значению также более эффективно, т. к. не требует копирования параметров. Если требуется запретить изменение параметра внутри функции, используется модификатор const. Рекомендуется ставить const перед всеми параметрами, изменение которых в функции не предусмотрено (по заголовку будет понятно, какие параметры в ней будут изменяться, а какие нет).

## 2.2. Локальные и глобальные переменные

- Переменные, которые используются внутри данной функции, называются локальными. Память для них выделяется в стеке, поэтому после окончания работы функции они удаляются из памяти. Нельзя возвращать указатель на локальную переменную, т. к. память, выделенная такой переменной, будет освобождаться.
- Глобальные переменные – это переменные, описанные вне функций. Они видны во всех функциях, где нет локальных переменных с такими именами.

## 2.3. Передача одномерных массивов как параметров функции

При использовании массива как параметра функции, в функцию передается указатель на его первый элемент, т. е. массив всегда передается по адресу. При этом теряется информация о количестве элементов в массиве, поэтому размерность массива следует передавать как отдельный параметр.

```

void print(int a[100],int n) //вывод массива на печать
{
    for(int i=0;i<n;i++)
        cout<<a[i]<<" ";
    cout<<"\n";
}

```

---

Так как в функцию передается указатель на начало массива (передача по адресу), то массив может быть изменен за счет операторов тела функции.

## 2.4. Передача строк в качестве параметров функций

Строка в Си++ - это массив символов, заканчивающийся нуль-символом – ‘\0’ (нуль-терминатором). По положению нуль-терминатора определяется фактическая длина строки. Количество элементов в таком массиве на 1 больше, чем изображение строки.

Для работы со строками существуют специальные библиотечные функции, которые содержатся в заголовочном файле **string.h**.

Строки при передаче в функции могут передаваться как одномерные массивы типа `char` или как указатели типа `char*`. В отличие от обычных массивов в функции не указывается длина строки, т. к. в конце строки есть признак конца строки `/0`.

---

```
//Функция поиска заданного символа в строке
int find(char *s, char c)
{
for (int I=0; I<strlen(s); I++)
if(s[I]==c) return I;
return -1
}
```

---

## 2.5. Передача многомерных массивов в функцию

Многомерный массив – это массив, элементами которого служат массивы. Например, массив `int a[4][5]` – это массив из указателей `int*`, которые содержат имена одноименных массивов из 5 целых элементов:

Рис. Выделение памяти под массив, элементами которого являются массивы.

При передаче многомерных массивов в функцию все размерности должны передаваться в качестве параметров.

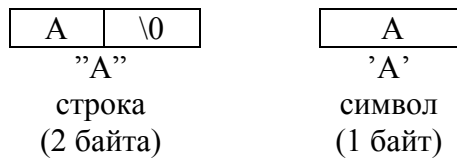
---

```
const int N=4; //глобальная переменная
void transp(int a[][N], int n) // транспонирование матрицы
{
    int r;
    for(int I=0; I<n; I++)
    for(int j=0; j<n; j++)
    if(I<j)
    {
        r[a[I][j]];
        a[I][j]=a[j][I];
        a[j][I]=r;
    }
}
```

---

## 2.6. Строки

Строка в C++ – это массив символов, заканчивающийся нуль-символом – `'\0'` (нуль-терминатором). По положению нуль-терминатора определяется фактическая длина строки. Количество элементов в таком массиве на 1 больше, чем изображение строки.



**Рис. 4. Представление строки и символа**

Присвоить значение строке с помощью оператора присваивания нельзя. Поместить строку в массив можно либо при вводе, либо с помощью инициализации.

```
char s1[10]="string1";//инициализация
char s2[]="string2";//инициализация
char s3[10];
cin>>s3;//ввод
//выделение памяти под динамическую строку
char *s4=new char[strlen(s3)+1];
strcpy(s4,s3);//копирование строки s3 в строку s4
```

Для работы со строками существуют специальные библиотечные функции, которые содержатся в заголовочном файле `string.h`.

Прототип функции	Краткое описание	Примечание
<code>unsigned <b>strlen</b>(const char* s);</code>	Вычисляет длину строки s.	
<code>int <b>strcmp</b>(const char* s1, const char* s2);</code>	Сравнивает строки s1 и s2.	Если s1<s2, то результат отрицательный, если s1==s2, то результат равен 0, если s2>s1 – результат положительный.
<code>int <b>strncmp</b>(const char* s1, const char* s2);</code>	Сравнивает первые n символов строк s1 и s2.	Если s1<s2, то результат отрицательный, если s1==s2, то результат равен 0, если s2>s1 – результат положительный.
<code>char* <b>strcpy</b>(char* s1, const char* s2);</code>	Копирует символы строки s1 в строку s2.	
<code>char* <b>strncpy</b>(char* s1, const char* s2, int n);</code>	Копирует n символов строки s1 в строку s2.	Конец строки отбрасывается или дополняется пробелами.
<code>char* <b>strcat</b>(char* s1, const char* s2);</code>	Приписывает строку s2 к строке s1	
<code>char* <b>strncat</b>(char* s1, const char* s2);</code>	Приписывает первые n символов строки s2 к строке s1	
<code>char* <b>strdup</b>(const char* s);</code>	Выделяет память и переносит в нее копию строки s	При выделении памяти используются функции

Строки при передаче в функции могут передаваться как одномерные массивы типа `char` или как указатели типа `char*`. В отличие от обычных массивов в функции не указывается длина строки, т. к. в конце строки есть признак конца строки `\0`.

### 3. Постановка задачи

1. Используя функции сформировать с помощью ДСЧ одномерный массив и вывести его на печать.
2. Выполнить обработку одномерного массива в соответствии с вариантом, используя функции, результат вывести на печать.
3. Используя функции сформировать с помощью ДСЧ двумерный массив и вывести его на печать.
4. Выполнить обработку двумерного массива в соответствии с вариантом, используя функции, результат вывести на печать.
5. Ввести с клавиатуры строку символов и обработать ее в соответствии со своим вариантом, используя функции.

### 4. Варианты

Вариант	Одномерный массив	Двумерный массив	Строки
1	Отсортировать по возрастанию только четные элементы массива.	Перевернуть все четные строки матрицы.	Удалить все гласные буквы из строки.
2	Удалить из массива все четные элементы.	Перевернуть все четные столбцы матрицы.	Подсчитать количество слов в строке.
3	Найти количество простых чисел в массиве.	Перевернуть все нечетные строки матрицы.	Перевернуть каждое четное слово в строке.
4	Найти количество чисел Фибоначчи в массиве.	Перевернуть все нечетные столбцы матрицы.	Удалить каждое четное слово из строки.
5	Удалить все простые числа из массива.	Отсортировать по убыванию все строки матрицы.	Отсортировать слова в строке в лексикографическом порядке (по алфавиту).
6	Удалить из массива все числа Фибоначчи.	Отсортировать по убыванию столбцы матрицы.	Удалить из строки все слова, начинающиеся на гласную букву.
7	Отсортировать по возрастанию только положительные элементы массива.	Меня местами строки матрицы, отсортировать по возрастанию ее первый столбец.	Удалить из строки все слова, заканчивающиеся на гласную букву.
8	Удалить из массива все элементы с четными номерами.	Меня местами столбцы матрицы, отсортировать по возрастанию ее первую строку.	Удалить все гласные буквы из строки.

9	Отсортировать по возрастанию только те элементы массива, которые являются простыми числами.	Все четные строки матрицы сдвинуть циклически на $K$ элементов вправо.	Подсчитать количество слов в строке.
10	Удалить из массива все элементы равные $\min(a[1], a[3], \dots, a[2n-1])$ .	Все нечетные строки матрицы сдвинуть циклически на $K$ элементов влево.	Перевернуть каждое четное слово в строке.
11	Создать новый массив из номеров элементов, значения которых равны 0.	Перевернуть все четные строки матрицы.	Удалить каждое четное слово из строки.
12	Сформировать массив, в котором будут только элементы исходного массива, заканчивающиеся на цифру $K$ .	Перевернуть все четные столбцы матрицы.	Отсортировать слова в строке в лексикографическом порядке (по алфавиту).
13	Отсортировать по возрастанию только четные элементы массива.	Перевернуть все нечетные строки матрицы.	Удалить из строки все слова, начинающиеся на гласную букву.
14	Удалить из массива все четные элементы.	Перевернуть все нечетные столбцы матрицы.	Удалить из строки все слова, заканчивающиеся на гласную букву.
15	Найти количество простых чисел в массиве.	Отсортировать по убыванию все строки матрицы.	Удалить все гласные буквы из строки.
16	Найти количество чисел Фибоначчи в массиве.	Отсортировать по убыванию все столбцы матрицы.	Подсчитать количество слов в строке.
17	Удалить все простые числа из массива.	Меня местами строки матрицы, отсортировать по возрастанию ее первый столбец.	Перевернуть каждое четное слово в строке.
18	Удалить из массива все числа Фибоначчи.	Меня местами столбцы матрицы, отсортировать по возрастанию ее первую строку.	Удалить каждое четное слово из строки.
19	Отсортировать по возрастанию только положительные элементы массива.	Все четные строки матрицы сдвинуть циклически на $K$ элементов вправо.	Отсортировать слова в строке в лексикографическом порядке (по алфавиту).



20	Удалить из массива все элементы с четными номерами.	Все нечетные строки матрицы сдвинуть циклически на K элементов влево.	Удалить из строки все слова, начинающиеся на гласную букву.
21	Отсортировать по возрастанию только те элементы массива, которые являются простыми числами.	Перевернуть все четные строки матрицы.	Удалить из строки все слова, заканчивающиеся на гласную букву.
22	Удалить из массива все элементы равные $\min(a[1], a[3], \dots, a[2n-1])$ .	Перевернуть все четные столбцы матрицы.	Удалить все гласные буквы из строки.
23	Создать новый массив из номеров элементов, значения которых равны 0.	Перевернуть все нечетные строки матрицы.	Подсчитать количество слов в строке.
24	Сформировать массив, в котором будут только элементы исходного массива, заканчивающиеся на цифру K.	Перевернуть все нечетные столбцы матрицы.	Перевернуть каждое четное слово в строке.
25	Сформировать два массива. В первый массив включить элементы из исходного массива с четными номерами, а во второй с нечетными.	Отсортировать по убыванию все строки матрицы.	Удалить каждое четное слово из строки.

## 5. Методические указания

1. Формирование, печать и обработку массивов и строк оформить в виде функции. Массивы передавать как параметры функций.
2. Реализовать массивы как псевдинамические, их размерности передавать как параметры функций.
3. Формирование массивов выполнить с использованием ДСЧ. В массивы записывать и положительные, и отрицательные числа.
4. Ввод/вывод строк организовать с помощью функций:
  - `char* gets(char*s)`
  - `int puts(char *s)`
5. Для обработки строк использовать стандартные функции из библиотечного файла `<string.h>`
6. Сортировку массивов организовать с помощью одного из простых методов сортировки, рассмотренных в лабораторной работе №3.

7. Функция `main()` должна содержать только описание массивов/строк и вызовы функций для формирования, печати и обработки массивов/строк.

## **6. Содержание отчета**

1. Постановка задачи (общая и для конкретного варианта).
2. Определения функций, используемых для формирования, печати и обработки массивов/строк (для каждой задачи).
3. Определение функции `main()`.
4. Результаты тестов.

# Лабораторная работа №5

## Динамические массивы

### 1. Цель работы:

1. Получить практические навыки выделения, перераспределения и освобождения памяти при работе с динамическими массивами

### 2. Теоретические сведения

Для работы с динамической памятью используют указатели. С их помощью осуществляется доступ к участкам динамической памяти, которые называются динамическими переменными. Динамические переменные создаются с помощью специальных функций и операций. Они существуют либо до конца работы программ, либо до тех пор, пока не будут уничтожены с помощью специальных функций или операций.

Для создания динамических переменных используют операцию `new`, определенную в C++:

```
указатель = new имя_типа [инициализатор];  
где инициализатор – выражение в круглых скобках.
```

Операция `new` позволяет выделить и сделать доступным участок динамической памяти, который соответствует заданному типу данных. Если задан инициализатор, то в этот участок будет занесено значение, указанное в инициализаторе.

```
int* x=new int(5);
```

Для удаления динамических переменных используется операция `delete`, определенная в C++:

```
delete указатель;
```

где указатель содержит адрес участка памяти, ранее выделенный с помощью операции `new`.

```
delete x;
```

Операция `new` при использовании с массивами имеет следующий формат:

```
new тип_массива
```

Такая операция выделяет для размещения массива участок динамической памяти соответствующего размера, но не позволяет инициализировать элементы массива. Операция `new` возвращает указатель, значением которого служит адрес первого элемента массива. При выделении динамической памяти размеры массива должны быть полностью определены.

---

```
//выделение динамической памяти 100*sizeof(int) байт  
int* a = new int[100];
```

---

При формировании матрицы сначала выделяется память для массива указателей на одномерные массивы, а затем в цикле с параметром выделяется память под `n` одномерных массивов.

---

```
/*выделение динамической памяти под двумерный динамический массив*/  
int** form_matr(int n,int m)  
{  
int **matr=new int*[n]; //выделение памяти по массив указателей  
for(int i=0;i<n;i++)  
//выделение памяти 100*sizeof(int) байт для массива значений  
matr[i]=new int [m];
```

```
return matr;//возвращаем указатель на массив указателей
}
```

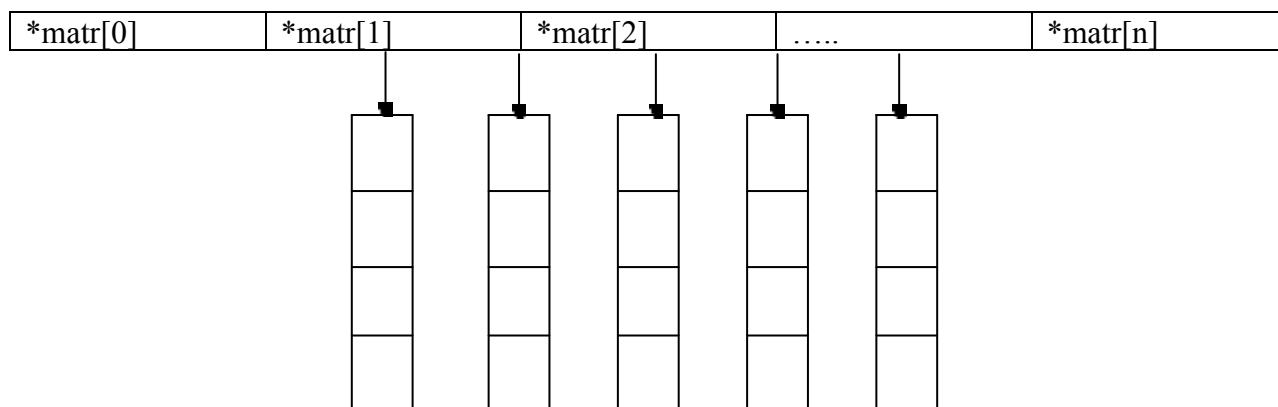


Рис. Выделение памяти под двумерный массив

Изменять значение указателя на динамический массив надо аккуратно, т. к. этот указатель затем используется при освобождении памяти с помощью операции delete.

```
/*освобождает память, выделенную под массив, если а адресует его начало*/
delete[] a;
```

Удаление из динамической памяти двумерного массива осуществляется в порядке обратном его созданию, т. е. сначала освобождается память, выделенная под одномерные массивы с данными, а затем память, выделенная под одномерный массив указателей.

```
int find(int **matr,int m,int I)
{
    for(int i=0;i<m;i++)
        if(matr[I][i]<0) return 1;
    return 0;
}
```

При удалении из динамической матрицы строк или столбцов создается новая матрица нужного размера, в которую переписываются данные из старой матрицы. Затем старая матрица удаляется.

```
int **del(int **matr,int &n,int m)
{//удаление четных строк
    int k=0,t=0;
    for(int i=0;i<n;i++)
        if(i % 2!=0)k++;{//количество нечетных строк
    //выделяем память под новую матрицу
    int **matr2=form_matr(k,m);
    for(i=0;i<n;i++)
        if(i % 2!=0)
        {
    //если строка нечетная, то переписываем ее в новую матрицу
        for(int j=0;j<m;j++)
            matr2[t][j]=matr[i][j];
            t++;
        }
}
```

```

        n=t;//изменяем количество строк
//возвращаем указатель на новую матрицу как результат функции
return matr2;
}

```

### 3. Постановка задачи

1. Сформировать динамический одномерный массив, заполнить его случайными числами и вывести на печать.
2. Выполнить указанное в варианте задание и вывести полученный массив на печать.
3. Сформировать динамический двумерный массив, заполнить его случайными числами и вывести на печать.
4. Выполнить указанное в варианте задание и вывести полученный массив на печать.

### 4. Варианты

№ варианта	Одномерный массив	Двумерный массив
1	Удалить первый четный элемент	Добавить строку с заданным номером
2	Удалить первый отрицательный элемент	Добавить столбец с заданным номером
3	Удалить элемент с заданным ключом (значением)	Добавить строку в конец матрицы
4	Удалить элемент равный среднему арифметическому элементов массива	Добавить столбец в конец матрицы
5	Удалить элемент с заданным номером	Добавить строку в начало матрицы
6	Удалить N элементов, начиная с номера K	Добавить столбец в начало матрицы
7	Удалить все четные элементы	Добавить K строк в конец матрицы
8	Удалить все элементы с четными индексами	Добавить K столбцов в конец матрицы
9	Удалить все нечетные элементы	Добавить K строк в начало матрицы
10	Удалить все элементы с нечетными индексами	Добавить K столбцов в начало матрицы
11	Добавить элемент в начало массива	Удалить строку с номером K
12	Добавить элемент в конец массива	Удалить столбец с номером K
13	Добавить K элементов в начало массива	Удалить строки, начиная со строки K1 и до строки K2
14	Добавить K элементов в конец массива	Удалить столбцы, начиная со столбца K1 и до столбца K2
15	Добавить K элементов, начиная с номера N	Удалить все четные строки
16	Добавить после каждого отрицательного элемента его модуль	Удалить все четные столбцы
17	Добавить после каждого четного элемента элемент со значением 0	Удалить все строки, в которых есть хотя бы один нулевой элемент
18	Добавить по K элементов в начало и в конец массива	Удалить все столбцы, в которых есть хотя бы один нулевой элемент

19	Добавить элемент с номером K	Удалить строку, в которой находится наибольший элемент матрицы
20	Удалить элемент с заданным номером	Добавить строки после каждой четной строки матрицы
21	Удалить N элементов, начиная с номера K	Добавить столбцы после каждого четного столбца матрицы
22	Удалить все четные элементы	Добавить K строк, начиная со строки с номером N
23	Удалить все элементы с четными индексами	Добавить K столбцов, начиная со столбца с номером N
24	Удалить все нечетные элементы	Добавить строку после строки, содержащей наибольший элемент
25	Удалить все элементы с нечетными индексами	Добавить столбец после столбца, содержащего наибольший элемент

## 5. Методические указания

1. Для выделения памяти под массивы использовать операцию new, для удаления массивов из памяти – операцию delete.
2. Для выделения памяти, заполнения массивов, удаления и добавления элементов (строк, столбцов) написать отдельные функции. В функции main() должны быть размещены только описания переменных и обращения к соответствующим функциям:

```
int main()
{
    int n;
    cout<<"N?";cin>>n;
    person*mas=form_mas(n);
    init_mas(mas,n);
    print_mas(mas,n);
    return 1;
}
```

3. Для реализации интерфейса использовать текстовое меню:

```
....
do
{
cout<<"1. Формирование массива\n";
cout<<"2. Печать массива\n";
cout<<"3. Удаление из массива\n";
cout<<"4. Добавление в массив\n";
cout<<"5. Выход\n";
cin>>k;
switch (k)
{
case 1: mas=form_mas(SIZE);input_mas(mas,SIZE); break;//выделение памяти и заполнение
case 2: print_mas(mas,SIZE); break;//печать
case 3: del_mas(mas,SIZE);break;//удаление
case 4: add_mas(mas,SIZE);break;//добавление
}
while (k!=5);//выход
```

4. При удалении элементов (строк, столбцов) предусмотреть ошибочные ситуации, т. е. ситуации, в которых будет выполняться попытка удаления элемента (строки, столбца) из пустого массива или количество удаляемых элементов будет превышать количество

имеющихся элементов (строк, столбцов). В этом случае должно быть выведено сообщение об ошибке.

## 6. Содержание отчета

1. Постановка задачи (общая и для конкретного варианта).
2. Определения функций для реализации поставленных задач.
3. Определение функции main().
4. Тесты

# Лабораторная работа №6

## Массивы структур и массивы строк

### 1. Цель работы:

1. Получить практические навыки работы с динамическими строковыми данными.
2. Получить практические навыки работы со структурами.
3. Получить практические навыки организации динамических массивов с элементами сложной структуры.

### 2. Теоретические сведения

#### 2.1. Структуры

Структура – это объединенное в единое целое множество поименованных элементов данных. Элементы структуры (поля) могут быть различного типа, они все должны иметь различные имена.

---

```
struct Date          //определение структуры
{
    int day;
    int month;
    int year;
};
```

```
Date birthday;      //переменная типа Date
```

---

Для переменных одного и того же структурного типа определена операция присваивания. При этом происходит поэлементное копирование.

Доступ к элементам структур обеспечивается с помощью уточненных имен:  
имя\_структуры.имя\_элемента

---

```
//присваивание значений полям переменной birthday
birthday.day=11; birthday.month=3; birthday.year=1993;
Date Data;
// присваивание значения переменной birthday переменной Data
Data=birthday;
```

---

Из элементов структурного типа можно организовывать массивы также как из элементов стандартных типов.

---

```
Date mas[15]; //массив структур
```

```
//ввод значений массива
for(int i=0;i<15;i++)
```

```

{
    cout<<"\nEnter day:";cin>>mas[i].day;
    cout<<"\nEnter month:";cin>>mas[i].month;
    cout<<"\nEnter year:";cin>>mas[i].year;
}

```

---

### 3. Постановка задачи

1. Сформировать динамический массив из элементов структурного типа. Структурный тип определен в варианте.
2. Распечатать сформированный массив.
3. Выполнить поиск элементов в массиве, удовлетворяющих заданному в варианте условию и сформировать из них новый массив.
4. Распечатать полученный массив.
5. Сформировать динамический массив, состоящий из динамических строк.
6. Распечатать сформированный массив.
7. Выполнить обработку этого массива.
8. Распечатать полученный массив.

### 4 Варианты

№ варианта	Структура	Критерий для поиска в массиве структур	Задание для обработки массива строк
1	<pre> struct person {     char*name;     char *adres;     int age; }; </pre>	Имена начинаются на букву 'А'	Добавить строку с заданным номером
2	<pre> struct date {     int day;     char*month;     int year; }; </pre>	Даты с летними месяцами	Удалить строку с заданным номером
3	<pre> struct student {     char*name;     int kurs;     float rating }; </pre>	Студенты первого курса	Добавить строку в конец массива
4	<pre> struct employee {     char*name;     float salary;     int stage }; </pre>	Сотрудники со стажем больше 10 лет	Удалить строку из конца матрицы
5	<pre> struct pupil {     char*name;     int age;     float rating }; </pre>	Ученики со средним баллом больше 4	Добавить строку в начало массива



	};		
6	struct person { char*name; int age; };	Возраст больше 25 лет	Удалить строку из начала массива
7	struct date { int day; char*month; int year; };	Даты после 2000 года	Добавить К строк в конец массива
8	struct student { char*name; int kurs; float rating };	Студенты, у которых рейтинг меньше 3	Удалить К строк из конца матрицы
9	struct employee { char*name; float salary; int stage };	Сотрудники, у которых имя начинается на букву 'Л'	Добавить К строк в начало массива
10	struct pupil { char*name; int age; float rating };	Ученики, у которых фамилия "Иванов"	Удалить К строк из начала массива
11	struct person { char*name; int age; };	Возраст меньше 18	Удалить строку с номером К
12	struct date { int day; char*month; int year; };	Дата принадлежит первой декаде месяца	Добавить строку с номером К
13	struct student { char*name; int kurs; float rating };	Студены пятого курса	Удалить строки, начиная со строки К1 и до строки К2
14	struct employee { char*name; float salary; int stage };	Сотрудники со стажем меньше 3 лет	Добавить строки, начиная со строки К1 и до строки К2

	};		
15	struct pupil { char*name; int age; float rating };	Ученики со средним баллом равным 4.5	Удалить все строки, которые начинаются на букву 'F'
16	struct person { char*name; int age; };	Имена начинаются на букву 'A'	Удалить все четные строки
17	struct date { int day; char*month; int year; };	Даты с зимними месяцами	Удалить все строки, в которых есть хотя бы одна цифра
18	struct student { char*name; int kurs; float rating };	Студенты первого курса у которых рейтинг меньше 3	Удалить все столбцы, в которых есть хотя бы одна буква 'A'
19	struct employee { char*name; float salary; int stage };	Сотрудники со стажем больше 10 лет и заработной платой больше 15000	Удалить самую длинную строку массива
20	struct pupil { char*name; int age; float rating };	Ученики 13 лет со средним баллом больше 4	Добавить строки после каждой четной строки массива
21	struct person { char*name; int age; };	Возраст больше 25 лет и фамилия начинается на букву 'C'	Удалить каждую нечетную строку массива
22	struct date { int day; char*month; int year; };	Зимние даты после 2000 года	Добавить K строк, начиная со строки с номером N
23	struct student { char*name; int kurs; float rating };	Студенты 1 и 2 курса, у которых рейтинг меньше 3	Удалить K строк, начиная со строки с номером N

	};		
24	struct employee { char*name; float salary; int stage };	Сотрудники, у которых имя начинается на букву 'Л' и заработная плата меньше 6000	Добавить строку после самой длинной строки массива
25	struct pupil { char*name; int age; float rating };	Ученики, у которых фамилия "Иванов" и рейтинг больше 4	Добавить строку после самой короткой строки массива

## 5. Методические указания

1. Для выделения памяти под массивы использовать операцию new, для удаления массивов из памяти – операцию delete.
2. Для формирования и печати структур написать отдельные функции:

```

person make_person()
{
    int Age; char Name[20];
    cout<<"Name?";
    cin>>Name;
    cout<<"Age?";
    cin>>Age;
    person p;
    p.name=new char[strlen(Name)+1];
    strcpy(p.name,Name);
    p.age=Age;
    return p;
}
void print_person(person p)
{
    cout<<"\nName:  "<<p.name<<"\t"<<"Age:  "<<p.age;
}

```

3. Для выделения памяти, заполнения массивов, поиска заданных элементов написать отдельные функции. В функции main() должны быть размещены только описания переменных и обращения к соответствующим функциям.
5. Если в массиве отсутствуют элементы, соответствующие критерию поиска, то должно быть выведено сообщение о том, что требуемые элементы не найдены.
6. При удалении строк предусмотреть ситуации, в которых будет выполняться попытка удаления строки из пустого массива или количество удаляемых элементов будет превышать количество имеющихся строк. В этом случае должно быть выведено сообщение об ошибке.

## 6. Содержание отчета

1. Постановка задачи (общая и для конкретного варианта).
2. Определения функций для реализации поставленных задач.
3. Определение функции main().
4. Тесты

# Лабораторная работа №7

## Функции в C++

### 1. Цель работы:

- 1) Получить практические навыки работы с функциями;
- 2) получить практические навыки работы с шаблонами функций;
- 3) получить практические навыки работы с указателями функций.

### 2. Теоретические сведения

#### 2.1. Функции с начальными значениями параметров (по-умолчанию)

В определении функции может содержаться начальное (умалчиваемое) значение параметра. Это значение используется, если при вызове функции соответствующий параметр опущен. Все параметры, описанные справа от такого параметра, также должны быть умалчиваемыми.

---

```
const int N=20;//количество элементов массива
char mas1[N][10]);//массив имен
int mas2[N]//массив возрастов
void init(int i, char* name="Вася ", int age=17)
{
strcpy(mas1[i],name);
mas2[i]=age;
}
```

---

Примеры использования функции `init()`:

```
1. for (int i=1;i<N;i++)
    init(i) ;
```

Всем элементам массива `mas1` присваивается значение «Вася», всем элементам массива `mas2` присваивается значение 17.

```
2. for (int i=1;i<N;i++)
    {
    char Name[10];
    cout<<"Введите имя:"; cin>>Name;
    init(i,Name) ;
    }
```

Всем элементам массива `mas1` присваивается значение переменной `Name`, всем элементам массива `mas2` присваивается значение 17.

#### 2.2. Функции с переменным числом параметров

В C++ допустимы функции, у которых при компиляции не фиксируется число параметров, и, кроме того, может быть неизвестен тип этих параметров. Количество и тип параметров становится известным только в момент вызова, когда явно задан список фактических параметров. Каждая функция с переменным числом параметров должна иметь хотя бы один обязательный параметр. Определение функции с переменным числом параметров:

тип имя (явные параметры, . . . )

```
{
    тело функции
}
```

После списка обязательных параметров ставится запятая, а затем многоточие, которое показывает, что дальнейший контроль соответствия количества и типов параметров при обработке вызова функции производить не нужно. При обращении к функции все параметры и обязательные, и необязательные будут размещаться в памяти друг за другом. Следовательно, определив адрес обязательного параметра как  $p=&k$ , где  $p$  – указатель, а  $k$  – обязательный параметр, можно получить адреса и всех остальных параметров: оператор  $k++$ ; выполняет переход к следующему параметру списка. Еще одна сложность заключается в определении конца списка параметров, поэтому каждая функция с переменным числом параметров должна иметь механизм определения количества и типов параметров. Существует два подхода:

- 1) известно количество параметров, которое передается как обязательный параметр;
- 2) известен признак конца списка параметров.

---

```
//Найти среднее арифметическое последовательности
//чисел, если известно количество чисел
#include <iostream.h>
float sum(int k, . . .)
//явный параметр k задает количество чисел
{
    int *p=&k;//настроили указатель на параметр k
    int s=0;
    for(;k!=0;k--)
        s+=*(++p);
    return s/k;
}
void main()
{
    //среднее арифметическое 4+6
    cout<<"\n4+6="<<sum(2,4,6);
    //среднее арифметическое 1+2+3+4
    cout<<"\n1+2++3+4="<<sum(4,1,2,3,4);
}
```

---

Для доступа к списку параметров используется указатель  $*p$  типа `int`. Он устанавливается на начало списка параметров в памяти, а затем перемещается по адресам фактических параметров ( $++p$ ).

---

```
/*Найти среднее арифметическое последовательности чисел, если
известен признак конца списка параметров */
#include<iostream.h>
int sum(int k, ...)
{
    int *p = &k; //настроили указатель на параметр k
    int s = *p; //значение первого параметра присвоили s
    for(int i=1;p!=0;i++) //пока нет конца списка
        s += *(++p);
    return s/(i-1);
}
```

```

void main()
{
    //находит среднее арифметическое 4+6
    cout<<"\n4+6="<<sum(4,6,0);
    //находит среднее арифметическое 1+2+3+4
    cout<<"\n1+2++3+4="<<sum(1,2,3,4,0);
}

```

---

### 2.3. Перегрузка функций

Цель перегрузки состоит в том, чтобы функция с одним именем по-разному выполнялась и возвращала разные значения при обращении к ней с различными типами и различным числом фактических параметров. Для обеспечения перегрузки необходимо для каждой перегруженной функции определить возвращаемые значения и передаваемые параметры так, чтобы каждая перегруженная функция отличалась от другой функции с тем же именем. Компилятор определяет, какую функцию выбрать по типу фактических параметров.

---

```

#include <iostream.h>
#include <string.h>

//сравнение двух целых чисел
int max(int a, int b)
{
    if (a>b) return a;
    else return b;
}

//сравнение двух вещественных чисел
float max(float a, float b)
{
    if(a>b) return a;
    else return b;
}

//сравнение двух строк
char* max(char* a, char* b)
{
    if (strcmp(a,b)>0) return a;
    else return b;
}

void main()
{
    int a1,b1;
    float a2, b2;
    char s1[20];
    char s2[20];

    cout<<"\nfor int:\n";
    cout<<"a=?";cin>>a1;
    cout<<"b=?";cin>>b1;
    cout<<"\nMAX="<<max(a1,b1)<<"\n";
}

```

```

cout<<"\nfor float:\n";
cout<<"a=";<<cin>>a2;
cout<<"b=";<<cin>>b2;
cout<<"\nMAX="<<max(a2,b2)<<"\n";

cout<<"\nfor char*:\n";
cout<<"a=";<<cin>>s1;
cout<<"b=";<<cin>>s2;
cout<<"\nMAX="<<max(s1,s2)<<"\n";
}

```

---

Правила описания перегруженных функций:

- Перегруженные функции должны находиться в одной области видимости.
- Перегруженные функции могут иметь параметры по умолчанию, при этом значения одного и того же параметра в разных функциях должны совпадать. В разных вариантах перегруженных функций может быть разное количество умалчиваемых параметров.
- Функции не могут быть перегружены, если описание их параметров отличается только модификатором `const` или наличием ссылки: функции `int& f1(int&, const int&){...}` и `int f1(int, int){...}` – не являются перегруженными, т. к. компилятор не сможет узнать какая из функций вызывается, потому что нет синтаксических отличий между вызовом функции, которая передает параметр по значению и функции, которая передает параметр по ссылке.

### 2.3. Шаблоны функций

Шаблоны вводятся для того, чтобы автоматизировать создание функций, обрабатывающих разнотипные данные. Например, алгоритм сортировки можно использовать для массивов различных типов. При перегрузке функции для каждого используемого типа определяется своя функция. Шаблон функции определяется один раз, но определение параметризуется, т. е. тип данных передается как параметр шаблона.

```

template <class имя_типа [,class имя_типа]>
заголовок_функции
{
    тело функции
}

```

Таким образом, шаблон семейства функций состоит из 2 частей – заголовка шаблона: `template<список параметров шаблона>` и обыкновенного определения функции, в котором вместо типа возвращаемого значения и/или типа параметров, записывается имя типа, определенное в заголовке шаблона.

---

```

//сравнение двух чисел любого типа
template<class T>
T max(T a, T b)
{
    if (a>b) return a;
    else return b;
}

```

---

Шаблон служит для автоматического формирования конкретных описаний функций по тем вызовам, которые компилятор обнаруживает в программе. Например, если в программе вызов функции осуществляется как `max(1,5)`, то компилятор сформирует определение функции `int max(int a, int b){...}`.

## 2.4. Указатель на функцию

Каждая функция характеризуется типом возвращаемого значения, именем и списком типов ее параметров. Если имя функции использовать без последующих скобок и параметров, то он будет выступать в качестве указателя на эту функцию, и его значением будет выступать адрес размещения функции в памяти. Это значение можно будет присвоить другому указателю. Тогда этот новый указатель можно будет использовать для вызова функции.

Указатель на функцию определяется следующим образом:

тип\_функции (\*имя\_указателя) (спецификация параметров)

В определении указателя количество и тип параметров должны совпадать с соответствующими типами в определении функции, на которую ставится указатель.

Вызов функции с помощью указателя имеет вид:

(\*имя\_указателя) (список фактических параметров);

---

```
#include <iostream.h>
int f1(char* S)
{
    cout<<S<<"\n";
    return 1;
}

void main()
{
    char s[20]="\nfunction1";
    int(*ptr1)(char*); //указатель на функцию
    ptr1=f1;          //указателю присваивается адрес функции f1
    cout<<((*ptr1)(s)); //вызов функции f1 с помощью указателя
}
```

---

Указатели на функции удобно использовать в тех случаях, когда функцию надо передать в другую функцию как параметр.

---

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

typedef float(*fptr)(float); //тип-указатель на функцию уравнения

/*решение уравнения методом половинного деления, уравнение
передается с помощью указателя на функцию */
float root(fptr f, float a, float b, float e)
{
    float x;
    do
    {
        x=(a+b)/2; //находим середину отрезка
        if ((*f)(a)*f(x)<0) //выбираем отрезок
            b=x;
        else a=x;
    }
}
```



```

    }
    while ((*f) (x) > e && fabs(a-b) > e);
    return x;
}

//функция, для которой ищется корень
float testf(float x)
{
    return x*x-1;
}

void main()
{
    /*в функцию root передается указатель на функцию, координаты
отрезка и точность */
    float res=root(testf,0,2,0.0001);
    cout<<"\nX="<<res;
}

```

---

## 2.5. Численные методы решения уравнений

Довольно часто на практике приходится решать уравнения вида:

$$F(x) = 0, \quad (1),$$

где функция  $F(x)$  определена и непрерывна на некотором конечном или бесконечном интервале  $\alpha < x < \beta$ .

Всякое значение  $\bar{x}$  такое, что  $F(\bar{x}) \equiv 0$ , называется корнем уравнения, а нахождение этого значения и есть решение уравнения.

На практике в большинстве случаев найти точное решение возникшей математической задачи не удастся. Поэтому важное значение приобрели численные методы, позволяющие найти приближенное значение корня. Под численными методами подразумеваются методы решения задач, сводящиеся к арифметическим и некоторым логическим действиям над числами, т.е. к тем действиям, которые выполняет ЭВМ.

Существует множество численных методов. Рассмотрим только три из них:

- метод итераций;
- метод Ньютона;
- метод половинного деления.

### 2.5.1. Метод итераций

Представим уравнение  $F(x) = 0$  в виде:

$$x = f(x). \quad (2)$$

Это уравнение получается выделением  $x$  из уравнения  $F(x)$  и переносом того, что осталось, т.е.  $f(x)$ , в левую часть уравнения. Иначе можно получить уравнение (2) следующим способом: левую и правую часть уравнения (1) умножить на произвольную константу  $\lambda$  и прибавить к левой и правой части  $x$ , т.е. получаем уравнение вида:

$$x = x + \lambda F(x), \quad (3)$$

где  $f(x) = x + \lambda F(x)$ .

На заданном отрезке  $[a; b]$  выберем точку  $x_0$  – нулевое приближение – и найдем

$$x_1 = f(x_0),$$

потом найдем:

$$x_2 = f(x_1),$$

и т.д.

Таким образом, процесс нахождения корня уравнения сводится к последовательному вычислению чисел:

$$x_n = f(x_{n-1}) \quad n = 1, 2, 3, \dots$$

Этот процесс называется методом итераций.

Если на отрезке  $[a; b]$  выполнено условие:

$$|f'(x)| \leq q < 1,$$

то процесс итераций сходится, т.е.

$$\lim_{n \rightarrow \infty} x_n = \bar{x}$$

Процесс итераций продолжается до тех пор, пока

$$|x_n - x_{n-1}| \leq \varepsilon,$$

где  $\varepsilon$  – заданная абсолютная погрешность корня  $x$ . При этом будет выполняться:

$$|\bar{x} - x_n| \leq \varepsilon.$$

### 2.5.2. Метод Ньютона

Пусть уравнение  $F(x) = 0$  имеет один корень на отрезке  $[a; b]$ , причем  $F'(x)$  и  $F''(x)$  определены, непрерывны и сохраняют постоянные знаки на отрезке  $[a; b]$ .

Выберем на отрезке  $[a; b]$  произвольную точку  $x_0$  – нулевое приближение.

Затем найдем:

$$x_1 = x_0 - \frac{F(x_0)}{F'(x_0)},$$

потом

$$x_2 = x_1 - \frac{F(x_1)}{F'(x_1)}$$

Таким образом, процесс нахождения корня уравнения сводится к вычислению чисел  $x_n$  по формуле:

$$x_n = x_{n-1} - \frac{F(x_{n-1})}{F'(x_{n-1})}, \quad n = 1, 2, 3, \dots$$

Этот процесс называется методом Ньютона.

Процесс вычисления продолжается до тех пор, пока не будет выполнено условие:

$$|x_n - x_{n-1}| \leq \varepsilon$$

Точку  $x_0$  необходимо выбирать так, чтобы выполнялось условие:

$$F(x_0) \cdot F''(x_0) > 0,$$

иначе метод не будет сходиться.

### 2.5.3. Метод половинного деления

Пусть уравнение  $F(x) = 0$  имеет один корень на отрезке  $[a; b]$ . Функция  $F(x)$  непрерывна на отрезке  $[a; b]$ .

Метод половинного деления заключается в следующем:

Сначала выбираем начальное приближение, деля отрезок пополам, т.е.

$$x_0 = (a+b)/2.$$

Если  $F(x_0) = 0$ , то  $x_0$  является корнем уравнения. Если  $F(x_0) \neq 0$ , то выбираем тот из отрезков, на концах которого функция имеет противоположные знаки. Полученный отрезок снова делим пополам и выполняем действия сначала и т.д.

Процесс деления отрезка продолжаем до тех пор, пока длина отрезка, на концах которого функция имеет противоположные знаки, не будет меньше заданного числа  $\varepsilon$ .

### 3. Постановка задачи

1. Написать функцию с умалчиваемыми параметрами в соответствии с вариантом, продемонстрировать различные способы вызова функции:
  - с параметрами заданными явно,
  - с опущенными параметрами
  - часть параметров задана явно, а часть опущена.
2. Написать функцию с переменным числом параметров в соответствии с вариантом, продемонстрировать вызов функции с различным числом параметров.
3. Написать перегруженные функции в соответствии с вариантом. Написать демонстрационную программу для вызова этих функций.
4. Написать шаблон функций вместо перегруженных функций из задания 3. Написать демонстрационную программу для вызова этих функций. списка параметров
5. Решить уравнение указанным в варианте методом. Уравнение передать в функцию как параметр с помощью указателя.

### 4. Варианты

№ варианта	Функция с умалчиваемыми параметрами	Функция с переменным числом параметров	Перегруженные функции и шаблон функции	Передача функции как параметра другой функции с помощью указателя
1	Печать фамилии, имени и отчества	Минимальный элемент в списке параметров	Среднее арифметическое массива	Метод итераций $3 \sin \sqrt{x} + 0,35x - 3,8 = 0$ Отрезок, содержащий корень: [2;3] Точное значение: 2,2985
2	Печать фамилии, имени и возраста	Максимальный элемент в списке параметров	Количество отрицательных элементов в массиве	Метод Ньютона $3 \sin \sqrt{x} + 0,35x - 3,8 = 0$ Отрезок, содержащий корень: [2;3] Точное значение: 2,2985
3	Печать фамилии, курса и группы	Количество четных элементов в списке параметров	Максимальный элемент в массиве	Метод половинного деления $3 \sin \sqrt{x} + 0,35x - 3,8 = 0$ Отрезок, содержащий корень: [2;3] Точное значение: 2,2985
4	Печать фамилии, имени и рейтинга	Среднее арифметическое элементов в списке параметров	Минимальный элемент в массиве	Метод итераций $0,25x^3 + x - 1,2502 = 0$ Отрезок, содержащий корень: [0;2] Точное значение: 1,0001
5	Печать фамилии, курса и рейтинга	Максимальный из элементов в списке параметров,	Сортировка массива методом простого обмена	Метод Ньютона $0,25x^3 + x - 1,2502 = 0$ Отрезок, содержащий корень: [0;2]

		стоящих на четных местах		Точное значение: 1,0001
6	Печать фамилии, адреса и возраста	Максимальный из элементов в списке параметров, стоящих на нечетных местах	Сортировка массива методом простого выбора	Метод половинного деления $0,25x^3 + x - 1,2502 = 0$ Отрезок, содержащий корень: [0;2] Точное значение: 1,0001
7	Печать названия экзамена, количества сдающих и среднего балла	Минимальный из элементов в списке параметров, стоящих на четных местах	Сортировка массива методом простого включения	Метод итераций $x - \frac{1}{3 + \sin 3,6x} = 0$ Отрезок, содержащий корень: [0;0,85] Точное значение: 0,2624
8	Печать названия экзамена, даты экзамена и среднего балла	Минимальный из элементов в списке параметров, стоящих на нечетных местах	Поиск заданного элемента в массиве	Метод Ньютона $x - \frac{1}{3 + \sin 3,6x} = 0$ Отрезок, содержащий корень: [0;0,85] Точное значение: 0,2624
9	Печать координат точки	Среднее арифметическое из элементов в списке параметров, стоящих на четных местах	Поиск заданного элемента в отсортированном массиве	Метод половинного деления $x - \frac{1}{3 + \sin 3,6x} = 0$ Отрезок, содержащий корень: [0;0,85] Точное значение: 0,2624
10	Вычисление и печать расстояния от точки с координатами x1,y1 до центра координат	Среднее арифметическое из элементов в списке параметров, стоящих на нечетных местах	Удаление элемента с заданным номером из динамического массива	Метод итераций $0,1x^2 - x \ln x = 0$ Отрезок, содержащий корень: [1;2] Точное значение: 1,1183
11	Вычисление и печать расстояния от точки с координатами x1,y1 до точки с координатами x2,y2	Минимальный элемент в списке параметров	Удаление элемента с заданным ключом из динамического массива	Метод Ньютона $0,1x^2 - x \ln x = 0$ Отрезок, содержащий корень: [1;2] Точное значение: 1,1183
12	Печать фамилии, имени и	Максимальный элемент в списке	Добавление элемента с заданным	Метод половинного деления $0,1x^2 - x \ln x = 0$

	отчества	параметров	номером в динамический массив	Отрезок, содержащий корень: [1;2] Точное значение: 1,1183
13	Печать фамилии, имени и возраста	Количество четных элементов в списке параметров	Добавление элемента после элемента с заданным номером в динамический массив	Метод итераций $3x - 4\ln x - 5 = 0$ Отрезок, содержащий корень: [2;4] Точное значение: 3,2300
14	Печать фамилии, курса и группы	Среднее арифметическое элементов в списке параметров	Номер максимального элемента в массиве	Метод Ньютона $3x - 4\ln x - 5 = 0$ Отрезок, содержащий корень: [2;4] Точное значение: 3,2300
15	Печать фамилии, имени и рейтинга	Максимальный из элементов в списке параметров, стоящих на четных местах	Среднее арифметическое массива	Метод половинного деления $3x - 4\ln x - 5 = 0$ Отрезок, содержащий корень: [2;4] Точное значение: 3,2300
16	Печать фамилии, курса и рейтинга	Максимальный из элементов в списке параметров, стоящих на нечетных местах	Количество отрицательных элементов в массиве	Метод итераций $\arccos x - \sqrt{1 - 0,3x^3} = 0$ Отрезок, содержащий корень: [0;1] Точное значение: 0,5629
17	Печать фамилии, адреса и возраста	Минимальный из элементов в списке параметров, стоящих на четных местах	Добавление элемента с заданным номером в динамический массив	Метод Ньютона $\arccos x - \sqrt{1 - 0,3x^3} = 0$ Отрезок, содержащий корень: [0;1] Точное значение: 0,5629
18	Печать названия экзамена, количества сдающих и среднего балла	Минимальный из элементов в списке параметров, стоящих на нечетных местах	Сортировка массива методом простого обмена	Метод половинного деления $\arccos x - \sqrt{1 - 0,3x^3} = 0$ Отрезок, содержащий корень: [0;1] Точное значение: 0,5629
19	Печать названия экзамена, даты экзамена и среднего балла	Среднее арифметическое из элементов в списке параметров, стоящих на четных местах	Минимальный элемент в массиве	Метод итераций $\sqrt{1 - 0,4x^2} - \arcsin x = 0$ Отрезок, содержащий корень: [0;1] Точное значение: 0,7672

20	Печать координат точки	Среднее арифметическое из элементов в списке параметров, стоящих на нечетных местах	Сортировка массива методом простого выбора	Метод Ньютона $\sqrt{1 - 0,4x^2} - \arcsin x = 0$ Отрезок, содержащий корень: [0;1] Точное значение: 0,7672
21	Вычисление и печать расстояния от точки с координатами x1,y1 до центра координат	Минимальный элемент в списке параметров	Сортировка массива методом простого включения	Метод половинного деления $\sqrt{1 - 0,4x^2} - \arcsin x = 0$ Отрезок, содержащий корень: [0;1] Точное значение: 0,7672
22	Вычисление и печать расстояния от точки с координатами x1,y1 до точки с координатами x2,y2	Максимальный элемент в списке параметров	Поиск заданного элемента в массиве	Метод итераций $e^x - e^{-x} - 2 = 0$ Отрезок, содержащий корень: [0;1] Точное значение: 0,8814
23	Печать фамилии, имени и отчества	Количество четных элементов в списке параметров	Поиск заданного элемента в отсортированном массиве	Метод Ньютона Метод итераций $e^x - e^{-x} - 2 = 0$ Отрезок, содержащий корень: [0;1] Точное значение: 0,8814
24	Печать фамилии, имени и возраста	Среднее арифметическое элементов в списке параметров	Удаление элемента с заданным номером из динамического массива	Метод половинного деления Метод итераций $e^x - e^{-x} - 2 = 0$ Отрезок, содержащий корень: [0;1] Точное значение: 0,8814
25	Печать фамилии, курса и группы	Максимальный из элементов в списке параметров, стоящих на четных местах	Удаление элемента с заданным ключом из динамического массива	Метод итераций $x - 2 + \sin \frac{1}{x} = 0$ Отрезок, содержащий корень: [1;2] Точное значение: 1,3077

## 5. Методические указания

1. В функции с умалчиваемыми параметрами использовать структурированный тип данных.

2. При демонстрации вызова функции с умалчиваемыми параметрами учесть, что опускать параметры функции можно только с конца.
3. В функции с переменными числом параметров можно использовать любой механизм определения конца списка параметров (передачу количества параметров как параметр функции или использование признака конца списка параметров).
4. Перегрузить функции для массивов типа char, int, и double.
5. Инстанцировать шаблон функции для типов char, int, и double.
6. Для нахождения корня уравнения написать как минимум две функции. Одна функция реализует уравнение, для которого вычисляется корень, другая - метод решения уравнения, указанный в варианте. Первая функция передается во вторую как параметр, с помощью указателя.
7. Точность нахождения корня уравнения выбирается не менее 0.001.
8. Полученный результат вычисления корня сравнить с точным значением, заданным в задании.

## **6. Содержание отчета**

1. Постановка задачи (общая и для конкретного варианта).
2. Определения функций для реализации поставленных задач.
3. Определение функции main().
4. Тесты

# Лабораторная работа №8

## Динамические структуры данных

### 1. Цель работы:

- 1) Получить практические навыки работы с однонаправленными списками;
- 2) получить практические навыки работы с двунаправленными списками;
- 3) получить практические навыки работы с деревьями.

### 2. Краткие теоретические сведения

Во многих задачах требуется использовать данные, у которых конфигурация, размеры и состав могут меняться в процессе выполнения программы. Для их представления используют динамические информационные структуры. К таким структурам относят:

- однонаправленные списки;
- двунаправленные списки;
- стек;
- очередь;
- бинарные деревья.

Они отличаются способом связи отдельных элементов и/или допустимыми операциями. Динамическая структура может занимать несмежные участки динамической памяти.

#### 2.1. Однонаправленные списки

Наиболее простой динамической структурой является однонаправленный список, элементами которого служат объекты структурного типа (рис.).

Рис.. Линейный однонаправленный список

Описание простейшего элемента такого списка выглядит следующим образом:

```
struct имя_типа
{
    информационное поле;
    адресное поле;
};
```

- информационное поле – это поле любого, ранее объявленного или стандартного, типа;
- адресное поле – это указатель на объект того же типа, что и определяемая структура, в него записывается адрес следующего элемента списка.



---

```
struct point
{
    int data;    //информационное поле
    point* next; //адресное поле
};
```

---

Информационных полей может быть несколько.

Для того, чтобы создать список, нужно создать сначала первый элемент списка, а затем в цикле добавить к нему остальные элементы. Добавление может выполняться как в начало, так и в конец списка.

---

```
//создание однонаправленного списка
//добавление в конец
point* make_list(int n)
{
    point* beg; //указатель на первый элемент
    point* p, *r; //вспомогательные указатели
    beg=new(point); //выделяем память под первый элемент
    cout<<"\n?";
    cin>>beg->data; //вводим значение информационного поля
    beg->next=0; //обнуляем адресное поле
    //ставим на этот элемент указатель p (последний элемент)
    p=beg;
    for(int i=0; i<n-1; i++)
    {
        r=new(point); //создаем новый элемент
        cout<<"\n?";
        cin>>r->data;
        r->next=0;
        p->next=r; //связываем p и r
    }
    //ставим на r указатель p (последний элемент)
    p=r;
    return beg; //возвращаем beg как результат функции
}
```

---

Для обработки списка организуется цикл, в котором нужно переставлять указатель p с помощью оператора `p=p->next` на следующий элемент списка до тех пор, пока указатель p не станет равен 0, т. е. будет достигнут конец списка.

---

```
void print_list(point* beg)
//печать списка
{
    point* p=beg; //начало списка
    while(p!=0)
    {
        cout<<p->data<<"\t";
        p=p->next; //переход к следующему элементу
    }
}
```

---

В динамические структуры легко добавлять элементы и из них легко удалять элементы, т. к. для этого достаточно изменить значения адресных полей.

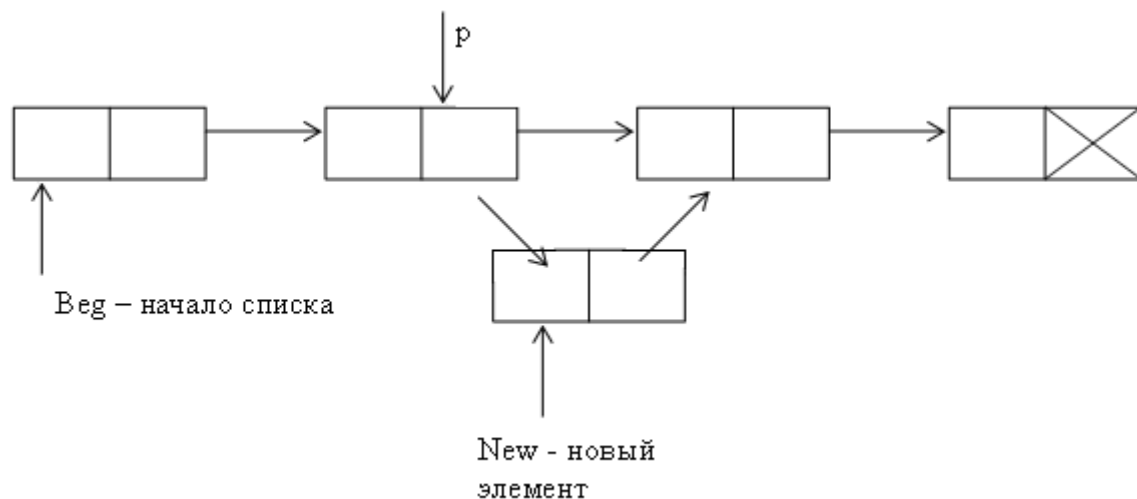


Рис. Добавление элемента в список

---

```

point* add_point(point* beg, int k)
//добавление элемента с номером k
{
    point*p=beg;//встали на первый элемент
    point*New=new(point);//создали новый элемент
    cout<<"Key?";cin>>New->data;
    if(k==0)//добавление в начало, если k=0
    {
        New->next=beg;
        beg=New;
        return beg;
    }
    for(int i=0;i<k-1&&p!=0;i++)
    p=p->next;//проходим по списку до(k-1) элемента или до конца
    if(p!=0)//если k-й элемент существует
    {
        New->next=p->next;//связываем New и k-й элемент
        p->next=New;//связываем (k-1)элемент и New
    }
    return beg;
}

```

---

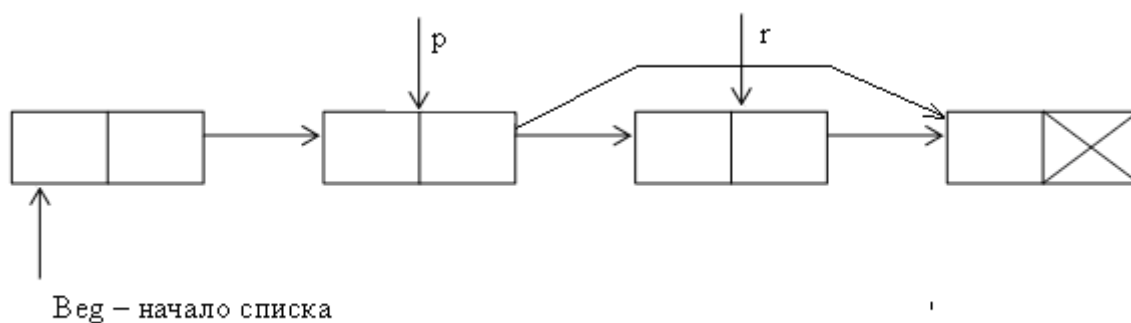


Рис. Удаление элемента из списка

---

```

point* del_point(point*beg,int k)
//удаление элемента с номером k из списка

```

```

{
    point*p=beg;
    if(k==0)//удаление первого элемента
    {
        beg=beg->next;
        delete p;
        return beg;
    }
//проходим по списку до элемента с номером k-1
    for(int i=1;i<k&& p->next!=0;i++)
        p=p->next;
/*если такого элемента в списке нет, то возвращаем указатель на
начало списка в качестве результата функции*/
    if (p->next==0) return beg;
    point* r=p->next;//ставим указатель r на k-й элемент
    p->next=r->next;//связываем k-1 и k+1 элемент
    delete r;//удаляем k-й элемент из памяти
    return beg;
}

```

---

## 2.1. Двухнаправленные списки

Двухнаправленный список имеет два адресных поля, которые указывают на следующий элемент списка и на предыдущий. Поэтому двигаться по такому списку можно как слева направо, так и справа налево.

```

//формирование двухнаправленного списка
struct point
{
    char *key;//адресное поле - динамическая строка
    point *next;//указатель на следующий элемент
    point *pred;//указатель на предыдущий элемент
};
point* make_point()
//создание одного элемента
{
    point*p=new(point);
    p->next=0;p->pred=0;//обнуляем указатели
    char s[50];
    cout<<"\nEnter string:";
    cin>>s;
    p->key=new char[strlen(s)+1];//выделение памяти под строку
    strcpy(p->key,s);
    return p;
}
point*make_list(int n)
//создание списка
{
    point *p,*beg;
    beg=make_point();//создаем первый элемент
    for(int i=1;i<n;i++)
    {
        p=make_point();//создаем один элемент
//добавление элемента в начало списка
        p->next=beg;//связываем p с первым элементом
    }
}

```

```

        beg->pred=p;//связываем первый элемент с p
        beg=p;// p становится первым элементом списка
    }
    return beg;
}

```

---

### 2.3. Очередь и стек

Очередь и стек – это частные случаи однонаправленного списка.

В стеке добавление и удаление элементов осуществляются с одного конца, который называется вершиной стека. Поэтому для стека можно определить функции:

- top() – доступ к вершине стека
- pop() – удаление элемента из вершины;
- push() – добавление элемента в вершину.

Такой принцип обслуживания называют LIFO (last in – first out, последний пришел, первый ушел).

В очереди добавление осуществляется в один конец, а удаление из другого конца. Такой принцип обслуживания называют FIFO (first in – first out, первый пришел, первый ушел).

Для очереди также определяют функции:

- front() – доступ к первому элементу;
- back() – доступ к последнему элементу;
- pop() – удаление элемента из конца;
- push() – добавление элемента в начало.

### 2.4. Бинарные деревья

Бинарное дерево – это динамическая структура данных, состоящая из узлов, каждый из которых содержит, кроме данных, не более двух ссылок на различные бинарные деревья. На каждый узел имеется ровно одна ссылка.

Описать такую структуру можно следующим образом:

```

struct point
{
    int data;//информационное поле
    point *left;//адрес левого поддерева
    point *right;//адрес правого поддерева
};

```

---

Начальный узел называется корнем дерева. Узел, не имеющий поддеревьев, называется листом. Исходящие узлы называются предками, входящие — потомками. Высота дерева определяется количеством уровней, на которых располагаются его узлы.

Если дерево организовано таким образом, что для каждого узла все ключи его левого поддерева меньше ключа этого узла, а все ключи его правого поддерева — больше, оно называется деревом поиска. Одинаковые ключи не допускаются. В дереве поиска можно найти элемент по ключу, двигаясь от корня и переходя на левое или правое поддерево в зависимости от значения ключа в каждом узле. Такой поиск гораздо эффективнее поиска по списку, поскольку время поиска определяется высотой дерева, а она пропорциональна двоичному логарифму количества узлов.

В идеально сбалансированном дереве количество узлов справа и слева отличается не более чем на единицу.

Линейный список можно представить как вырожденное бинарное дерево, в котором каждый узел имеет не более одной ссылки. Для списка среднее время поиска равно половине длины списка.

Деревья и списки являются рекурсивными структурами, т. к. каждое поддерево также является деревом. Таким образом, дерево можно определить как рекурсивную структуру, в которой каждый элемент является:

- либо пустой структурой;
- либо элементом, с которым связано конечное число поддеревьев.

Действия с рекурсивными структурами удобнее всего описываются с помощью рекурсивных алгоритмов.

### 2.4.1. Обход дерева

Для того, чтобы выполнить определенную операцию над всеми узлами дерева, все узлы надо обойти. Такая задача называется обходом дерева. При обходе узлы должны посещаться в определенном порядке. Существуют три принципа упорядочивания. Рассмотрим дерево, представленное на рис.

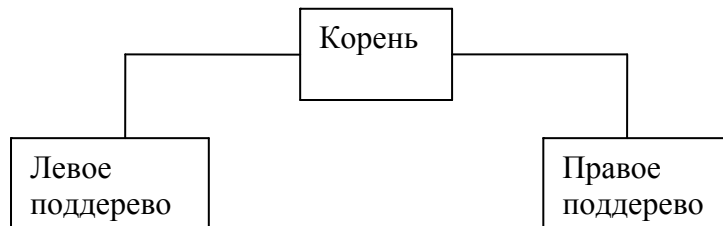


Рис. Бинарное дерево

На этом дереве можно определить три метода упорядочивания:

Слева направо: Левое поддерево – Корень – Правое поддерево;

Сверху вниз: Корень – Левое поддерево – Правое поддерево;

Снизу вверх: Левое поддерево – Правое поддерево – Корень.

Эти три метода можно сформулировать в виде рекурсивных алгоритмов.

---

```

void Run(point*p)
//обход слева направо
{
    if(p)
    {
        <обработка p->data>
        Run(p->left); //переход к левому поддереву
        Run(p->right); //переход к правому поддереву
    }
}
  
```

---

Если в качестве операции обработки узла поставить операцию вывода информационного поля, то мы получим функцию для печати дерева.

### 2.4.2. Формирование дерева

---

```

//построение дерева поиска
Point* first(int d) //формирование первого элемента дерева
{
    Point* p=new Point;
    p->key=d;
    p->left=0;
    p->right=0;
    return p;
}
//добавление элемента d в дерево поиска
Point* Add(Point*root,int d)
{
    Point*p=root; //корень дерева
  
```

```

    Point*r;
//флаг для проверки существования элемента d в дереве
    bool ok=false;
    while (p&&!ok)
    {
        r=p;
        if(d==p->key)ok=true;//элемент уже существует
        else
            if(d<p->key)p=p->left;//пойти в левое поддерево
            else p=p->right;//пойти в правое поддерево
    }
    if(ok) return p;//найдено, не добавляем
    //создаем узел
    Point* New_point=new Point();//выделили память
    New_point->key=d;
    New_point->left=0;
    New_point->right=0;
// если d<r->key, то добавляем его в левое поддерево
    if(d<r->key) r->left=New_point;
// если d>r->key, то добавляем его в правое поддерево
    else r->right =New_point;
    return New_point;
}

```

---

```

//построение идеально сбалансированного дерева
point* Tree(int n,point* p)
{
    point*r;
    int nl,nr;
    if(n==0){p=NULL;return p;}

    nl=n/2;
    nr=n-nl-1;
    r=new point;
    cout<<"?";
    cin>>r->data;
    r->left=Tree(nl,r->left);
    r->right=Tree(nr,r->right);
    p=r;
    return p;
}

```

---

### 3. Постановка задачи

1. Сформировать однонаправленный список, тип информационного поля указан в варианте.
2. Распечатать полученный список.
3. Выполнить обработку списка в соответствии с заданием.
4. Распечатать полученный список.
5. Удалить список из памяти.
6. Сформировать двунаправленный список, тип информационного поля указан в варианте.
7. Распечатать полученный список.

8. Выполнить обработку списка в соответствии с заданием.
9. Распечатать полученный список.
10. Удалить список из памяти.
11. Сформировать идеально сбалансированное бинарное дерево, тип информационного поля указан в варианте.
12. Распечатать полученное дерево.
13. Выполнить обработку дерева в соответствии с заданием, вывести полученный результат.
14. Преобразовать идеально сбалансированное дерево в дерево поиска.
15. Распечатать полученное дерево.

#### 4. Варианты

№ варианта	Однонаправленный	Двунаправленный	Бинарное дерево
1	Тип информационного поля int. Удалить из списка все элементы с четными информационными полями.	Тип информационного поля char*. Добавить в список элемент с заданным номером.	Тип информационного поля char. Найти количество элементов с заданным ключом.
2	Тип информационного поля double. Удалить из списка все элементы с четными номерами (2, 4, 6 и т. д.).	Тип информационного поля char*. Добавить в список элементы с номерами 1, 3, 5 и т. д.	Тип информационного поля int. Найти максимальный элемент в дереве.
3	Тип информационного поля int. Удалить из списка первый элемент с четным информационным полем.	Тип информационного поля double. Добавить в список элемент после элемента с заданным информационным полем.	Тип информационного поля char*. Найти количество листьев в дереве.
4	Тип информационного поля int. Удалить из списка последний элемент с четным информационным полем.	Тип информационного поля char*. Добавить в список элемент с заданным номером.	Тип информационного поля double. Найти минимальный элемент в дереве.
5	Тип информационного поля char*. Добавить в список элемент после элемента с заданным информационным полем.	Тип информационного поля int. Удалить из списка все элементы с четными информационными полями.	Тип информационного поля char. Найти высоту дерева.
6	Тип информационного поля char*. Добавить в список элемент с заданным номером.	Тип информационного поля double. Удалить из списка все элементы с четными	Тип информационного поля int. Найти среднее арифметическое элементов дерева.

		номерами (2, 4, 6 и т. д.).	
7	Тип информационного поля double. Добавить в список после каждого элемента с отрицательным информационным полем элемент с информационным полем равным 0.	Тип информационного поля int. Удалить из списка первый элемент с четным информационным полем.	Тип информационного поля char*. Найти количество элементов дерева, начинающихся с заданного символа.
8	Тип информационного поля char*. Добавить в список элементы с номерами 1, 3, 5 и т. д.	Тип информационного поля int. Удалить из списка последний элемент с четным информационным полем.	Тип информационного поля char. Найти количество элементов с заданным ключом.
9	Тип информационного поля int. Удалить из списка все элементы с четными информационными полями.	Тип информационного поля char*. Добавить в список элементы с номерами 1, 3, 5 и т. д.	Тип информационного поля double. Найти максимальный элемент в дереве.
10	Тип информационного поля double. Удалить из списка все элементы с четными номерами (2, 4, 6 и т. д.).	Тип информационного поля char*. Добавить в список элемент после элемента с заданным информационным полем.	Тип информационного поля int Найти количество листьев в дереве.
11	Тип информационного поля int. Удалить из списка первый элемент с четным информационным полем.	Тип информационного поля char*. Добавить в список элемент с заданным номером.	Тип информационного поля double. Найти минимальный элемент в дереве.
12	Тип информационного поля int. Удалить из списка последний элемент с четным информационным полем.	Тип информационного поля char*. Добавить в список элемент с заданным номером.	Тип информационного поля char. Найти высоту дерева.
13	Тип информационного поля char*. Добавить в список элемент после элемента с заданным	Тип информационного поля double. Удалить из списка все элементы с отрицательными	Тип информационного поля int. Найти среднее арифметическое элементов дерева.



	информационным полем.	информационными полями.	
14	Тип информационного поля char*. Добавить в список элемент с заданным номером.	Тип информационного поля int. Удалить из списка последний элемент с четным информационным полем.	Тип информационного поля char. Найти количество элементов с заданным ключом.
15	Тип информационного поля double. Добавить в список после каждого элемента с отрицательным информационным полем элемент с информационным полем равным 0.	Тип информационного поля int. Удалить из списка все элементы с четными номерами (2, 4, 6 и т. д.).	Тип информационного поля char*. Найти количество элементов дерева, начинающихся с заданного символа
16	Тип информационного поля char*. Добавить в список элементы с номерами 1, 3, 5 и т. д.	Тип информационного поля int. Удалить из списка первый элемент с четным информационным полем.	Тип информационного поля int. Найти максимальный элемент в дереве.
17	Тип информационного поля int. Удалить из списка все элементы с четными информационными полями.	Тип информационного поля char*. Добавить в список элемент с заданным номером.	Тип информационного поля double. Найти количество листьев в дереве.
18	Тип информационного поля double. Удалить из списка все элементы с четными номерами (2, 4, 6 и т. д.).	Тип информационного поля char*. Добавить в список элемент с заданным номером.	Тип информационного поля int. Найти минимальный элемент в дереве.
19	Тип информационного поля int. Удалить из списка первый элемент с четным информационным полем.	Тип информационного поля char*. Добавить в список элементы с номерами 1, 3, 5 и т. д.	Тип информационного поля char. Найти высоту дерева.
20	Тип информационного поля int. Удалить из списка последний элемент с четным информационным полем.	Тип информационного поля char*. Добавить в список элемент после элемента с заданным информационным полем.	Тип информационного поля double. Найти среднее арифметическое элементов дерева.

21	Тип информационного поля char*. Добавить в список элемент после элемента с заданным информационным полем.	Тип информационного поля int. Удалить из списка все элементы с четными информационными полями.	Тип информационного поля char*. Найти количество элементов дерева, начинающихся с заданного символа.
22	Тип информационного поля char*. Добавить в список элемент с заданным номером.	Тип информационного поля double. Удалить из списка все элементы с четными номерами (2, 4, 6 и т. д.).	Тип информационного поля char. Найти количество элементов с заданным ключом.
23	Тип информационного поля double. Добавить в список после каждого элемента с отрицательным информационным полем элемент с информационным полем равным 0.	Тип информационного поля int. Удалить из списка первый элемент с четным информационным полем.	Тип информационного поля char*. Найти количество листьев в дереве.
24	Тип информационного поля char*. Добавить в список элементы с номерами 1, 3, 5 и т. д.	Тип информационного поля int. Удалить из списка последний элемент с четным информационным полем.	Тип информационного поля double. Найти максимальный элемент в дереве.
25	Тип информационного поля int. Удалить из списка все элементы с четными информационными полями.	Тип информационного поля char*. Добавить в список элементы с номерами 1, 3, 5 и т. д.	Тип информационного поля double. Найти минимальный элемент в дереве.

## 5. Методические указания

1. Описания структур для формирования списков/деревьев, а также функции для их обработки сохранить в библиотечном файле с расширением .h (например, point.h). Функцию main() сохранить в файле с расширением .cpp. Библиотечный файл подключить с помощью директивы #include "имя\_файла.h".
2. Для выделения памяти под информационные поля типа char\* использовать операцию new, для удаления из памяти – операцию delete.
3. Для формирования элементов списков/дерева написать отдельные функции.
4. Для формирования списков/дерева, удаления добавления элементов, поиска заданных элементов написать отдельные функции.
5. В функции main() должны быть размещены только описания переменных и обращения к соответствующим функциям.
6. Если в списке/дереве отсутствуют элементы, соответствующие критерию поиска (например, при удалении элемента с номером k, k больше, чем

количество элементов в списке), должно быть выведено сообщение о том, что требуемые элементы не найдены.

7. Интерфейс реализовать с помощью текстового меню.

## **6. Содержание отчета**

1. Постановка задачи (общая и для конкретного варианта).
2. Определения функций для реализации поставленных задач.
3. Определение функции main().
4. Тесты.

# Лабораторная работа №9

## Хранение данных на внешних носителях

### 1. Цель работы:

1. Получение практических навыков записи структурированной информации в файлы в стиле C;
2. Получение практических навыков записи структурированной информации в файлы в стиле C++;

### 2. Краткие теоретические сведения

#### 2.1. Поточковый ввод-вывод в стиле C

Особенностью C является отсутствие в этом языке структурированных файлов. Все файлы рассматриваются как не структурированная последовательность байтов. При таком подходе понятие файла распространяется и на различные устройства.

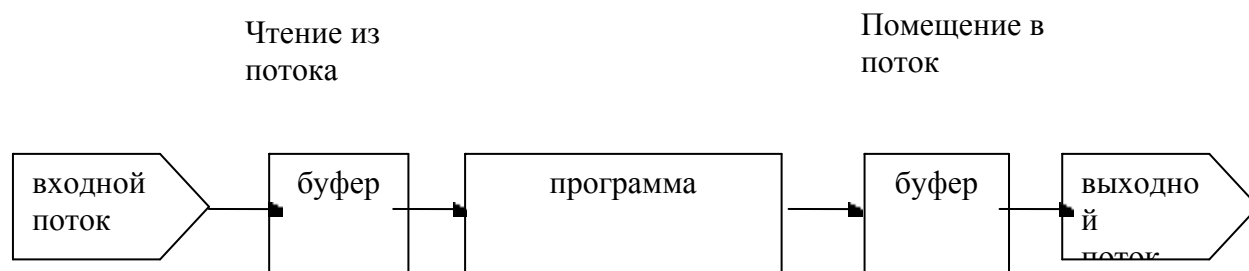
В C существуют средства ввода-вывода. Все операции ввода-вывода реализуются с помощью функций, которые находятся в библиотеке C. Библиотека C поддерживает три уровня ввода-вывода:

- потоковый ввод-вывод;
- ввод-вывод нижнего уровня;
- ввод-вывод для консоли и портов (зависит от ОС).

Поток – это абстрактное понятие, относящееся к любому переносу данных от источника к приемнику.

Чтение данных из потока называется извлечением, вывод в поток – помещением, или включением.

Поток определяется как последовательность байтов и не зависит от конкретного устройства, с которым производится обмен (оперативная память, файл на диске, клавиатура или принтер). Обмен с потоком для увеличения скорости передачи данных производится, как правило, через специальную область оперативной памяти — буфер. Буфер накапливает байты, и фактическая передача данных выполняется после заполнения буфера. При вводе это дает возможность исправить ошибки, если данные из буфера еще не отправлены в программу.



При работе с потоком можно:

- Открывать и закрывать потоки (связывать указатели на поток с конкретными файлами);
- вводить и выводить строку, символ, форматированные данные, порцию данных произвольной длины;
- анализировать ошибки ввода-вывода и достижения конца файла;

- управлять буферизацией потока и размером буфера;
- получать и устанавливать указатель текущей позиции в файле.

Функции библиотеки ввода-вывода находятся в заголовочном файле <stdio.h>.

Прежде чем начать работать с потоком, его надо инициировать, т. е. открыть. При этом поток связывается со структурой предопределенного типа FILE, определение которой находится в библиотечном файле <stdio.h>. В структуре находится указатель на буфер, указатель на текущую позицию файла и т. п. При открытии потока, возвращается указатель на поток, т. е. на объект типа FILE.

```
#include <stdio.h>;
. . . . .
FILE *fp;
. . . . .
fp= fopen( "t.txt", "r");
```

где fopen(<имя\_файла>, <режим\_открытия>) - функция для инициализации файла. Существуют следующие режимы для открытия файла:

Режим	Описание режима открытия файла
r	Файл открывается для чтения, если файл не существует, то выдается ошибка при исполнении программы.
w	Файл открывается для записи, если файл не существует, то он будет создан, если файл уже существует, то вся информация из него стирается.
a	Файл открывается для добавления, если файл не существует, то он будет создан, если существует, то информация из него не стирается, можно выполнять запись в конец файла
r+	Файл открывается для чтения и записи, изменить размер файла нельзя, если файл не существует, то выдается ошибка при исполнении программы.
w+	Файл открывается для чтения и записи, если файл не существует, то он будет создан, если файл уже существует, то вся информация из него стирается.
a+	Файл открывается для чтения и записи, если файл не существует, то он будет создан, если существует, то информация из него не стирается, можно выполнять запись в конец файла

Поток можно открыть в текстовом (t) или двоичном (b) режиме. По умолчанию используется текстовый режим. В явном виде режим указывается следующим образом:

- "r+b" или "rb" - двоичный (бинарный) режим;
- "r+t" или "rt" – текстовый режим.

В файле stdio.h определена константа EOF, которая сообщает об окончании файла (отрицательное целое число).

При открытии потока могут возникать следующие ошибки:

- файл, связанный с потоком не найден (при чтении из файла);
- диск заполнен (при записи);
- диск защищен от записи (при записи) и т. п.

В этих случаях указатель на поток приобретет значение NULL (0). Указатель на поток, отличный от аварийного не равен 0.

Для вывода об ошибке при открытии потока используется стандартная библиотечная функция из файла <stdio.h>

```
void perror (const char*s);
```

```
if ((fp=fopen("t.txt", "w")==NULL)
{
```

```
// выводит строку символов с сообщением об ошибке
perror("\ношибка при открытии файла");
exit(0);
}
```

---

После работы с файлом, его надо закрыть

```
fclose(<указатель_на_поток>);
```

Когда программа начинает выполняться, автоматически открываются несколько потоков, из которых основными являются:

- стандартный поток ввода (stdin);
- стандартный поток вывода (stdout);
- стандартный поток вывода об ошибках (stderr).

По умолчанию stdin ставится в соответствие клавиатура, а потокам stdout и stderr - монитор. Для ввода-вывода с помощью стандартных потоков используются функции:

- getchar()/putchar() – ввод-вывод отдельного символа;
- gets()/puts() – ввод-вывод строки;
- scanf()/printf() – форматированный ввод/вывод.

Аналогично работе со стандартными потоками выполняется ввод-вывод в потоки, связанные с файлами.

Для символьного ввода-вывода используются функции:

- int fgetc(FILE\*fp), где fp – указатель на поток, из которого выполняется считывание. Функция возвращает очередной символ в форме int из потока fp. Если символ не может быть прочитан, то возвращается значение EOF.
- int fputc(int c, FILE\*fp), где fp – указатель на поток, в который выполняется запись, c – переменная типа int, в которой содержится записываемый в поток символ. Функция возвращает записанный в поток fp символ в форме int. Если символ не может быть записан, то возвращается значение EOF.

Для построчного ввода-вывода используются следующие функции:

- char\* fgets(char\* s, int n, FILE\* f), где char\*s – адрес, по которому размещаются считанные байты, int n – количество считанных байтов, FILE\* f – указатель на файл, из которого производится считывание.

Прием байтов заканчивается после передачи n-1 байтов или при получении управляющего символа '\n'. Управляющий символ тоже передается в принимающую строку. Строка в любом случае заканчивается '\0'. При успешном завершении работы функция возвращает указатель на прочитанную строку, при неуспешном – 0.

- int puts(char\* s, FILE\* f), где char\*s – адрес, из которого берутся записываемые в файл байты, FILE\* f – указатель на файл, в который производится запись.

Символ конца строки ('\0') в файл не записывается. Функция возвращает EOF, если при записи в файл произошла ошибка, при успешной записи возвращает неотрицательное число.

Для блочного ввода-вывода используются функции:

- int fread(void\*ptr, int size, int n, FILE\*f), где void\*ptr – указатель на область памяти, в которой размещаются считанные из файла данные, int size – размер одного считываемого элемента, int n – количество считываемых элементов, FILE\*f – указатель на файл, из которого производится считывание.

В случае успешного считывания функция возвращает количество считанных элементов, иначе – EOF.

- int fwrite(void\*ptr, int size, int n, FILE\*f), где void\*ptr – указатель на область памяти, в которой размещаются считанные из файла данные, int

size – размер одного записываемого элемента, int n – количество записываемых элементов, FILE\*f – указатель на файл, в который производится запись.

В случае успешной записи функция возвращает количество записанных элементов, иначе – EOF.

В некоторых случаях информацию удобно записывать в файл без преобразования, т. е. в символьном виде пригодном для непосредственного отображения на экран. Для этого можно использовать функции форматированного ввода-вывода:

- int fprintf(FILE \*f, const char\*fmt, . . . ) , где FILE\*f – указатель на файл, в который производится запись, const char\*fmt – форматная строка, . . . – список переменных, которые записываются в файл.

Функция возвращает число записанных символов.

- int fscanf(FILE \*f, const char\*fmt, par1, par2, . . . ) , где FILE\*f – указатель на файл, из которого производится чтение, const char\*fmt – форматная строка, par1, par2, . . . – список переменных, в которые заносится информация из файла.

Функция возвращает число переменных, которым присвоено значение.

Средства прямого доступа дают возможность перемещать указатель текущей позиции в потоке на нужный байт. Для этого используется функция

- int fseek(FILE \*f, long off, int org) , где FILE \*f – указатель на файл, long off – позиция смещения, int org – начало отсчета.

Смещение задается выражение или переменной и может быть отрицательным, т. е. возможно перемещение как в прямом, так и в обратном направлениях. Начало отсчета задается одной из определенных в файле <stdio.h> констант:

```
SEEK_SET ==0 – начало файла;  
SEEK_CUR==1 – текущая позиция;  
SEEK_END ==2 – конец файла.
```

Функция возвращает 0, если перемещение в потоке выполнено успешно, иначе возвращает ненулевое значение.

## 2.2. Обработка элементов файла

Для того чтобы удалить элемент из файла нужно использовать вспомогательный файл. Во вспомогательный файл переписываются все элементы исходного файла за исключением тех, которые требуется удалить. После этого исходный файл удаляется из памяти, а вспомогательному файлу присваивается имя исходного файла.

---

```
void del(char *filename)  
{//удаление записи с номером x  
FILE *f;//исходный файл  
FILE*temp;//вспомогательный файл  
//открыть исходный файл для чтения  
f=fopen(filename,"rb");  
//открыть вспомогательный файл для записи  
temp=fopen("temp","wb")  
student a;//буфер для чтения данных из файла  
//считываем данные из исходного файла в буфер  
for(long i=0; fread(&a,sizeof(student),1,f);i++)  
    if(i!=x)//если номер записи не равен x  
    {  
        //записываем данные из буфера во временный файл  
        fwrite(&a,sizeof(student)1,temp);  
    }  
else
```

```

    {
        cout<<a<<" - is deleting...";
    }
    fclose(f); //закрываем исходный файл
    fclose(temp); //закрываем временный файл
    remove(filename); //удаляем исходный файл
    rename("temp", filename); //переименовываем временный файл
}

```

Для корректировки элементов файла используется аналогичный алгоритм. Данные из исходного файла переписываются во вспомогательный файл, но записи, которые нужно изменить записываются в откорректированном виде.

Для добавления элементов в начало или в середину файла также используется вспомогательный файл, в который в нужное место добавляются новые данные.

Для добавления элементов конец файла достаточно открыть его в режиме “a” или “a+” (для добавления) и записать новые данные в конец файла.

```

f=fopen(filename,"ab");//открыть файл для добавления
cout<<"\nHow many records would you add to file?";
cin>>n;
for(int i=0;i<n;i++)
{
    //прочитать объект
    fwrite(&a,sizeof(student),1,f);//записать в файл
}
fclose(f); //закреть файл

```

### 2.3. Поточный ввод-вывод в стиле C++

C++ предоставляет возможность ввода/вывода как на низком уровне – неформатированный ввод-вывод, так и на высоком – форматированный ввод-вывод. При неформатированном вводе/выводе передача информации осуществляется блоками байтов данных без какого-либо преобразования. При форматированном - байты группируются таким образом, чтобы их можно было воспринимать как типизированные данные (целые числа, строки символов, числа с плавающей запятой и т. п.)

По направлению обмена потоки можно разделить на

- входные (данные вводятся в память),
- выходные (данные выводятся из памяти),
- двунаправленные (допускающие как извлечение, так и включение).

По виду устройств, с которыми работает поток, потоки можно разделить на стандартные, файловые и строковые:

- стандартные потоки предназначены для передачи данных от клавиатуры и на экран дисплея,
- файловые потоки — для обмена информацией с файлами на внешних носителях данных (например, на магнитном диске),
- строковые потоки — для работы с массивами символов в оперативной памяти.

Для работы со стандартными потоками библиотека C++ содержит библиотеку `<iostream.h>`. При этом в программе автоматически становятся доступными объекты:

- `cin` - объект, соответствует стандартному потоку ввода,
- `cout` - объект, соответствует стандартному потоку вывода.

Оба эти потока являются буферизированными.

Форматированный ввод/вывод реализуется через две операции:

- `<<` - вывод в поток;
- `>>` - чтение из потока.



Использование файлов в программе предполагает следующие операции:

- создание потока;
- открытие потока и связывание его с файлом;
- обмен (ввод/вывод);
- уничтожение потока;
- закрытие файла.

Для работы со файловыми потоками библиотека C++ содержит библиотеки:

- `<ifstream.h>` – для работы с входными потоками,
- `<ofstream.h>` – для работы с выходными потоками
- `<fstream.h>` – для работы с двунаправленными потоками.

В библиотечных файлах потоки описаны как классы, т. е. представляют собой пользовательские типы данных (аналогично структурам данных). В описание класса, кроме данных, добавляются описания функций, обрабатывающих эти данные (соответственно компонентные данные и компонентные функции (методы)). Обращаться к компонентным функциям можно также как и к компонентным данным с помощью `.` или `->`.

Для создания файлового потока используются специальные методы – конструкторы, которые создают поток соответствующего класса, открывают файл с указанным именем и связывают файл с потоком:

- `ifstream(const char *name, int mode = ios::in);` //входной поток
- `ofstream(const char *name, int mode = ios::out | ios::trunc);` //выходной поток
- `fstream(const char *name, int mode = ios::in | ios::out);` //двунаправленный поток

Вторым параметром является режим открытия файла. Вместо значения по умолчанию можно указать одно из следующих значений, определенных в классе `ios`.

<code>ios::in</code>	открыть файл для чтения;
<code>ios::out</code>	открыть файл для записи;
<code>ios::ate</code>	установить указатель на конец файла, читать нельзя, можно только записывать данные в конец файла;
<code>ios::app</code>	открыть файл для добавления;
<code>ios::trunc</code>	если файл существует, то создать новый;
<code>ios::binary</code>	открыть в двоичном режиме;
<code>ios::nocreate</code>	если файл не существует, выдать ошибку, новый файл не открывать
<code>ios::noreplace</code>	если файл существует, выдать ошибку, существующий файл не открывать;

Открыть файл в программе можно с использованием либо конструкторов, либо метода `open`, имеющего такие же параметры, как и в соответствующем конструкторе.

```
fstream f; //создает файловый поток f
//открывается файл, который связывается с потоком
f.open("../f.dat", ios::in);
// создает и открывает для чтения файловый поток f
fstream f ("../f.dat", ios::in);
```

После того как файловый поток открыт, работать с ним можно также как и со стандартными потоками `cin` и `cout`. При чтении данных из входного файла надо контролировать, был ли достигнут конец файла после очередной операции вывода. Это можно делать с помощью метода `eof()`.

Если в процессе работы возникнет ошибочная ситуация, то потоковый объект принимает значение равное 0.

Когда программа покидает область видимости потокового объекта, то он уничтожается, при этом перестает существовать связь между потоковым объектом и

физическим файлом, а сам файл закрывается. Если файл требуется закрыть раньше, то используется метод `close()`.

---

```
//создание файла из элементов типа person
struct person
{
char name[20];
int age;
};
person *mas;//динамический массив
fstream f("f.dat",ios::out);//двунаправленный файловый поток
int n;
cout<<"N?";
cin>>n;
mas=new person[n];//создаем динамический массив
for(int i=0;i<n;i++)
{
    cout<<"?";
//ввод одного элемента типа person из стандартного потока cin
    cin>>mas[i].name;
    cin>>mas[i].age;
}
//запись элементов массива в файловый поток
for(i=0;i<n;i++)
{

    f<<mas[i].name;f<<"\n";
    f<<mas[i].age;f<<"\n";

}
f.close();//закрытие потока

//чтение элементов из файла
person p;
f.open("f.dat",ios::in);//открываем поток для чтения
do
{
/*читаем элементы типа person из файлового потока f в переменную
p*/
    f>>p.name;
    f>>p.age;
    // если достигнут конец файла, выходим из цикла
    if (f.eof())break;
    //вывод на экран
    cout<<p.name<<" "<<p.age<<"\n";

}while(!f.eof());
f.close();//закрытие потока
```

---

### **3. Постановка задачи**

1. Используя ввод-вывод в стиле C создать файл и записать в него структурированные данные.

2. Вывести созданный файл на экран.
3. Удалить из файла данные в соответствии с вариантом.
4. Добавить в файл данные в соответствии с вариантом.
5. Вывести измененный файл на экран.
6. Используя ввод-вывод в стиле С++ создать файл и записать в него структурированные данные.
7. Вывести созданный файл на экран.
8. Удалить из файла данные в соответствии с вариантом.
9. Добавить в файл данные в соответствии с вариантом.
10. Вывести измененный файл на экран.

№ вариант а	Структура данных	Удаление	Добавление
1	Структура "Абитуриент": <ul style="list-style-type: none"> <li>- фамилия, имя, отчество;</li> <li>- год рождения;</li> <li>- оценки вступительных экзаменов (3);</li> <li>- средний балл аттестата.</li> </ul>	Удалить элемент с указанным номером.	Добавить К элементов в начало файла
2	Структура "Сотрудник": <ul style="list-style-type: none"> <li>- фамилия, имя, отчество;</li> <li>- должность</li> <li>- год рождения;</li> <li>- заработная плата.</li> </ul>	Удалить элемент с указанной фамилией	Добавить К элементов в конец файла
3	Структура "Государство": <ul style="list-style-type: none"> <li>- название;</li> <li>- столица;</li> <li>- численность населения;</li> <li>- занимаемая площадь.</li> </ul>	Удалить все элементы, у которых численность меньше заданной.	Добавить элемент с номером К
4	Структура "Человек": <ul style="list-style-type: none"> <li>- фамилия, имя, отчество;</li> <li>- домашний адрес;</li> <li>- номер телефона;</li> <li>- возраст.</li> </ul>	Удалить все элементы с заданным возрастом.	Добавить N элементов с номером К
5	Структура "Человек": <ul style="list-style-type: none"> <li>- фамилия, имя, отчество;</li> <li>- год рождения;</li> <li>- рост;</li> <li>- вес.</li> </ul>	Удалить все элементы с указанным ростом и весом.	Добавить К элементов в начало файла
6	Структура "Школьник": <ul style="list-style-type: none"> <li>- фамилия, имя, отчество;</li> <li>- класс;</li> <li>- номер телефона;</li> <li>- оценки по предметам (математика, физика, русский язык, литература).</li> </ul>	Удалить все элементы, у которых есть 2 хотя бы по одному предмету.	Добавить К элементов в конец файла

7	<p>Структура "Студент":</p> <ul style="list-style-type: none"> <li>- фамилия, имя, отчество;</li> <li>- домашний адрес;</li> <li>- группа;</li> <li>- рейтинг.</li> </ul>	Удалить все элементы, у которых рейтинг меньше заданного.	Добавить элемент с номером К
8	<p>Структура "Покупатель":</p> <ul style="list-style-type: none"> <li>- фамилия, имя, отчество;</li> <li>- домашний адрес;</li> <li>- номер телефона;</li> <li>- номер кредитной карточки</li> </ul>	Удалить К элементов из начала файла.	Добавить N элементов с номером К
9	<p>Структура "Пациент":</p> <ul style="list-style-type: none"> <li>- фамилия, имя, отчество;</li> <li>- домашний адрес;</li> <li>- номер медицинской карты;</li> <li>- номер страхового полиса.</li> </ul>	Удалить элемент с заданным номером медицинской карты.	Добавить К элементов в начало файла
10	<p>Структура "Информация":</p> <ul style="list-style-type: none"> <li>- носитель;</li> <li>- объем;</li> <li>- название;</li> <li>- автор.</li> </ul>	Удалить первый элемент с заданным объемом информации.	Добавить К элементов в конец файла
11	<p>Структура "DVD-диск":</p> <ul style="list-style-type: none"> <li>- название фильма;</li> <li>- режиссер;</li> <li>- продолжительность;</li> <li>- цена.</li> </ul>	Удалить все элементы с ценой выше заданной.	Добавить элемент с номером К
12	<p>Структура "DVD- диск":</p> <ul style="list-style-type: none"> <li>- название;</li> <li>- режиссер;</li> <li>- продолжительность;</li> <li>- цена.</li> </ul>	Удалить первый элемент с заданной продолжительностью .	Добавить N элементов с номером К
13	<p>Структура "Спортивная команда":</p> <ul style="list-style-type: none"> <li>- название;</li> <li>- город;</li> <li>- количество игроков;</li> <li>- количество набранных очков.</li> </ul>	Удалить все элементы с количеством очков меньше заданного.	Добавить К элементов в начало файла
14	<p>Структура "Стадион":</p> <ul style="list-style-type: none"> <li>- название;</li> <li>- адрес;</li> <li>- вместимость;</li> <li>- виды спорта.</li> </ul>	Удалить элемент с заданным названием.	Добавить К элементов в конец файла

15	<p>Структура "Автомобиль":</p> <ul style="list-style-type: none"> <li>- марка;</li> <li>- год выпуска;</li> <li>- цена;</li> <li>- цвет.</li> </ul>	Удалить все элементы, у которых год выпуска меньше заданного.	Добавить элемент с номером К
16	<p>Структура "Владелец автомобиля":</p> <ul style="list-style-type: none"> <li>- фамилия, имя, отчество;</li> <li>- номер автомобиля;</li> <li>- телефон;</li> <li>- номер техпаспорта.</li> </ul>	Удалить элемент с заданным номером.	Добавить N элементов с номером К
17	<p>Структура "Фильм":</p> <ul style="list-style-type: none"> <li>- название;</li> <li>- режиссер;</li> <li>- год выпуска;</li> <li>- стоимость.</li> </ul>	Удалить все элементы, у которых стоимость превышает заданную.	Добавить К элементов в начало файла
18	<p>Структура "Книга":</p> <ul style="list-style-type: none"> <li>- название;</li> <li>- автор;</li> <li>- год издания;</li> <li>- количество страниц.</li> </ul>	Удалить К элементов из начала файла.	Добавить К элементов в конец файла
19	<p>Структура "Фильм":</p> <ul style="list-style-type: none"> <li>- название;</li> <li>- режиссер;</li> <li>- страна;</li> <li>- приносимая прибыль.</li> </ul>	Удалить К элементов, начиная с номера N из файла.	Добавить элемент с номером К
20	<p>Структура "Государство":</p> <ul style="list-style-type: none"> <li>- название;</li> <li>- государственный язык;</li> <li>- денежная единица;</li> <li>- курс валюты относительно \$.</li> </ul>	Удалить элемент с указанным названием.	Добавить N элементов с номером К
21	<p>Структура "Автомобиль":</p> <ul style="list-style-type: none"> <li>- марка;</li> <li>- серийный номер;</li> <li>- регистрационный номер;</li> <li>- год выпуска.</li> </ul>	Удалить все элементы с указанной маркой	Добавить К элементов в начало файла
22	<p>Структура "Владелец автомобиля":</p> <ul style="list-style-type: none"> <li>- фамилия, имя, отчество;</li> <li>- номер автомобиля;</li> <li>- номер техпаспорта;</li> <li>- отделение регистрации ГАИ.</li> </ul>	Удалить элемент с заданным номером.	Добавить К элементов в конец файла

23	1. Структура "Стадион": <ul style="list-style-type: none"> <li>- название;</li> <li>- год постройки;</li> <li>- количество площадок;</li> <li>- виды спорта.</li> </ul>	Удалить все элементы, у которых год постройки меньше заданного.	Добавить элемент с номером К
24	Структура "Студент": <ul style="list-style-type: none"> <li>- фамилия, имя, отчество;</li> <li>- номер телефона;</li> <li>- группа;</li> <li>- оценки по 3 основным предметам.</li> </ul>	Удалить все элементы из группы с указанным номером, у которых среднее арифметическое оценок меньше заданного.	Добавить N элементов с номером К
25	Структура "Студент": <ul style="list-style-type: none"> <li>- фамилия, имя, отчество;</li> <li>- дата рождения;</li> <li>- домашний адрес;</li> <li>- рейтинг.</li> </ul>	Удалить все элементы с указанным рейтингом	Добавить N элементов с номером К

## 5. Содержание отчета

1. Постановка задачи (общая и для конкретного варианта).
2. Определения функций для реализации поставленных задач.
3. Определение функции main().
4. Содержимое исходного файла
5. Содержимое модифицированного файла.