Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«Севастопольский государственный университет»

Методические указания
к лабораторным работам по дисциплине
«Алгоритмизация и программирование»
для студентов дневной и заочно форм обучения направлений
09.03.02 – «Информационные системы и технологии»
и 09.03.03 – «Прикладная информатика»

часть 1

Севастополь

2021

УДК 004.42 (075.8)

Методические указания к лабораторным работам по дисциплине «Алгоритмизация и программирование» для студентов дневной и заочной форм обучения направлений 09.03.02 – «Информационные системы и технологии» и 09.03.03 – «Прикладная информатика», часть 1 / Сост. В. Н. Бондарев, Т. И. Сметанина, А. К. Забаштанский., А.Ю. Абрамович – Севастополь: Изд-во СевГУ, 2021. – 104 с.

Методические указания предназначены для проведения лабораторных работ и обеспечения самостоятельной подготовки студентов по дисциплине «Алгоритмизация и программирование». Целью методических указаний является обучение студентов основным принципам структурного программирования, навыкам разработки программ на языках Java и C/C++.

Методические указания составлены в соответствии с требованиями программы дисциплины «Алгоритмизация и программирование» для студентов направлений 09.03.02 – «Информационные системы и технологии» и 09.03.03 – «Прикладная информатика « и утверждены на заседании кафедры «Информационные системы» протокол № ____

Допущено научно-методической комиссией института информационных технологий и систем управления в технических системах в качестве методических указаний.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цели и задачи лабораторных работ

Основная цель выполнения настоящих лабораторных работ – получение практических навыков создания и отладки программ на языке C/C++ с использованием сложных типов данных. В результате выполнения лабораторных работ студенты должны углубить знания основных теоретических положений дисциплины «Алгоритмизация и программирование», решая практические задачи на ЭВМ.

Студенты должны получить практические навыки работы в интегрированной системе программирования Dev-C++ (или Eclipse CDT), навыки разработки программ, использующих текстовые и типизированные файлы, структуры, динамические списковые и древовидные структуры данных.

Выбор вариантов и график выполнения лабораторных работ

Варианты заданий приведены в каждой лабораторной работе и уточняются преподавателем.

Лабораторная работа выполняется в два этапа. На первом этапе – этапе самостоятельной подготовки – студент должен выполнить следующее:

– проработать по конспекту и рекомендованной литературе, приведенной в конце настоящих методических указаний, основные теоретические положения лабораторной работы и подготовить ответы на контрольные вопросы;

– разработать алгоритм решения задачи и составить схему алгоритма;

– описать схему алгоритма;

– написать программу на языке C/C++ и описать ее;

– оформить результаты первого этапа в виде заготовки отчета по лабораторной работе.

На втором этапе, выполняемом в лабораториях кафедры, студент должен:

– отладить программу;

– разработать и выполнить тестовые примеры.

Выполнение лабораторной работы завершается защитой отчета. Студенты должны выполнять и защищать работы **строго по графику**.

График уточняется преподавателем, ведущим лабораторные занятия.

Требования к оформлению отчета

Отчёты по лабораторной работе оформляются каждым студентом индивидуально. Отчёт должен включать: название и номер лабораторной работы; цель работы; вариант задания и постановку задачи; результаты выполнения программы; выводы по работе; приложения. Содержание отчета указано в методических указаниях к каждой лабораторной работе.

ЛАБОРАТОРНАЯ РАБОТА №1
«ОСНОВЫ РАБОТЫ В ОПЕРАЦИОННОЙ СИСТЕМЕ WINDOWS»

1.1 Цель работы

Изучение основных приемов управления компьютером средствами операционной системы WINDOWS, исследование средств операционной системы для работы с файлами.

1.2 Краткие теоретические сведения

1.2.1 Основные понятия ОС

ОС представляет комплекс системных программных средств. Основная функция ОС – посредническая. Она заключается в обеспечении взаимодействия пользователя и программно-аппаратных средств компьютера (*интерфейс пользователя*), взаимодействия между программным и аппаратным обеспечением (*аппаратно-программный интерфейс*), взаимодействия между разными видами программного обеспечения (*программный интерфейс*).

Взаимодействие пользователя и ОС может происходить с помощью ввода команд через *интерфейс командной строки*. Основным устройством управления в данном случае является клавиатура. Управляющие команды вводят в командную строку, где их можно редактировать. Исполнение команды начинается после ее утверждения, например, нажатием клавиши Enter. Для персональных компьютеров интерфейс командной строки обеспечивается ОС MS DOS.

Графические ОС реализуют более сложный тип интерфейса, основанный на взаимодействии активных и пассивных элементов управления, отображаемых на экране. В качестве активного элемента управления выступает *указатель мыши*. В качестве пассивных элементов выступают экранные кнопки, значки, флажки, строки меню и др. К графическим ОС относится, например, ОС Windows.

Все современные ОС обеспечивают создание файловой системы, предназначенной для хранения данных на дисках и обеспечения доступа к ним. Сведения о том, в каком месте диска записан тот или иной файл, хранятся в специальных *таблицах размещения файлов* (FAT-таблицах). Данные о месте положения файлов представляются пользователю для выполнения операций с файлами. К ним относят следующие операции:

- создание файлов и присвоение им имен;
- создание каталогов (папок) и присвоение им имен;
- переименование файлов и каталогов (папок);
- копирование и перемещение файлов;
- удаление файлов и каталогов;
- навигация по файловой структуре с целью доступа к заданному файлу, каталогу.

Файл – это именованный набор данных на постоянном носителе [4]. Создание файла состоит в присвоении ему имени и регистрации его в файловой системе. По способам именования файлов различают «короткое» и «длинное» имя. В MS DOS используется *соглашение 8.3*. В этом случае имя файла состоит из двух частей: *имени* (8 символов) и *расширения имени* (3 символа). Как имя, так и расширение могут включать только алфавитно-цифровые символы латинского алфавита. Имена файлов, записанные в соответствии с *соглашением 8.3*, считаются короткими. В ОС Windows 95 было введено длинное имя (256 знаков). Длинное имя может содержать любые символы, кроме 9 специальных: / \ : * ? ″<> | . Рекомендуется не использовать пробелы в именах файлов, а заменять их символами подчеркивания. Нежелательно в корневой папке диска хранить файлы с длинными именами, так как в этой папке ограничено количество единиц хранения. При этом, чем длиннее имена, тем меньше файлов можно разместить в корневой папке.

Расширение имени файла указывает, к какому типу относятся данные файла, и в каком формате они записаны. Так, расширение .txt соответствует текстовому файлу, .exe, .com – исполняемым файлам программ, .sys – системным файлам, .pas, .c – файлам, содержащим программы на языках Паскаль, Си.

Каталог (директория) – папка на диске, в которой хранятся файлы. Каталоги могут быть вложенными (папка в папке). Каталоги образуют иерархическую структуру, необходимую для удобного доступа к файлам. Верхним уровнем вложенности иерархической структуры является *корневой каталог* диска. Для доступа к файлу указывается путь, ведущий от корневого каталога к файлу. *Путь* – это последовательность имен каталогов, указывающая местоположение файла на диске. При записи пути доступа к файлу *указывают полное имя* и все промежуточные каталоги, разделяемые символом ‘\’, и имя файла, например:

Рисунок 1.1 – Структура пути к файлу

1.2.2 Основные приемы управления ОС

1.2.2.1 Управление ОС с помощью команд MS DOS

Запустить интерпретатор командной стоки, встроенной в Windows, можно двумя способами:

- через быстрое меню (правой клик на кнопке «Пуск», выбрать «Command prompt» или «Командная строка»);
- через диалог «Выполнить» (ввести команду cmd).

Основные команды ОС MS DOS приведены в таблице 1.1.

1.2.2.2 Управление Windows с помощью мыши

В Windows большую часть команд можно выполнить с помощью мыши. С мышью связан активный элемент – *указатель мыши*. Указатель мыши можно позиционировать на *объектах* и *элементах управления* Windows, отображаемых на

экране. В исходном состоянии на экране Windows (рабочем столе) можно наблюдать несколько экранных значков и Панель задач. Значки – это графическое представление объектов Windows, а Панель задач – один из элементов управления Windows.

Таблица 1.1 – Основные команды операционной системы MS DOS

Команда	Параметры	Действие	Пример	Примечание
md	<каталог>	Создать новый каталог	md IVANOV	Создание в текущем каталоге каталога IVANOV
cd	<каталог>	Перейти в указанный каталог (или указать путь)	cd LAB1	Относительно текущего каталога каталог LAB1 становится текущим
	.. (две точки)	Перейти в каталог предыдущего уровня	cd ..	Каталог IVANOV становится текущим
	\	Перейти в корневую папку	cd \	Текущим каталогом становится корневой каталог
rd	<каталог>	Удалить каталог	rd LAB1	Удаление пустого каталога
dir	<каталог>	Просмотреть содержимое каталога	dir IVANOV	Просмотр содержимого каталога IVANOV
copy	con <имя файла>	Создать файл	copy con my.txt	Создание файла my.txt. Завершение <Ctrl>+Z
	<источник> <приемник>	Копировать файл	copy a.txt b.txt	Копирование из файла a.txt в файл b.txt
	<имя файла> con	Вывести файл на экран	copy my.txt con	На экране отобразится содержимое my.txt
del	<имя файла>	Удалить файл	del my.txt	Удаление файла my.txt
ren	<старое имя> <новое имя>	Переименовать файл	ren a.txt c.txt	Переименование файла «a.txt»в«c.txt»
notepad	<имя файла>	Редактировать файл	notepad c.txt	Редактирование файла c.txt с помощью редактора «Блокнот»
type	<имя файла>	Вывод файла на экран	type c.txt	Вывод файла c.txt на экран
cls		Очистка экрана	cls	Очистка экрана
help		Вызов справки	help	Перечень всех команд
	<имя команды>	Вызов справки по указанной команде	help dir	Описание команды dir и ее параметров

Основные приемы управления с помощью мыши:

– *щелчок* – быстрое нажатие и отпускание клавиши мыши;

– *двойной щелчок* – два щелчка, выполненные с малым интервалом между ними;

– *перетаскивание* (drag-and-drop) – выполняется путем перемещения мыши при нажатой левой клавише (обычно сопровождается перемещением экранного объекта, на котором установлен указатель);

– *специальное перетаскивание* – выполняется, как и *перетаскивание*, но при нажатой правой кнопке;

– *зависание* – наведение указателя мыши на значок объекта или на элемент управления и задержка его на некоторое время (при этом на экране обычно появляется *всплывающая подсказка*).

1.2.2.3 Управление файловой структурой с помощью программы Проводник

Проводник (Explorer) – служебная программа, относящаяся к категории диспетчеров файлов. Она предназначена для навигации по файловой системе и ее обслуживания. Вызвать Проводник можно, щелкнув правой клавишей мыши на кнопке «Пуск», и в появившемся меню щелкнуть левой клавишей мыши на пункте «Проводник». Окно Проводника представлено на рисунке 1.2. Окно проводника имеет две рабочие области: левую панель, называемую *панелью папок*, и правую панель, называемую *панелью содержимого*.

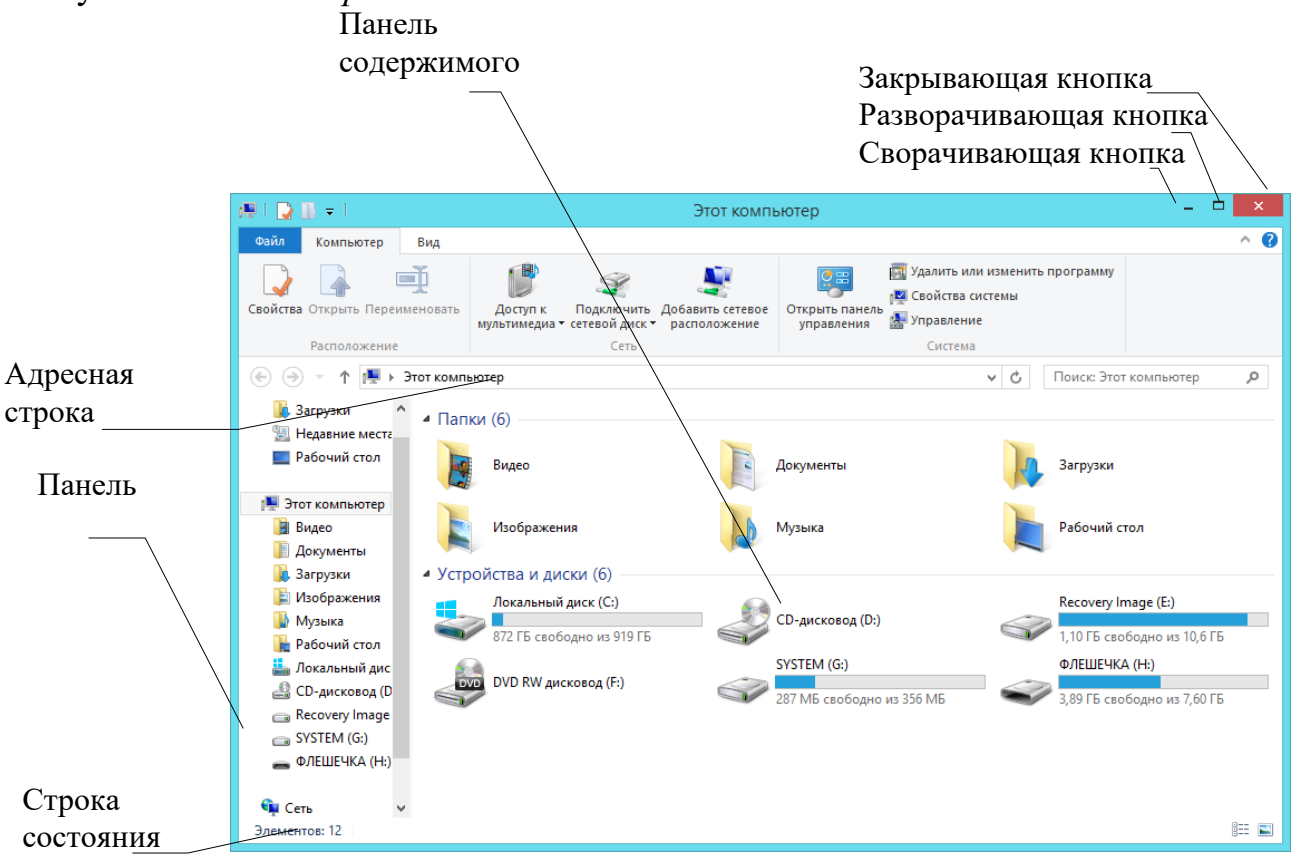

Рисунок 1.2 – Окно программы Проводник

Навигацию по файловой структуре выполняют на левой панели Проводника, на которой изображена структура папок. Папки могут быть *развернуты* или *свернуты*, а также *раскрыты* или *закрыты*. Свернутые папки отмечены знаком «+». Щелчок левой клавишей мыши разворачивает папку, при этом значок меняется на «–». Для то-

го, чтобы раскрыть папку, надо щелкнуть на ее значок. Содержимое раскрытой папки отображается на правой панели.

Запуск программы и открытие документов. Эта операция выполняется двойным щелчком на значке программы или документа на правой панели Проводника (панели содержимого).

Копирование и перемещение файлов и папок. Папку, из которой происходит копирование, называют *источником*; папку, в которую выполняется копирование, называют *приемником*. Копирование выполняется методом перетаскивания значка объекта с правой панели Проводника на левую панель.

Если папка-источник и папка-приемник принадлежат одному диску, то при перетаскивании выполняется перемещение, а если разным дискам, то копирование. Когда необходимо контролировать выполняемую операцию, делают специальное перетаскивание при нажатой правой кнопке мыши. В этом случае при отпускании правой кнопки появляется контекстное меню, в котором можно выбрать выполняемую операцию.

Удаление файлов и папок. На левой панели открыть папку, содержащую удаляемый объект, а на правой панели выделить мышью нужный объект (или группу объектов, удерживая клавишу Shift или Ctrl). Щелкнуть по кнопке Удалить на панели инструментов или нажать клавишу Delete на клавиатуре.

Использование буфера обмена. Система Windows создает область памяти, называемую *буфером обмена*. Принцип работы с буфером обмена очень прост и состоит в следующем:

а) открытие папки-источника и выделение щелчком нужного объекта;

б) *копирование* или *вырезка* объекта в буфер;

в) открытие папки-приемника и помещение в нее объекта из буфера обмена.

Три указанные операции (Копировать, Вырезать, Вставить) можно выполнить разными способами. Классический вариант состоит во входе в пункт меню «Правка» и выборе соответствующих подпунктов. Другой вариант – воспользоваться соответствующими кнопками панели инструментов. Самый эффективный способ – использовать комбинации клавиш клавиатуры:

CTRL + C (CTRL + INSERT) – копировать в буфер;

CTRL + X (SHIFT + DELETE) – вырезать в буфер;

CTRL + V (SHIFT + INSERT) – вставить из буфера.

Через буфер обмена можно переносить блоки текстов из одного документа в другой, можно переносить иллюстрации, файлы, папки, а также многие другие объекты.

1.2.2.4 Управление Windows с помощью диалога «Выполнить»

Диалог «Выполнить» представляет собой инструмент, интегрированный в операционную систему Windows. Он обеспечивает быстрый доступ к программам, папкам, документам и другим ресурсам системы (рисунок 1.3).

Диалог «Выполнить» не имеет ничего общего с командной строкой Windows, но может использоваться для ее запуска. Чаще всего его используют для доступа к элементам в панели управления, для этого необходимо знать соответствующие команды.

Самый простой и быстрый способ запустить диалог «Выполнить» – использовать сочетание клавиш Win + R.

В Windows 7 и предыдущих версиях ОС можно добавить команду «Выполнить» в меню «Пуск», а в Windows 8.1 существует функция в меню WinX, которая открывается нажатием сочетания клавиш Win + X или щелчком правой кнопки мыши на кнопке «Пуск».

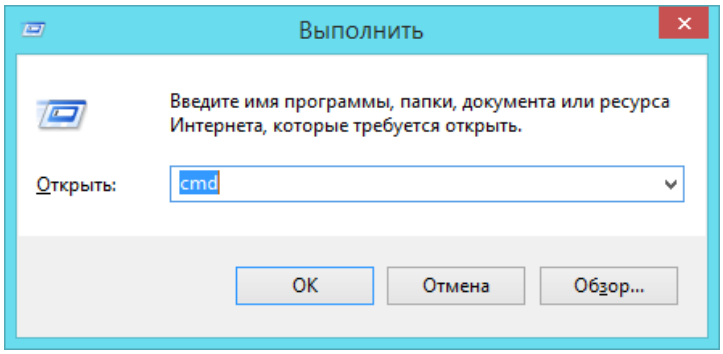

Рисунок 1.3 – Диалог «Выполнить»

Как правило, многие программы могут быть запущены вводом имени приложения. Например – «firefox», «excel» или «mspaint». Однако не все команды интуитивны. Например, чтобы открыть Word, необходимо ввести команду «winword».

В таблице 1.2 приведены примеры команд.

Таблица 1.2 – Основные команды диалога «Выполнить»

Команда	Действие
explorer	открытие проводника Windows
shutdown	завершение работы компьютера
logoff	выход из системы
shutdown -r	перезагрузка
control	открытие панели управления
taskmgr	диспетчер задач
osk	запуск экранной клавиатуры
desk.cpl	окно свойств экрана
controlkeyboard	окно свойств клавиатуры
controlmouse	окно свойств мыши
controlprinters	окно свойств принтеров
appwiz. cpl	быстрый доступ в меню добавления/удаления программ
cleanmgr	запуск инструмента «Очистка диска»
%appdata%	открытие папки с данными установленных приложений
cmd	запуск командной строки
regedit	запуск редактора реестра
msinfo32	запуск программы «Сведения о системе»

1.3 Порядок выполнения работы

1.3.1 На рабочем столе создать структуру папок по рисунку 1.4.

1.3.2 Создать ярлык к диску «C:» на рабочем столе.

1.3.3 Открыть «Блокнот» [4] и создать текст, содержащий четыре строчки по шаблону:

Студент: Фамилия Имя Отчество;
Курс, Номер группы;
Дата: День Месяц Год;
Время: Часы Минуты.

1.3.4 Сохранить созданный текстовый файл во вложенную папку «Лр1» под именем «ПерсональныеДанные.txt». Скопировать этот файл во вложенную папку «Лр2» и переименовать его как «Архив.txt».

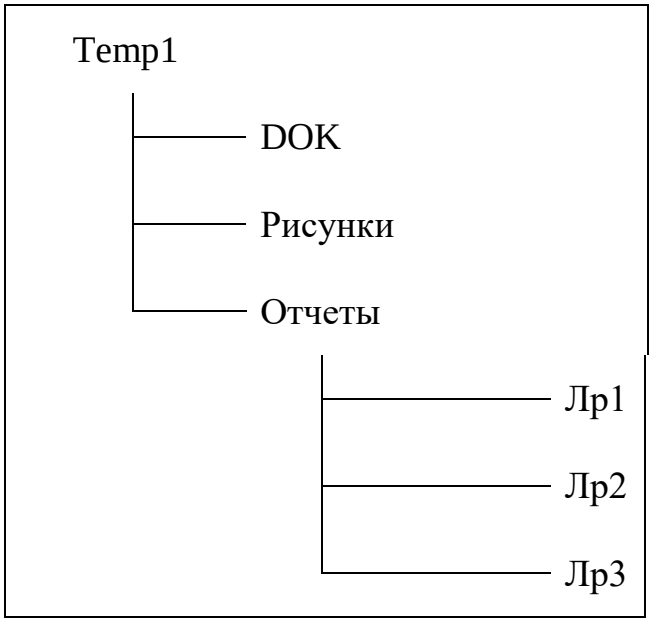

Рисунок 1.4 – Структура папок

1.3.5 С помощью редактора «Paint» создать рисунок [5], как можно более похожий на рисунок 1.5.

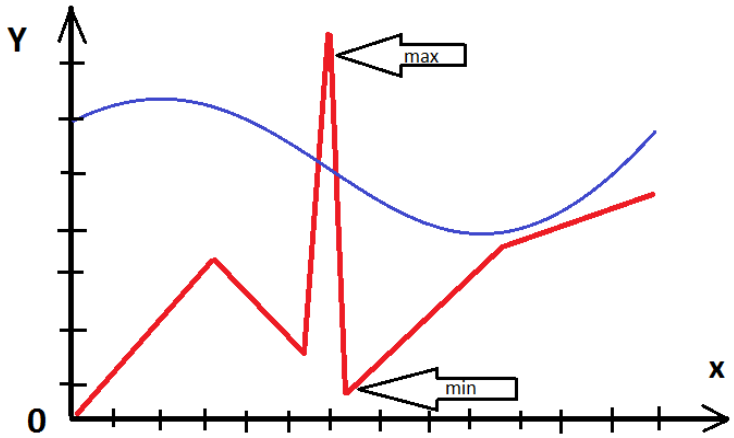

Рисунок 1.5 – Графики

1.3.6 Сохранить графический файл в папку «Рисунки» под именем «ЛабораторнаяРабота_1.bmp». Скопировать этот файл во вложенную папку «Лр1» и переименовать как «Архив.bmp».

1.3.7 Сохранить рисунок в разных форматах и проанализировать занимаемый объем в разных форматах; результаты анализа отобразить в отчете в виде таблицы с полями: расширение, размер.

1.3.8 Используя командное окно и команды MS DOS выполнить следующее:

- создать новые папки «P1» и «P2» в папке «DOK»;
- создать новый файл «text1.txt» в папке «P1» и скопировать его в папку «P2»;
- переименовать файл «text1.txt» в папке «P1» на новое имя «text2.txt»;
- удалить созданные файлы и папки.

1.3.9 Запустить стандартные программы, используя окно «Выполнить»

Открыть стандартные программы «Калькулятор» и «Блокнот». В «Калькуляторе» вычислить значения выражений:

- $\cos 90^{\circ}$;
- 5^{-3};
- 10!;
- $2/\pi$.

Результаты последовательно скопировать в «Блокнот», разместив их на разных строчках. Полученный текст сохранить в файле «Вычисления.txt» в папку «DOK».

1.3.10 Сделать скриншот экрана с помощью кнопки «Print Screen» или комбинацией клавиш «Alt»+«Print Screen». Вставить в «Paint». Отредактировать так, чтобы осталась только панель задач. Сохранить под именем «ПанельЗадач.bmp» в папку «Рисунки».

1.3.11 Открыть стандартную программу «WordPad» [4]. Выбрать один из контрольных вопросов, набрать сам вопрос, соблюдая такое же форматирование текста, и дать на него ответ. Сохранить в папке «DOK» под именем «ЛабораторнаяРабота_1.rtf». Выделить набранный текст и через буфер обмена скопировать в стандартную программу «Блокнот» («Notepad») [4]. Сравнить и найти отличия в визуальном представлении документов. Результаты анализа отобразить в отчете.

1.3.12 Открыть проводник и окна с выполненными заданиями. Предъявить результаты выполнения преподавателю.

1.3.13 Сделать резервную копию работы (файлы и папки) на индивидуальный носитель, затем удалить их с учебного компьютера.

1.3.14 Оформить отчет и защитить работу перед преподавателем.

1.4 Содержание отчета

Цель работы, постановка задачи, способы работы с файлами в программе Проводник и при помощи команд MS DOS, выполнение работы с иллюстрацией полученных результатов в виде скриншотов, выводы по работе (сравнительный анализ размеров графического файла в разных форматах, различие возможностей встроенных программ «Notepad» и «WordPad», удобство рассмотренных способов работы с файлами).

1.5 Контрольные вопросы

1.5.1 Назначение и функции операционной системы.
1.5.2 Файлы и имена файлов.
1.5.3 Каталоги и путь доступа к файлу.
1.5.4 Основные команды MS DOS и примеры их использования.
1.5.5 Различие функциональности стандартных программ WordPad и Notepad.
1.5.6 Особенности форматов хранения файлов.
1.5.7 Назовите и объясните основные приемы работы с мышью.
1.5.8 Объясните структуру окна программы Проводник.
1.5.9 Как выполнить создание файла и папки в программе Проводник?
1.5.10 Как выполнить копирование и перемещение файла в программе Проводник?
1.5.11 Как выполнить удаление файла в программе Проводник?
1.5.12 Как использовать буфер обмена для копирования группы файлов?
1.5.13 Приведите примеры команд диалога «Выполнить».

ЛАБОРАТОРНАЯ РАБОТА №2
«ПРОГРАММИРОВАНИЕ ЛИНЕЙНЫХ И РАЗВЕТВЛЯЮЩИХСЯ АЛГОРИТМОВ»

2.1 Цель работы

Изучение структуры С-программы.

Формирование навыков программирования алгоритмов линейной и разветвляющейся структуры на языке С.

Исследование особенностей ввода-вывода значений стандартных типов в языках С/С++.

2.2 Краткие теоретические сведения

Для выполнения лабораторной работы необходимо внимательно изучить структуру С-программы, основные типы данных и операторы языка С, управляющие инструкции, позволяющие реализовывать программы линейной и разветвляющейся структуры, а также ознакомиться со средствами ввода-вывода языков С/С++ и математическими функциями, описанными в файле **<math.h>**.

2.2.1 Структура С-программы

Любая *С-программа* состоит из *функций* и *переменных*. Функции содержат *инструкции*, описывающие вычисления, которые необходимо выполнить, а переменные хранят значения, используемые в процессе этих вычислений. Любая программа начинает свои вычисления с первой инструкции функции `main`. Таким образом, каждая С-программа должна содержать функцию `main`. Эта функция может не иметь параметров, и тогда ее заголовок – `main()`. Если функция `main` имеет параметры, то эти параметры выбираются из *строки вызова* (*командной строки*).

Левая фигурная скобка (`{`) должна начинать *тело* каждой функции. Соответствующая *правая фигурная скобка* (`}`) должна заканчивать каждую функцию. Так что тело функции представляет собой *блок*, ограниченный фигурными скобками `{}`. Блок содержит *выражения*, заканчивающиеся *точкой с запятой* (`;`), которые в языке С называют *инструкциями*. Общая структура функции `main` такова:

```
main()
{   инструкция_1
    инструкция_2
    ............
    инструкция_n
}
```

Помимо функции `main` в тексте программы могут располагаться определения *вспомогательных функций*.

В системах программирования С перед началом этапа компиляции выполняется программа предварительной *(препроцессорной)* обработки. Эта программа подчиняется специальным командам *(директивам препроцессора)*, которые указывают, что в программе перед ее компиляцией нужно выполнить определенные преобразования. Обычно эти преобразования состоят во включении других текстовых файлов в файл, подлежащий компиляции, и выполнении различных текстовых замен. Так, например, появление директивы

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

приводит к тому, что *препроцессор* подставляет на место этой директивы текст файла `<stdio.h>`, т. е. происходит включение информации о *стандартной библиотеке ввода-вывода*.

2.2.2 Переменные и основные типы данных языка С

В С любая *переменная* должна быть описана раньше, чем она будет использована; обычно все переменные описываются в начале функции перед первой исполняемой инструкцией. В *декларации (описании)* объявляются свойства переменных. Декларация состоит из названия типа и списка переменных, например:

```
int width, height;
float distance;
int exam_score;
char c;
```

В первом описании имеется *список переменных*, содержащий два имени (`width` и `height`). Обе переменные описываются как целые (`int`). Целочисленная переменная `exam_score` описана отдельно, хотя ее можно добавить к первому списку целых переменных.

Переменные можно инициализировать в месте их описания, например:

```
float distance=15.9;
```

Здесь переменной `distance` присваивается начальное значение `15.9`.

Имя переменной – это любая последовательность символов, содержащая *буквы*, *цифры* и *символы подчеркивания* (_), которая не начинается с цифры. Язык С *чувствителен к регистру* – прописные и строчные буквы различаются.

В С существует всего несколько *базовых типов*:

`char` – единичный байт, который может содержать одну литеру;

`int` – целое;

`float` – число с плавающей точкой одинарной точности;

`double` – число с плавающей точкой повышенной точности.

Имеется также несколько *квалификаторов*, которые можно использовать вместе с указанными базовыми типами. Например, квалификаторы `short` (короткий) и `long` (длинный) применяются к целым:

```
short int counter; long int amount;
```

В таких описаниях слово `int` можно опускать, что обычно и делается.

Квалификаторы `signed` (со знаком) и `unsigned` (без знака) можно применять к типу `char` и любому целому типу. Тип `long double` предназначен для арифметики с плавающей точкой повышенной точности.

Если операнды оператора принадлежат разным типам, то они *приводятся* к общему типу. Автоматически производятся лишь те преобразования, которые без какой-либо потери информации превращают операнды с меньшим диапазоном значений в операнды с большим диапазоном значений.

2.2.3 Основные операторы языка С

В таблице 2.1. показаны *приоритеты* и *порядок вычисления* всех операторов языка С. Операторы, перечисленные на одной строке, имеют одинаковый приоритет; строки упорядочены по убыванию приоритетов. *Унарные операторы* +, −, и * имеют более высокий приоритет, чем те же операторы в *бинарном* варианте.

Таблица 2.1 – Приоритеты и порядок вычисления операторов

Операторы	Порядок выполнения
`() [] -> .`	`слева направо`
`! ~ ++ -- + - * & (тип) sizeof`	`справа налево`
`*  /  %`	`слева направо`
`+  -`	`слева направо`
`<<  >>`	`слева направо`
`<  <=  >  >=`	`слева направо`
`==  !=`	`слева направо`
`&`	`слева направо`
`^`	`слева направо`
`|`	`слева направо`
`&&`	`слева направо`
`||`	`слева направо`
`?:`	`справа налево`
`= += -= *= /= %= &= ^= |= <<= >>=`	`справа налево`
`,`	`слева направо`

2.2.3.1 Арифметические операторы

К *арифметическим операторам* относятся: *сложение* (+), *вычитание* (−), *деление* (/), *умножение* (*) и *остаток* (%). Все операции (за исключением остатка) определены для переменных типа `int`, `char` и `float`. Остаток не определен для переменных типа `float`. Деление целых сопровождается отбрасыванием дробной части.

Все арифметические операции с плавающей точкой производятся над операндами *двойной точности* (`double`). После того, как получен результат с двойной точностью, он приводится к типу левой части выражения, если левая часть присутствует.

2.2.3.2 Операторы отношения

В языке С определены следующие *операторы отношения*: *проверка на равенство* (==), *проверка на неравенство* (!=), *меньше* (<), *меньше или равно* (<=), *больше* (>), *больше или равно* (>=).

Все перечисленные операторы вырабатывают результат целого типа (`int`). Если данное отношение между операндами ложно, то значение этого целого – ноль. Значения, отличные от нуля, интерпретируются как истинные.

2.2.3.3 Логические операторы

К *логическим операторам* относятся:

`&&` – *логическое И* (and);

`||` – *логическое ИЛИ* (or);

`!` – *логическое НЕ* (not).

Операндами логических операторов могут быть любые числа. Результат логического выражения – единица, если истина, и ноль, если ложь. Вычисление выражений, содержащих логические операторы, производится слева направо и прекращается, как только удается определить результат.

2.2.3.4 Операторы присваивания

К *операторам присваивания* относятся =, +=, −=, *= и /=, а также *префиксные* и *постфиксные* операторы ++ и – –. Все операторы присваивания присваивают переменной результат вычисления выражения.

В одном выражении оператор присваивания может встречаться несколько раз. Вычисления производятся *справа налево*. Например: `a=(b=c)*d;` вначале переменной `b` присваивается значение переменной `c`, затем выполняется операция умножения на `d`, и результат присваивается переменной `a`.

Типичный пример использования *многократного присваивания*:

```
a=b=c=d=e=f=0;
```

Операторы +=, −=, *=, /= являются *укороченной* формой записи оператора присваивания. Например:

```
a+=b; // a=a+b;          a*=b; // a=a*b;
a-=b; // a=a-b;          a/=b; // a=a/b;
```

Многие компиляторы генерируют код, который выполняется быстрее при использовании таких операторов присваивания.

Префиксные и постфиксные операторы присваивания ++ и −− используют для увеличения *(инкремент)* и уменьшения *(декремент)* на единицу значения переменной. Эти операторы можно использовать как в префиксной форме (помещая перед переменной, например, `++n`), так и в постфиксной форме (помещая после переменной: `n++`).

2.2.4 Основные средства ввода-вывода языков C/C++

2.2.4.1 Основные средства ввода-вывода языка C: функции printf и scanf

В языке C ввод-вывод данных реализуется с помощью *внешних функций*. Одними из наиболее универсальных и полезных функций ввода и вывода являются соответственно функции `scanf` и `printf`, описанные в головном файле `<stdio.h>`.

Функцию `printf` можно использовать для вывода любой комбинации символов, целых и вещественных чисел, строк, беззнаковых целых, длинных целых и беззнаковых длинных целых.

Типичный пример использования функции `printf`:

```
#include<stdio.h>
#include<windows.h> // для переключения кодовых страниц
main()
{
 SetConsoleCP(1251); // установка кодовой страницы
                     //win-cp 1251 в поток ввода
 SetConsoleOutputCP(1251); // установка кодовой страницы
                           //win-cp 1251 в поток вывода
int age=22; // возраст Эрика
 float income=534.7263; // доход Эрика
 printf("\nВозраст Эрика: %d. Его доход: $%.2f.\n",
           age,income);
 return 0;
}
```

Функция `printf` выводит на экран значения своих аргументов `age` и `income` в формате, который определяется *управляющей строкой*, являющейся первым параметром этой функции. Все символы этой строки, кроме *спецификаций формата* (`%d` и `%.2f`) и *управляющей последовательности* (`\n`), выводятся на экран без изменений.

Таким образом, при выполнении функции `printf` произойдет следующее. Последовательность символов `\n` переведет курсор на новую строку. В результате последовательность символов «`Возраст Эрика: `» будет выведена с начала новой строки. Смволы `%d` – это спецификация формата для целой переменной. Вместо `%d` будет подставлено значение переменной `age`. Далее будет выведена литеральная строка «`. Его доход: $`». `%.2f` – это спецификация формата для вещественной переменной, а также указание формата для вывода только двух цифр после десятичной точки. Вместо `%.2f` будет выведено значение переменной `income` в указанном формате. В заключение управляющая последовательность `\n` вызовет переход на новую строку.

Как видно из рассмотренного примера, спецификации формата помещаются внутри печатаемой строки. Вслед за этой строкой должен стоять ноль или более переменных, разделенных запятыми. Каждой спецификации в управляющей строке

функции printf должна соответствовать переменная адекватного типа. Если используется несколько спецификаций, то всем им должны соответствовать переменные того типа, который задается спецификацией.

Формально спецификацию формата можно определить следующим образом:
%[флаг][ширина][.точность][h|l|L]символ_формата,
где ширина – минимальное количество позиций, отводимых под выводимое значение, точность – количество позиций, отводимых под дробную часть числа.

Модификатор h указывает, что соответствующий аргумент должен печататься как short или unsigned short; l сообщает, что аргумент имеет тип long или unsigned long; L информирует, что аргумент принадлежит типу long double.

Возможные значения полей флаг и символ_формата приведены в таблицах 2.2 и 2.3.

Функция scanf служит для ввода данных. Подобно функции printf, scanf использует спецификацию формата, сопровождаемую списком вводимых переменных. Каждой вводимой переменной в функции scanf должна соответствовать спецификация формата.

Таблица 2.2 – Описание значений поля флаг

Флаг	Назначение
–	выравнивание по левому краю поля
+	печать числа всегда со знаком
Пробел	если первая литера – не знак, то числу должен предшествовать пробел

Таблица 2.3 – Описание значений поля символ_формата

Символ формата	Тип выводимого объекта
c	char; единичная литера
s	char *; строка
d,i	int; знаковая десятичная запись
o	int; беззнаковая восьмеричная запись
u	int; беззнаковое десятичное целое
x,X	int; беззнаковая шестнадцатеричная запись
f	double; вещественное число с фиксированной точкой
e,E	double; вещественное число с плавающей точкой
g,G	double; вещественное число в виде %f или %e в зависимости от значения
p	void *; указатель
%	знак процента %

Перед именами переменных следует ставить символ &. Этот символ означает *«взять адрес переменной»*. Ниже приведен пример программы, использующей функцию scanf:

```
#include<stdio.h>
#include<windows.h>
main()
{ int weight, // вес
      height; // рост
       SetConsoleCP(1251);
       SetConsoleOutputCP(1251);
  printf("Введите Ваш вес:\n"); scanf("%d",&weight);
  printf("Введите Ваш рост:\n"); scanf("%d",&height);
  printf("\nВаш вес = %d;\nВаш рост = %d.\n",
             weight,height);
  return 0;
}
```

Здесь `&weight` и `&height` – это адреса переменных `weight` и `height` соответственно.

2.2.4.2 Средства ввода-вывода языка C++: объекты cin и cout

В C++ вывод на экран выполняется с помощью *объекта стандартного потока вывода* `cout`, а ввод с клавиатуры осуществляется с помощью *объекта стандартного потока ввода* `cin`. Для использования этих объектов необходимо подключить головной файл `<iostream.h>`. Для обработки форматного ввода-вывода при помощи так называемых *манипуляторов потока* необходимо подключить головной файл `<iomanip.h>`.

Рассмотрим пример использования объекта `cout`:

```
#include <iomanip>
#include <iostream>
#include <windows.h>
using namespace std;
main()
{   int age=22; // возраст Эрика
    float income=534.726; // доход Эрика
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);
    cout<<'\n'<<"Возраст Эрика: "<<age<<'.'
        <<" Его доход: $"<<setprecision(2)<<fixed
         <<income<<'.'<<endl;
    return 0;
}
```

Вывод в поток выполняется с помощью *операции поместить в поток* <<. В рассматриваемом примере объекту `cout` с помощью операции << передаются значения, которые необходимо вывести на экран. Операция << является «более ин-

теллектуальной» по сравнению с функцией printf, так как определяет тип выводимых данных. Для вывода нескольких объектов операция << используется в *сцепленной форме*.

Манипулятор потока endl вызывает переход на новую строку и очищает *буфер вывода*, т. е. заставляет буфер немедленно вывести данные, даже если он полностью не заполнен. *Очистка (сброс)* буфера вывода может быть также выполнена манипулятором flush.

При выводе значения переменной income используется манипулятор потока setprecision. Использование setprecision(2) – это указание формата для вывода только двух цифр после десятичной точки. Поскольку манипулятор setprecision принимает параметр, он называется *параметризированным манипулятором потока*. Для вывода чисел с плавающей точкой в фиксированной форме используется манипулятор fixed.

Для установки ширины поля вывода может быть использован манипулятор потока setw, принимающий в качестве параметра число символьных позиций, в которые будет выведено значение.

Ввод потока осуществляется с помощью *операции взять из потока* >>. Эта операция применяется к объекту стандартного потока ввода cin. Рассмотрим пример использования объекта cin:

```
#include<iostream>
#include<windows.h>
using namespace std;main()
{   int weight, // вес
        height; // рост
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);
    cout<<"Введите Ваш вес:"<<endl;
    cin>>weight;
    cout<<"Введите Ваш рост:"<<endl;
    cin>>height;
    cout<<"\nВаш вес = "<<weight<<';'<<endl
        <<"Ваш рост = "<<height<<'.'<<endl;
    return 0;
}
```

Здесь считываемые значения размещаются в переменных weight и height. В процессе ввода последовательность символов, набранная на клавиатуре, преобразуется в необходимое *внутреннее представление* (в рассматриваемом примере целочисленное).

2.2.5 Управляющие инструкции языка C: if-else, условное выражение, switch

2.2.5.1 Инструкция `if-else`

Инструкция `if-else` используется для принятия решения. Ниже представлен ее синтаксис:

```
if (выражение) инструкция_1 else инструкция_2
```

причем `else`-часть может быть опущена. Сначала вычисляется `выражение`, и, если оно истинно, то выполняется `инструкция_1`. Если `выражение` ложно и существует `else`-часть, то выполняется `инструкция_2`. Необходимо отметить, что `else`-*часть всегда относится к ближайшей* `if`-*части*.

2.2.5.2 Условное выражение

Условное выражение, написанное с помощью *тернарного оператора* «`?:`», представляет собой другой способ записи инструкции `if-else`. В выражении

```
выражение1 ? выражение2 : выражение3
```

первым вычисляется `выражение1`. Если его значение отлично от нуля, то вычисляется `выражение2` и значение этого выражения становится значением всего условного выражения. В противном случае вычисляется `выражение3`, и его значение становится значением условного выражения. Таким образом, чтобы установить в `z` наибольшее из `a` и `b`, можно записать:

```
z = (a > b) ? a : b; // z=max(a,b)
```

2.2.5.3 Инструкция `switch`

Инструкция `switch` используется для выбора одного из многих путей. Она проверяет, совпадает ли значение выражения с одним из значений, входящих в некоторое множество целых констант, и выполняет соответствующую этому значению ветвь программы:

```
switch (выражение){
  case конст_выражение_1: последовательность_инструкций_1
  case конст_выражение_2: последовательность_инструкций_2
  ..................................................
  case конст_выражение_n: последовательность_инструкций_n
default: последовательность_инструкций_n+1
```

Константные выражения всех `case`-ветвей должны быть целочисленными и должны отличаться друг от друга.

Инструкция `switch` выполняется следующим образом. Вначале вычисляется `выражение`, и результат сравнивается с каждой `case`-константой. Если одна из `case`-констант равна значению выражения, управление переходит на последовательность инструкций с соответствующей `case`-меткой. Если ни с одной из `case`-констант нет совпадения, управление передается на последовательность инструкций с `default`-меткой, если такая имеется, в противном случае ни одна из после-

довательностей инструкций switch не выполняется. Ветви case и default можно располагать в любом порядке.

Для иллюстрации инструкции switch рассмотрим программу, которая определяет вид литеры, введенной пользователем с клавиатуры. Вид литеры – это цифра (digit), *пробельная литера* (white space) или другая литера (other). К пробельным литерам относят пробел (' '), *литеру табуляции* ('\t') и *литеру новая-строка* ('\n'). Ниже представлен текст программы.

```
#include<iostream>
using namespace std;
main()
// определение вида литеры, введенной с клавиатуры
// (цифра, пробельная литера или другая литера)
{   char c; // литера, вводимая с клавиатуры
    c=cin.get(); // ввод литеры с клавиатуры
    switch(c){
    case '0': case '1': case '2': case '3': case '4':
    case '5': case '6': case '7': case '8': case '9':
        // цифра
        cout<<"digit"<<endl; break;
    case ' ': case '\t': case '\n':
        // пробельная литера
        cout<<"white space"<<endl; break;
    default:
        // другая литера
        cout<<"other"<<endl; break;
    }
   return 0;
}
```

Литера, заключенная в одиночные кавычки, представляет целое значение, равное коду этой литеры (в кодировке, принятой на данной машине). Это так называемая *литерная константа*. Например, 'A' есть литерная константа; в наборе ASCII ее значение равняется 65. '\t' и '\n' обозначают коды литеры табуляции и литеры новая-строка соответственно.

Функция-элемент get объекта cin вводит одиночный символ из потока и возвращает этот символ в качестве *значения вызова функции*. Для вывода одиночного символа в поток используется функция-элемент put, которая в качестве аргумента принимает значение кода символа. Следует отметить, что стандартная библиотека ввода-вывода <stdio.h> включает функции для чтения и записи одной литеры getchar и putchar, аналогичные функциям-элементам get и put соответственно.

Инструкция break вызывает немедленный выход из инструкции switch. Поскольку выбор ветви case реализуется как переход на метку, то после выполнения

одной ветви `case`, если ничего не предпринять, программа «провалится вниз» на следующую ветвь. Инструкция `break` – наиболее распространенное средство выхода из инструкции `switch`.

2.2.6 Математические функции файла <math.h>

Ниже приведен перечень основных математических функций, описанных в головном файле <math.h>. Аргументы `x` и `y` функций имеют тип `double`; все функции возвращают значения типа `double`. Углы в тригонометрических функциях задаются *в радианах*.

`sin(x)` – *синус* x;
`cos(x)` – *косинус* x;
`tan(x)` – *тангенс* x;
`asin(x)` – *арксинус* x в диапазоне $[-\pi/2, \pi/2]$, $x \in [-1,1]$;
`acos(x)` – *арккосинус* x в диапазоне $[0, \pi]$, $x \in [-1,1]$;
`atan(x)` – *арктангенс* x в диапазоне $[-\pi/2, \pi/2]$;
`atan2(y,x)` – *арктангенс* y/x в диапазоне $[-\pi, \pi]$;
`sinh(x)` – *гиперболический синус* x;
`cosh(x)` – *гиперболический косинус* x;
`tanh(x)` – *гиперболический тангенс* x;
`exp(x)` – *экспоненциальная функция* e^x;
`log(x)` – *натуральный логарифм* $\ln(x)$, $x > 0$;
`log10(x)` – *десятичный логарифм* $\lg(x)$, $x > 0$;
`pow(x,y)` – x^y;
`sqrt(x)` – $\sqrt{x}$, $x \ge 0$;
`ceil(x)` – *наименьшее целое* в виде `double`, которое $\ge x$;
`floor(x)` – *наибольшее целое* в виде `double`, которое $\le x$;
`fabs(x)` – *абсолютное значение* $|x|$;
`fmod(x,y)` – *остаток от деления* x *на* y в виде числа с плавающей точкой.

2.2.7 Пример программы разветвляющейся структуры

Ниже представлен текст С-программы, вычисляющей значение функции:

$$z = f(x) = \begin{cases} \sin(2x + \pi/7), \text{ если } x \le a, \\ x^{3,21} + \tan(x), \text{ если } a < x < b, \\ \sqrt{x}\lg(x), \text{ если } x \ge b. \end{cases}$$

Значения параметров a, b и аргумента x вводятся с клавиатуры. Результаты вычислений выводятся на дисплей в *формате с плавающей точкой*.

```
#include <math.h>
#include <stdio.h>
#include <windows.h>
#define PI 3.14159265
main()
// вычисление значения функции z=f(x)
{
   float a, // параметр a
         b, // параметр b
         x, // аргумент x
         z; // значение функции z
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);
    // ввод значений параметров a, b и аргумента x
    printf("Введите значение параметра a: ");
    scanf("%f",&a);
    printf("Введите значение параметра b: ");
    scanf("%f",&b);
    printf("Введите значение аргумента x: ");
    scanf("%f",&x);
    // вычисление значения функции z
    if (x<=a) z=sin(2*x+PI/7);
    else if (x<b) // a<x<b
         z=pow(x,3.21)+tan(x);
    else // x>=b
         z=sqrt(x)*log10(x);
        // вывод значения функции z в формате
        //с плавающей точкой
    printf("Значение функции z = %e\n",z);
    return 0;
}
```

С помощью директивы

```
#define PI 3.14159265
```

оказывается возможным указать препроцессору, чтобы он при любом появлении идентификатора `PI` в тексте программы заменял его на значение `3.14159265` ($\approx\pi$).

Функция `clrscr` библиотеки `<conio.h>` выполняет очистку экрана и перемещение курсора в левый верхний угол.

Функция `getch` библиотеки `<conio.h>` используется для ввода символов с клавиатуры без их высвечивания на экране. С помощью `getch` можно считать коды основных клавиш (ASCII-коды) и *расширенные коды*. Расширенные коды – это коды верхнего ряда клавиш, коды правой части клавиатуры и коды комбинаций клавиш `Alt`, `Ctrl`, `Shift` с другими клавишами. В случае считывания расширенных кодов при первом обращении функция `getch` возвращает нулевое значение, а ее повторный вызов позволяет получить расширенный код.

Используя возможности языка C++, вышеприведенную программу можно переписать следующим образом:

```
#include <iostream>
#include <math.h>
using namespace std;
const double pi=3.14159265;
main()
{    // вычисление значения функции z=f(x)
    // объявление переменных
    //.................................
    // ввод значений параметров a, b и аргумента x
    cout<<"Введите параметр a: ";
    cin>>a;
    cout<<"Введите параметр b: ";
    cin>>b;
    cout<<"Введите аргумент x: ";
    cin>>x;
    // вычисление значения функции z
    //.................................
        // вывод значения функции z в формате
        //с плавающей точкой
    cout.setf(ios::scientific, ios::floatfield);
    cout<<"Значение функции z = f(x) = "<<z<<endl;
    cout<<"Нажмите любую клавишу...";
    return 0;
}
```

Как видно, в C++ вместо *символических констант*, которые создаются директивой препроцессора `#define`, имеется возможность использования *константных переменных*. Строка `const double pi=3.14159265;` использует *спецификацию* `const` для объявления *константы* `pi`, имеющей значение `3.14159265`. После определения константной переменной изменить ее значение нельзя. Следует отметить, что в C++ отдается предпочтение использованию константных переменных, а не символических констант. Константные переменные, в отличие от символических констант, являются данными определенного типа, и их имена видны отладчику.

Строка `cout.setf(ios::scientific,ios::floatfield);` устанавливает *экспоненциальный формат* вывода вещественных чисел (*формат с плавающей точкой*). Для отображения чисел в *формате с фиксированной точкой* следует записать `cout.setf(ios::fixed,ios::floatfield);` здесь функция-элемент `setf` объекта `cout` используется для управления установкой *флагов формата* вывода вещественных чисел `ios::scientific` или `ios::fixed`, которые содержатся в *статическом элементе данных* `ios::floatfield`.

2.3 Варианты заданий

Составить структурную схему алгоритма и написать две программы (на языке С и С++) вычисления функции $z = f(x)$. Варианты функций по указанию преподавателя выбирать из приведенных ниже. Значения параметров a, b и аргумента x вводятся с клавиатуры. Результаты вычислений выводятся на дисплей в *формате с плавающей точкой*. В первой программе для ввода-вывода использовать `scanf` и `printf`, а во второй `cin` и `cout`.

1) $z = \begin{cases} \sin(x) + x^2, если\ x <= a \\ \cos(x) + \sin(x), если\ a < x < b \\ \tan(x) + \cos(x), если\ x >= b \end{cases}$

2) $z = \begin{cases} e^x - \sin(x), если \quad x <= a \\ \cos(x) + |x|, если \quad a < x < b \\ x^5, если \quad x >= b \end{cases}$

3) $z = \begin{cases} e^x, если\ x <= a \\ e^x + \cos(x), если\ a < x < b \\ \tan(x), если\ x >= b \end{cases}$

4) $z = \begin{cases} 1.7 \cdot \sin(x), если \quad x <= a \\ \cos(x) + x^2, если \quad a < x < b \\ x^5, если \quad x >= b \end{cases}$

5) $z = \begin{cases} \ln(x) + \sin(x), если \quad x <= a \\ \ln(x) + \cos(x), если \quad a < x < b \\ \tan(x), если \quad x >= b \end{cases}$

6) $z = \begin{cases} e^x \cdot \sin(x), если \quad x <= a \\ \tan(x) + x^2, если \quad a < x < b \\ x^7, если \quad x >= b \end{cases}$

7) $z = \begin{cases} |x|, если \quad x <= a \\ |x| + \cos(x), если \quad a < x < b \\ \tan(x), если \quad x >= b \end{cases}$

8) $z = \begin{cases} 1.3 + \sin(x), если \quad x <= a \\ \tan(x) + x^2, если \quad a < x < b \\ x^7, если \quad x >= b \end{cases}$

9) $z = \begin{cases} \sin(|x|), если \quad x <= a \\ \cos(|x|), если \quad a < x < b \\ \tan(e^x), если \quad x >= b \end{cases}$

10) $z = \begin{cases} e^x, если \quad x <= a \\ \cos(x^2), если \quad a < x < b \\ \tan(x) + 8, если \quad x >= b \end{cases}$

11) $z = \begin{cases} x^2 + \sin(x), если \quad x <= a \\ \cos(x^2), если \quad a < x < b \\ \tan(x), если \quad x >= b \end{cases}$

12) $z = \begin{cases} \ln(x), если \quad x <= a \\ 1, если \quad a < x < b \\ e^x, если \quad x >= b \end{cases}$

13) $z = \begin{cases} |x| + \sin(x), если \quad x <= a \\ \cos(|x|), если \quad a < x < b \\ \tan(x), если \quad x >= b \end{cases}$

14) $z = \begin{cases} \ln(x) \cdot \sin(x), если \quad x <= a \\ x^2 \cdot \cos(x), если \quad a < x < b \\ x^5, если \quad x >= b \end{cases}$

15) $z = \begin{cases} \ln(x) + |x|, если \quad x <= a \\ x^3 \cdot \cos(x), если \quad a < x < b \\ x^4, если \quad x >= b \end{cases}$

16) $z = \begin{cases} e^x + |x|, если \quad x <= a \\ \tan(x) + x^2 + |x|, если \quad a < x < b \\ x^7, если \quad x >= b \end{cases}$

17) $z = \begin{cases} e^x - \sin(x), если \quad x <= a \\ \cos(x - 2.3), если \quad a < x < b \\ x^5, если \quad x >= b \end{cases}$

18) $z = \begin{cases} e^x \cdot \sin(x) - |x|, если \quad x <= a \\ \tan(x) + x^2 + |x|, если \quad a < x < b \\ x^7, если \quad x >= b \end{cases}$

19) $z=\begin{cases} e^x\cdot\sin(x), если \quad x<=a \\ \cos(x)+x^2, если \quad a<x<b \\ x^5, если \quad x>=b \end{cases}$

20) $z=\begin{cases} e^x-\sin(x), если \quad x<=a \\ \tan(x)\cdot x^2, если \quad a<x<b \\ x^7+|x|, если \quad x>=b \end{cases}$

21) $z=\begin{cases} \log_{10} x, если \quad x<=a \\ e^x, если \quad a<x<b \\ e^x/(3+\sin(x)), если \quad x>=b \end{cases}$

22) $z=\begin{cases} 1.3+\sin(x), если \quad x<=a \\ e^x/(3+\sin(x)), если \quad a<x<b \\ \cos(x)+x^2, если \quad x>=b \end{cases}$

23) $z=\begin{cases} e^x/(3+\sin(x)), если \quad x<=a \\ \cos(x)+x^2, если \quad a<x<b \\ 1.3+\sin(x), если \quad x>=b \end{cases}$

24) $z=\begin{cases} 4, если: x<=a \\ \cos(x)+x^2, если: a<x<b \\ \cos(x-2.3), если: x>=b \end{cases}$

25) $z=\begin{cases} e^x/(3+\sin(x)), если \quad x<=a \\ \tan(x)+x^2, если \quad a<x<b \\ x^7, если \quad x>=b \end{cases}$

26) $z=\begin{cases} \cos(x), если \quad x<=a \\ \ln(x)\cdot\sin(x), если \quad a<x<b \\ 0, если \quad x>=b \end{cases}$

27) $z=\begin{cases} e^x\cdot\sin(x), если \quad x<=a \\ \tan(|x|)+x^2, если \quad a<x<b \\ x^7, если \quad x>=b \end{cases}$

28) $z=\begin{cases} e^x/(3+\sin(x)), если \quad x<=a \\ \ln(x)+x^2, если \quad a<x<b \\ 1+\sin(-x), если \quad x>=b \end{cases}$

29) $z=\begin{cases} e^x\cdot\sin(x), если \quad x<=a \\ \tan(x)+\ln(x), если \quad a<x<b \\ x^7, если \quad x>=b \end{cases}$

30) $z=\begin{cases} x-2\cos^2(x), если \quad x<=a \\ \ln(x)\cdot\sin(x), если \quad a<x<b \\ 1.3+\sin(x), если \quad x>=b \end{cases}$

2.4 Порядок выполнения работы

2.4.1 Проработать необходимый теоретический материал, опираясь на сведения и указания, представленные в предыдущем пункте.

2.4.2 Ответить на контрольные вопросы.

2.4.3 Внимательно изучить постановку задачи.

2.4.4 Выполнить анализ *области определения* и *области значений* вычисляемой функции z. Желательным является построение графика функции.

2.4.5 Разработать структурную схему алгоритма решения задачи.

2.4.6 Разработать *тестовые примеры*, которые должны включать, по крайней мере, по два значения аргумента из каждого интервала кусочно-заданной функции z. Тестовые примеры должны также отражать поведение функции z в граничных точках.

2.4.7 Написать *две* программы вычисления функции z. Одна программа для ввода-вывода данных должна использовать функции `scanf` и `printf`, а другая – объекты `cin` и `cout`. Программы *обязательно* должны содержать комментарии.

2.4.8 Выполнить *тестирование и отладку* написанных программ, используя разработанные тестовые примеры. Особое внимание следует обратить на значения функции z в граничных точках.

2.4.9 Подготовить отчет по работе.

2.5 Содержание отчета

Цель работы; постановка задачи и вариант задания; краткие теоретические сведения; математическое обоснование задачи (включает анализ области определения и области значений функции, подлежащей вычислению, а также, факультативно, график этой функции); структурная схема алгоритма решения задачи; тестовые примеры; тексты программ; результаты тестирования и отладки программ; выводы.

2.6 Контрольные вопросы

2.6.1 Опишите структуру С-программы.
2.6.2 В чем состоит препроцессорная обработка программы?
2.6.3 Перечислите и охарактеризуйте базовые типы языка С.
2.6.4 Перечислите и охарактеризуйте квалификаторы языка С.
2.6.5 Что такое приведение типа?
2.6.6 Перечислите и охарактеризуйте арифметические операторы.
2.6.7 Перечислите и охарактеризуйте операторы отношения.
2.6.8 Перечислите и охарактеризуйте логические операторы.
2.6.9 Перечислите и охарактеризуйте операторы присваивания.
2.6.10 Опишите средства ввода-вывода языка С.
2.6.12 Опишите средства ввода-вывода языка С++.
2.6.22 Опишите инструкцию if-else.
2.6.23 Что такое условное выражение?
2.6.14 Опишите инструкцию switch.
2.6.15 Опишите математические функции файла <math.h>.

ЛАБОРАТОРНАЯ РАБОТА №3
«ПРОГРАММИРОВАНИЕ АЛГОРИТМОВ ЦИКЛИЧЕСКОЙ СТРУКТУРЫ»

3.1 Цель работы

Получение навыков программирования алгоритмов циклической структуры на языке C/C++. Исследование эффективности применения различных видов циклов в задаче табулирования функции.

3.2 Краткие теоретические сведения

3.2.1 Циклы в языках C/C++

Цикл представляет собой участок программы, повторяемый многократно. В языках C/C++ имеется три разновидности циклов: **while**, **do-while** и **for**.

3.2.1.1 Цикл while

Цикл `while` имеет следующий синтаксис:

```
while (<выражение>) <инструкция>
```

В цикле `while` вначале вычисляется `<выражение>`. Если его значение отлично от нуля (т. е. имеет значение истина), то выполняется `<инструкция>` и вычисление выражения повторяется. Этот цикл продолжается до тех пор, пока `выражение` не станет равным нулю (примет значение ложь), после чего вычисления возобновятся с точки, расположенной сразу за инструкцией.

Перед входом в цикл `while` обычно инициализируют одну или несколько переменных для того, чтобы `<выражение>` имело какое-либо конкретное значение. Инструкция или последовательность инструкций (*составная инструкция*), составляющих *тело цикла*, должны, как правило, изменять значения одной или нескольких переменных, входящих в `выражение`, чтобы за конечное число *итераций,* `выражение` приняло нулевое значение и цикл завершился. Цикл `while` – это *цикл с предусловием*; это значит, что решение о выполнении еще одной итерации цикла принимается *перед* началом цикла.

Цикл `while` завершается в следующих случаях:

- обращение в ноль выражения в *заголовке цикла*;
- выполнение в теле цикла инструкции `break`;
- выполнение в теле цикла инструкции `return`.

В первых двух случаях управление передается в точку, расположенную сразу за циклом. В третьем случае происходит выход из функции.

3.2.1.2 Цикл do-while

Цикл `do-while` имеет следующий синтаксис:

```
do
    <инструкция>
while (<выражение>);
```

В цикле `do-while` сначала выполняется `<инструкция>`, затем вычисляется `<выражение>`. Если оно истинно (отлично от нуля), то `инструкция` выполняется снова и т. д. Когда `<выражение>` становится ложным (обращается в ноль), цикл заканчивает работу. Цикл `do-while` завершается в тех же случаях, что и цикл `while`. Цикл `do-while` – это *цикл с постусловием*, т.е. проверка условия продолжения цикла (`<выражение>`) выполняется *после* каждого прохождения тела цикла (`<инструкция>`).

3.2.1.3 Цикл for

Цикл `for` имеет следующий синтаксис:

```
for (<выражение1>; <выражение2>; <выражение3>)
    <инструкция>
```

Каждое из трех выражений `<выражение1>`, `<выражение2>`, `<выражение3>` можно опускать, но точку с запятой опускать нельзя. При отсутствии *<выражения1>* или *<выражения3>* считается, что их нет в конструкции цикла; при отсутствии *<выражения2>* полагается, что его значение всегда истинно.

Обычно `<выражение1>` служит для инициализации *счетчика цикла*, `<выражение2>` – для выполнения проверки на окончание цикла, `<выражение3>` – для модификации счетчика цикла.

Инструкция `for` эквивалентна конструкции:

```
<выражение1>;
while (<выражение2>)
    {
      <инструкция>
      <выражение3>;
    }
```

В языке C++ переменную можно объявить в *части инициализации* (`выражение1`) инструкции `for`. Например:

```
for (int counter=1;counter<=10;counter++)
{     // тело цикла
    //.............
}
```

3.2.1.4 Инструкции break и continue

Инструкции `break` и `continue` изменяют поток управления. Когда инструкция `break` выполняется в инструкциях `switch`, `while`, `do-while` или `for`, происходит немедленный выход из соответствующей инструкции, и управление передается в точку, расположенную сразу за инструкцией. Обычное назначение инструкции `break` – досрочно прервать цикл или пропустить оставшуюся часть инструкции `switch`. Необходимо заметить, что инструкция `break` вызывает немедленный выход из *самого внутреннего* из объемлющих ее циклов.

Инструкция `continue` в циклах `while`, `do-while` или `for` вызывает пропуск оставшейся части тела цикла, после чего начинается выполнение следующей итерации цикла. В циклах `while` и `do-while` после выполнения инструкции `continue` осуществляется проверка условия продолжения цикла. В цикле `for` после выполнения инструкции `continue` выполняется *выражение приращения* (`выражение3`), а затем осуществляется проверка условия продолжения цикла (`выражение2`). Таким образом, инструкция `continue` вынуждает ближайший объемлющий ее цикл начать следующий шаг итерации.

3.2.1.5 Цикл for и оператор запятая

Иногда в цикле `for` <выражение1> и <выражение3> представляются как *списки выражений*, разделенных запятыми. В этом случае запятая используется как *оператор запятая* или *операция следования*, гарантирующая, что список выражений будет вычисляться *слева направо*. Оператор запятая имеет самый низкий приоритет (см. таблицу 2.1 в лабораторной работе №2). Значение и тип списка выражений, разделенных запятыми, равны значению и типу самого правого выражения в списке.

Оператор запятая наиболее часто применяется в цикле `for`. Этот оператор дает возможность использовать несколько выражений задания начальных значений и (или) несколько выражений приращения переменных, что позволяет вести несколько индексов (управляющих переменных) параллельно, например:

```
for (int left=0,right=n-1;left<right;left++,right--)
{     // тело цикла
     //.............
}
```

Кроме цикла `for`, оператор запятая можно использовать в тех случаях, когда последовательность инструкций мыслится как одна операция, например:

```
temp=a[i],a[i]=a[i+1],a[i+1]=temp; // обмен
```

3.2.2 Пример программы табулирования функции

3.2.2.1 Постановка задачи

Написать программу табулирования (печати таблицы значений) кусочно-заданной функции $z = f(x)$ на интервале от $x_{нач}$ до $x_{кон}$ с шагом Δx. Таблицу снабдить заголовком и шапкой. Вид функции определяется формулой:

$$z = f(x) = \begin{cases} a^2, \text{ если } x < 0, \\ ax, \text{ если } 0 \le x < 10, \\ 5a, \text{ если } x \ge 10. \end{cases}$$

Если $a > 10$, то значения функции должны выводиться в виде целых чисел. Значения параметра a, а также $x_{нач}$, $x_{кон}$ и Δx вводятся с клавиатуры. Результаты вычислений выводятся в формате с фиксированной точкой.

3.2.2.2 Описание алгоритма решения задачи в словесной форме

В *словесной форме* алгоритм решения задачи можно сформулировать следующим образом:

1) Ввести с клавиатуры исходные данные: a, $x_{нач}$, $x_{кон}$ и Δx.
2) Вывести заголовок и шапку таблицы.
3) Положить $x = x_{нач}$.
4) *Вычислить значение функции* z *по вышеприведенной формуле.*
5) *Если* $a > 10$*, то привести значение функции* z *к целому типу.*
6) *Вывести на экран строку таблицы значений функции.*
7) *Увеличить значение* x *на величину* Δx.
8) Если значение x не превышает $x_{кон}$, то перейти к пункту 4, иначе закончить выполнение программы.

Так как пункты 4 – 7 алгоритма выполняются многократно, то для их выполнения необходимо организовать цикл. Напишем два варианта программы с использованием циклов `while` и `for`.

3.2.2.3 Использование цикла while

Ниже представлена программа табулирования функции с использованием цикла `while`:

```
#include <stdio.h>
main()
// программа табулирования функции z=f(x)
// на интервале от xn до xk с шагом dx
{   float a, // параметр
          x, // аргумент функции z
          xn, // начальное значение аргумента x
          xk, // конечное значение аргумента x
          dx, // шаг
          z; // значение функции z

    // ввод a, xn, xk, dx
    printf("input a: "), scanf("%f",&a);
    printf("input xn: "), scanf("%f",&xn);
    printf("input xk: "), scanf("%f",&xk);
```

```
    printf("input the step dx: "), scanf("%f",&dx);
  // вывод заголовка и шапки таблицы
    printf(" _____________________________\n");
    printf("  |     x     |  z = f(x)   |\n");
    printf(" _____________________________\n");
    // табулирование функции z
    x=xn; // начальное значение аргумента x
    while (x<=xk)
  { // вывод строки таблицы
     printf("  |  %-9.3f|",x); // вывод аргумента x
        // вычисление значения функции z
      (x<0) ? (z=a*a) : ((x<10) ? (z=a*x) : (z=5*a));
        // вывод значения функции z
      if (a>10)
        printf("  %-10d|\n",(int)z); // целочисленное z
        else
        printf("  %-10.3f|\n",z); // вещественное z
      x+=dx; // приращение аргумента x
     }
    printf(" _____________________________\n");
    printf("press any key");
    return 0;
}
```

Поскольку формулы, определяющие функцию z, являются достаточно короткими, то вычисление значения функции z целесообразно осуществлять не с помощью инструкции `if-else`, а с помощью условного выражения.

Вывод таблицы значений функции осуществляется средствами функции `printf` стандартной библиотеки ввода-вывода `<stdio.h>`.

3.2.2.4 Использование цикла for

Ниже представлена программа табулирования функции с использованием цикла **for**:

```
#include <iomanip>
#include <iostream>
using namespace std;
main()
// программа табулирования функции z=f(x)
// на интервале от xn до xk с шагом dx
{    // объявление переменных и очистка экрана
    //.................................
  // ввод a, xn, xk, dx
    cout<<"Введите параметр a:  ", cin>>a;
    cout<<"Введите xn: ", cin>>xn;
```

```
    cout<<"Введите xk: ", cin>>xk;
    cout<<"Введите шаг dx: ", cin>>dx;
 // вывод заголовка и шапки таблицы
    cout<<"Таблица значений функции z=f(x)"<<endl
        <<"  _____________________________"<<endl
        <<"  |     x      |  z = f(x)   |"<<endl
        <<"  _____________________________"<<endl;

 // табулирование функции z
    cout.precision(3), cout.setf(ios::showpoint);
    cout.setf(ios::left,ios::adjustfield);
    cout.setf(ios::fixed,ios::floatfield);
 for(x=xn;x<=xk;x+=dx)
{ // вывод строки таблицы
 cout<<"  |  "<<setw(9)<<x<<'|'; // вывод аргумента x
 // вычисление значения функции z
 (x<0) ? (z=a*a) : ((x<10) ? (z=a*x) : (z=5*a));
       // вывод значения функции z
       cout<<"  "<<setw(10);
       // целочисленное или вещественное z
      (a>10) ? cout<<(int)z : cout<<z;
       cout<<'|'<<endl;
}
    cout<<"  _____________________________"<<endl;
    return 0;
}
```

В вышеприведенной программе ввод-вывод данных осуществляется с помощью объектов `cin` и `cout`. Последовательность инструкций

```
cout.precision(3), cout.setf(ios::showpoint);
cout.setf(ios::left,ios::adjustfield);
cout.setf(ios::fixed,ios::floatfield);
```

задает формат для вывода значения аргумента x и значения функции y в строке таблицы. Для установки ширины поля используется манипулятор потока `setw`. *Функция-элемент* объекта `cout` `precision` позволяет задать точность печатаемого вещественного числа (т. е. число разрядов справа от десятичной точки). Функция-элемент `setf` используется для управления *флагами состояния формата*.

Флаг `ios::showpoint` устанавливается для вывода чисел с обязательной печатью десятичной точки и нулевых младших разрядов (нулей в конце числа). Так, например, значение `79.0` без установки `ios::showpoint` будет напечатано как `79`, а с установкой `ios::showpoint` – как `79.00000` (количество нулей определяется заданной точностью).

Флаги `ios::left` и `ios::right` содержатся в *статическом элементе данных* `ios::adjustfield` и позволяют выравнивать печать соответственно по левой или правой границам поля.

Флаги `ios::fixed` и `ios::scientific`, управляющие форматом вывода вещественных чисел, описаны в лабораторной работе №1 настоящих методических указаний.

3.3 Варианты заданий

Вычислить и вывести на экран в виде таблицы значения функции $z = f(x)$ на интервале от $x_{нач}$ до $x_{кон}$ с шагом Δx. Таблицу снабдить заголовком и шапкой. Вид функции z выбирать в соответствии с вариантами задания к лабораторной работе №2 настоящих методических указаний. Значения параметров a, b, а также $x_{нач}$, $x_{кон}$ и Δx вводятся с клавиатуры. Результаты вычислений выводятся в формате с фиксированной точкой.

3.4 Порядок выполнения работы

3.4.1 Проработать необходимый теоретический материал, опираясь на сведения и указания, представленные в предыдущем пункте, а также в литературе [2, 3, 5].

3.4.2 Ответить на контрольные вопросы.

3.4.3 Внимательно изучить постановку задачи.

3.4.4 Выполнить анализ *области определения* и *области значений* вычисляемой функции z. Желательным является построение графика функции. Для выполнения этого пункта можно воспользоваться результатами предыдущей лабораторной работы.

3.4.5 Разработать структурную схему алгоритма решения задачи.

3.4.6 Разработать *тестовые примеры* (таблицы значений функции). При разработке тестовых примеров целесообразно использовать *отлаженную* программу из предыдущей лабораторной работы для вычисления значений функции z.

3.4.7 Написать *два* варианта программы табулирования функции z, используя циклы **while** и **for**. В одном из вариантов ввод-вывод должен базироваться на функциях **scanf** и **printf**, а в другом – на объектах **cin** и **cout**. Программы *обязательно* должны содержать комментарии.

3.4.8 Выполнить *тестирование и отладку* написанных программ, используя разработанные тестовые примеры.

3.4.9 Подготовить отчет по работе.

3.5 Содержание отчета

Цель работы; постановка задачи и вариант задания; краткие теоретические сведения; математическое обоснование задачи (включает анализ области определения и области значений функции, подлежащей вычислению, а также, факультативно, график этой функции); структурная схема алгоритма решения задачи; тестовые примеры; тексты программ; результаты тестирования и отладки программ; выводы.

3.6 Контрольные вопросы

3.6.1 Что называют циклом?
3.6.2 Какие три элемента должны входить в цикл?
3.6.3 Перечислите типы циклов в языках C/C++.
3.6.4 Охарактеризуйте цикл с предусловием.
3.6.5 Охарактеризуйте цикл с постусловием.
3.6.6 Охарактеризуйте цикл **while**.
3.6.7 Охарактеризуйте цикл **do-while**.
3.6.8 Охарактеризуйте цикл **for**.
3.6.9 Опишите инструкции **break** и **continue**.
3.6.10 В каких случаях целесообразно применять оператор запятая?
3.6.11 Какой тип цикла лучше использовать в задаче табулирования функции? Почему?
3.6.12 Дайте характеристику флагам состояния формата, используемым в функции **setf** объекта **cout**.

ЛАБОРАТОРНАЯ РАБОТА №4
«ПРОГРАММИРОВАНИЕ АЛГОРИТМОВ ОБРАБОТКИ ОДНОМЕРНЫХ СТАТИЧЕСКИХ МАССИВОВ»

4.1 Цель работы

Изучить способы представления массивов в памяти ЭВМ, получить практические навыки реализации алгоритмов обработки одномерных массивов.

4.2 Краткие теоретические сведения

Для выполнения лабораторной работы необходимо изучить особенности представления и обработки одномерных массивов в языках C/C++, а также алгоритмы сортировки массивов методами прямого обмена, вставки и прямого выбора.

4.2.1 Одномерные массивы

Одномерный массив – это набор объектов одинакового типа, расположенных в *последовательной* группе ячеек памяти, доступ к которым осуществляется *по индексу*. В языке С одномерные массивы описываются следующим образом:

```
тип имя_массива[размер_массива];
```

В результате такого объявления компилятор отводит под массив память размером `sizeof(тип)*размер_массива` байтов. Операция (функция) `sizeof`, операндами (аргументами) которой могут являться константы, типы и переменные, в качестве результата возвращает количество байт, занимаемых ее операндом (аргументом).

Используя имя массива и индекс, можно обращаться к элементам массива: `имя_массива[индекс]`. Все элементы массива индексируются, *начиная с нуля* и заканчивая величиной, на единицу меньшей, чем размер массива, указанный при его описании. Например, если массив `data` объявлен как

```
double data[245];
```

то его `245` элементами типа `double` будут: `data[0]`, `data[1]`, …, `data[244]`. Индекс массива может быть как целым числом, так и целочисленным выражением. Квадратные скобки, внутри которых записывается индекс массива, рассматриваются как *оператор индексации*. Оператор индексации имеет самый высокий приоритет (см. таблицу 2.1 в лабораторной работе №2).

Элементам массива можно присваивать начальные значения (*инициализировать* их) при объявлении массива с помощью списка начальных значений, разделенных запятыми, заключенного в фигурные скобки:

```
int n[10]={32,27,64,18,95,14,90,70,60,37};
```

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

ментам массива **n** можно присвоить нулевые начальные значения с помощью объявления

```
int n[10]={0};
```

Если размер массива не указан в объявлении со списком инициализации, то количество элементов массива будет равно количеству элементов в списке начальных значений. Например, объявление

```
int n[]={1,2,3,4,5};
```

создает массив из пяти элементов.

4.2.2 Поиск в массивах

Напишем программу, которая для целочисленного массива из 10 элементов определяет, сколько положительных элементов располагается между его максимальным и минимальным элементами.

Запишем алгоритм в общем виде.

1 Определить, где в массиве расположены максимальный и минимальный элементы, то есть найти их индексы.

2 Просмотреть все элементы, расположенные между ними. Если элемент массива больше нуля, увеличить счетчик на единицу.

Перед написанием программы полезно составить тестовый пример, чтобы более наглядно представить себе алгоритм решения задачи. Ниже представлен пример массива из 10 чисел и обозначены искомые величины (рисунок 4.1):

0	1	2	3	4	5	6	7	8	9	индексы
6	-8	15	9	-1	3	5	-10	12	2	элементы массива
		МАКС	+		+	+	МИН			

Рисунок 4.1 – Целочисленный массив из 10 элементов

Ясно, что порядок расположения элементов в массиве заранее не известен, и сначала может следовать как максимальный, так и минимальный элемент, кроме того, они могут и совпадать. Поэтому прежде чем просматривать массив в поисках количества положительных элементов, требуется определить, какой из индексов больше. Ниже представлен текст соответствующей программы:

```
#include<stdio.h>
#include<windows.h>
const int N=10;
int main(){
     SetConsoleOutputCP(65001);
     int x[N];
     printf("введите %d элементов массива\n",N);
     int i=0; // Ввод значений элементов массива
     for(;i<N;i++)
     scanf("%d",&x[i]);
```

```
          int imax=1, // Принимаем за максимум и минимум первый
элемент
              imin=1;
     //Выполняем поиск максимального и минимального элемента
               for(i=0;i<N;i++){
                    if(x[i]>x[imax]) imax=i;
                    if(x[i]<x[imin]) imin=i;
               }
     // отладочная печать
     printf("\n индекс максимального =%d максимальный элемент
=%d", imax,x[imax]);
     printf("\n индекс минильного =%d минимальный элемент =%d",
imin,x[imin]);
     /* определяем границы просмотра массива для поиска положи-
тельных элементов, находящихся между максимальным и минимальным
элементами*/
          int ibeg,iend;
          if (imin<imax) {ibeg=imin; iend=imax;}
           else {ibeg=imax; iend=imin;}
       // поиск положительных элементов
          int count=0;  // счетчик
           for(i=ibeg;i<iend;i++)
               if (x[i]>0) count++;
           printf("\n    количество    положительных    элементов
=%d",count);
          return 0;
     }
```

В программе после нахождения каждой величины вставлена отладочная печать. Рекомендуется не пренебрегать этим способом отладки.

Массив просматривается, начиная с элемента, следующего за максимальным (минимальным), до элемента, предшествующего минимальному (максимальному). Индексы границ просмотра хранятся в переменных `ibeg` и `iend`.

Тестовых примеров для проверки программы должно быть, по крайней мере, три:

1) `x[imin]` расположен левее `x[imax];`

2) `x[imin]` расположен правее `x[imax];`

3) `x[imin]` и `x[imax]` совпадают.

Последняя ситуация имеет место, когда в массиве все элементы имеют одно и то же значение. Желательно также проверить, как работает программа, если `x[imin]` и `x[imax]` расположены рядом, а также в начале и в конце массива. Элементы массива нужно задавать как положительные, так и отрицательные.

Выше был рассмотрен пример задачи поиска элементов в массиве. Другой распространенной задачей является сортировка массива, то есть упорядочение его элементов в соответствии с какими-либо критериями – чаще всего по возрастанию или убыванию элементов.

Рассмотрим простейшие алгоритмы сортировки массивов.

4.2.3 Сортировка массивов

Сортировка прямым обменом

Сортировка обменом – метод, при котором все соседние элементы массива попарно сравниваются друг с другом и меняются местами в том случае, если предшествующий элемент больше последующего. В результате этого максимальный элемент постепенно смещается вправо и, в конце концов, занимает свое крайне правое место в массиве, после чего он исключается из дальнейшей обработки. Затем процесс повторяется, и свое место занимает второй по величине элемент, который также исключается из дальнейшего рассмотрения. Так продолжается до тех пор, пока вся последовательность не будет упорядочена. Указанный метод сортировки иногда называют методом «пузырька».

Пусть, например, требуется провести сортировку массива:

30, 17, 73, 47, 22, 11, 65, 54

Ход сортировки изображен на рисунке 4.2.

1-шаг 17, 30, 47, 22, 11, 65, 54, 73
2-шаг 17, 30, 22, 11, 47, 54, 65, 73
3-шаг 17, 22, 11, 30, 47, 54, 65, 73
4-шаг 17, 11, 22, 30, 47, 54, 65, 73
5-шаг 11, 17, 22, 30, 47, 54, 65, 73

Рисунок 4.2 – Сортировка методом пузырька

Ниже представлен фрагмент программы сортировки массива методом «пузырька».

```
const int N=8;      // размер массива
int i,j,       // индексы
     a[N],     // массив из целых чисел
     temp;     // временная переменная
// сортировка
       for (j=N;j>1;j--)
          for(i=0;i<j-1;i++)// сравнение двух соседних элемен-
тов
               if(a[i]<a[i+1]){
                    temp=a[i];    // перестановка элементов
                    a[i]=a[i+1];
                    a[i+1]=temp;
                  }
```

Сортировка методом вставки

При сортировке вставками из неупорядоченной последовательности элементов поочередно выбирается каждый элемент, сравнивается с предыдущим (уже упорядоченным) списком и помещается на соответствующее место в последнем.

Сортировку вставками рассмотрим на примере следующей неупорядоченной последовательности элементов: {38, 12, 80, 15, 36, 23, 74, 62}.

Процесс сортировки иллюстрирует рисунок 4.2, где для каждого этапа в скобках указан очередной элемент, который включается в уже упорядоченную часть исходной последовательности.

1-й этап	38 |	(12)	80	15	36	23	74	62
2-й этап	12	38 |	(80)	15	36	23	74	62
3-й этап	12	38	80 |	(15)	36	23	74	62
4-й этап	12	15	38	80 |	(36)	23	74	62
5-й этап	12	15	36	38	80 |	(23)	74	62
6-й этап	12	15	23	36	38	80 |	(74)	62
7-й этап	12	15	23	36	38	74	80 |	(62)
	12	15	23	36	38	62	74	80

Рисунок 4.3 – Сортировка массива методом вставки

Ниже представлен фрагмент программы сортировки массива методом вставки.

```
const int N=8;      // размер массива
 int i,j,     // индексы
     a[N],    // массив из целых чисел
     temp;    // временная переменная
// сортировка
  for(i=1;i<N;i++){
        temp=a[i];
        for(j=i-1;(j>=0)&&(a[j]>temp);j--) a[j+1]=a[j];
        a[j+1]=temp;
      }
```

Сортировка прямым выбором

Сортировка выбором состоит в том, что сначала в неупорядоченной последовательности выбирается минимальный элемент. Этот элемент исключается из дальнейшей обработки, а оставшаяся последовательность элементов принимается за исходную; процесс повторяется до тех пор, пока все элементы не будут выбраны. Очевидно, что выбранные элементы образуют упорядоченную последовательность.

Ход сортировки изображен на рисунке 4.3.

Начальное состояние массива	8	23	5	65	44	33	1	6
Шаг 1	1 |	23	5	65	44	33	8	6
Шаг 2	1	5|	23	65	44	33	8	6
Шаг 3	1	5	6 |	65	44	33	8	23
Шаг 4	1	5	6	8 |	44	33	65	23
Шаг 5	1	5	6	8	23 |	44	65	33
Шаг 6	1	5	6	8	23	33 |	65	44
Шаг 7	1	5	6	8	23	33	44 |	65
	1	5	6	8	23	33	44	65

Рисунок 4.4 – Сортировка методом прямого выбора

Фрагмент соответствующей процедуры представлен ниже:

```
const int N=8;  // размер массива
 int i,j,       // индексы
     a[N],      // массив из целых чисел
     temp,      // временная переменная хранит минимум
     imin;      // индекс минимума
     // сортировка
       for (i=0;i<N;i++){
          imin=i; temp=a[i];
          for(j=i+1;j<N;j++)
               if(a[j]<temp){
                    imin=j;
                    temp=a[imin];
                    }
               a[imin]=a[i];
               a[i]=temp;
             }
```

4.3 Варианты заданий

Значение константы N (размера массива) студент выбирает самостоятельно.

Вариант 1

В одномерном массиве, состоящем из *N* вещественных элементов, вычислить:

1) сумму положительных элементов массива;

2) произведение элементов массива, расположенных между максимальным по модулю и минимальным по модулю элементами.

Упорядочить элементы массива по убыванию, используя алгоритм сортировки методом прямого обмена.

Вариант 2

В одномерном массиве, состоящем из *N* вещественных элементов, вычислить:

1) сумму отрицательных элементов массива;

2) произведение элементов массива, расположенных между максимальным по модулю и минимальным по модулю элементами.

Упорядочить элементы массива по убыванию, используя алгоритм сортировки методом вставки.

Вариант 3

В одномерном массиве, состоящем из *N* вещественных элементов, вычислить:

1) произведение элементов массива с четными номерами;

2) сумму элементов массива, расположенных между первым и последним нулевыми элементами.

Преобразовать массив таким образом, чтобы сначала располагались все неотрицательные элементы, а потом все отрицательные. Выполнить сортировку каждой части массива по возрастанию методом прямого выбора.

Вариант 4

В одномерном массиве, состоящем из *N* вещественных элементов, вычислить:

1) сумму элементов массива с нечетными номерами;

2) сумму элементов массива, расположенных между первым и последним отрицательными элементами.

Заменить все элементы, модуль которых не превышает 1, на 1 и упорядочить элементы массива по возрастанию, используя алгоритм сортировки методом прямого обмена.

Вариант 5

В одномерном массиве, состоящем из *N* вещественных элементов, вычислить:

1) максимальный элемент массива;

2) сумму элементов массива, расположенных до последнего положительного элемента.

Заменить все элементы, модуль которых находится внутри отрезка `[a, b]` на число `a` и упорядочить элементы массива по возрастанию, используя алгоритм сортировки методом вставки. Значение `a` и `b` вводит пользователь.

Вариант 6

В одномерном массиве, состоящем из *N* вещественных элементов, вычислить:

1) минимальный элемент массива;

2) сумму элементов массива, расположенных между первым и последним положительными элементами.

Преобразовать массив таким образом, чтобы сначала располагались все элементы, большие 1, а потом – все остальные. Упорядочить каждую часть массива по возрастанию, используя алгоритм сортировки методом прямого выбора.

Вариант 7

В одномерном массиве, состоящем из *N* целых элементов, вычислить:

1) номер максимального элемента массива;

2) произведение элементов массива, расположенных между первым и вторым нулевыми элементами.

Преобразовать массив таким образом, чтобы в первой его половине располагались элементы, стоявшие на нечетных позициях, а во второй половине – элементы, стоявшие на четных позициях. Упорядочить каждую часть массива по возрастанию, используя алгоритм сортировки методом прямого обмена.

Вариант 8

В одномерном массиве, состоящем из *N* вещественных элементов, вычислить:

1) номер минимального элемента массива;

2) сумму элементов массива, расположенных между первым и вторым отрицательными элементами.

Преобразовать массив таким образом, чтобы сначала располагались все элементы, модуль которых не превышает 1, а потом – все остальные. Упорядочить каждую часть массива по возрастанию, используя алгоритм сортировки методом вставки.

Вариант 9

В одномерном массиве, состоящем из *N* вещественных элементов, вычислить:

1) максимальный по модулю элемент массива;

2) сумму элементов массива, расположенных между первым и вторым положительными элементами.

Преобразовать массив таким образом, чтобы элементы, большие 1, располагались после всех остальных. Упорядочить каждую часть массива по убыванию, используя алгоритм сортировки методом прямого выбора.

Вариант 10

В одномерном массиве, состоящем из *N* целых элементов, вычислить:

1) минимальный по модулю элемент массива;

2) сумму модулей элементов массива, расположенных после первого элемента, равного нулю.

Преобразовать массив таким образом, чтобы в первой его половине располагались элементы, стоявшие в четных позициях, а во второй половине – элементы, стоявшие в нечетных позициях. Упорядочить каждую часть массива по возрастанию, используя алгоритм сортировки методом прямого обмена.

Вариант 11

В одномерном массиве, состоящем из *N* вещественных элементов, вычислить:

1) номер минимального по модулю элемента массива;

2) сумму модулей элементов массива, расположенных после первого отрицательного элемента.

Сжать массив, удалив из него все элементы, величина которых находится внутри отрезка `[a, b]`. Упорядочить элементы массива по возрастанию, используя алгоритм сортировки методом вставки. Значение `a` и `b` вводит пользователь.

Вариант 12

В одномерном массиве, состоящем из *N* вещественных элементов, вычислить:

1) номер максимального по модулю элемента массива;

2) сумму элементов массива, расположенных после первого положительного элемента.

Преобразовать массив таким образом, чтобы сначала располагались все элементы, целая часть которых лежит внутри отрезка `[a, b]`, а потом – все остальные. Упорядочить каждую часть массива по возрастанию, используя алгоритм сортировки методом прямого обмена. Значение `a` и `b` вводит пользователь.

Вариант 13

В одномерном массиве, состоящем из *N* вещественных элементов, вычислить:

1) количество элементов массива, лежащих в диапазоне от `A` до `B`;

2) сумму элементов массива, расположенных после максимального элемента.

Упорядочить элементы массива по убыванию модулей элементов методом вставки. Значение `A` и `B` вводит пользователь.

Вариант 14

В одномерном массиве, состоящем из *N* вещественных элементов, вычислить:

1) количество нулевых элементов массива;

2) сумму элементов массива, расположенных после минимального элемента.

Упорядочить элементы массива по возрастанию модулей элементов методом прямого обмена.

Вариант 15

В одномерном массиве, состоящем из *N* вещественных элементов, вычислить:

1) количество элементов массива, больших **c** (значение **c** вводит пользователь);

2) произведение элементов массива, расположенных после максимального по модулю элемента.

Преобразовать массив таким образом, чтобы сначала располагались все неотрицательные элементы, а потом – все отрицательные. Упорядочить каждую часть массива по возрастанию, используя алгоритм сортировки методом вставки.

Вариант 16

В одномерном массиве, состоящем из *N* вещественных элементов, вычислить:

1) количество отрицательных элементов массива;

2) сумму модулей элементов массива, расположенных после минимального по модулю элемента.

Заменить все отрицательные элементы массива их квадратами и упорядочить элементы массива по возрастанию, используя алгоритм сортировки методом прямого обмена.

Вариант 17

В одномерном массиве, состоящем из *N* вещественных элементов, вычислить:

1) количество положительных элементов массива;

2) сумму элементов массива, расположенных после последнего элемента, равного нулю.

Преобразовать массив таким образом, чтобы сначала располагались все элементы, целая часть которых не превышает 1, а потом – все остальные. Упорядочить каждую часть массива по возрастанию, используя алгоритм сортировки методом прямого выбора.

Вариант 18

В одномерном массиве, состоящем из *N* целых элементов, вычислить:

1) количество элементов массива, меньших **C** (значение **c** вводит пользователь);

2) сумму положительных элементов массива, расположенных после последнего отрицательного элемента.

Преобразовать массив таким образом, чтобы сначала располагались все четные элементы, а потом – все нечетные. Упорядочить каждую часть массива по возрастанию, используя алгоритм сортировки методом вставки.

Вариант 19

В одномерном массиве, состоящем из *N* вещественных элементов, вычислить:

1) произведение отрицательных элементов массива;

2) сумму положительных элементов массива, расположенных до максимального элемента. Изменить порядок следования элементов в массиве на обратный.

Вариант 20

В одномерном массиве, состоящем из *N* вещественных элементов, вычислить:

1) произведение положительных элементов массива;

2) сумму элементов массива, расположенных до минимального элемента.

Упорядочить по возрастанию отдельно элементы, стоящие на четных местах, и элементы, стоящие на нечетных местах, методом вставки.

Вариант 21

В одномерном массиве, состоящем из *N* вещественных элементов, вычислить:

1) сумму элементов массива, значения которых превышают 3;

2) произведение элементов массива, расположенных до максимального и после минимального элементов.

Упорядочить элементы массива по возрастанию, используя алгоритм сортировки методом прямого обмена.

Вариант 22

В одномерном массиве, состоящем из *N* вещественных элементов, вычислить:

1) сумму элементов массива, значения которых превышают 1;

2) произведение элементов массива, расположенных после максимального по модулю и до минимального по модулю элемента.

Упорядочить элементы массива по убыванию, используя алгоритм сортировки методом вставки.

Вариант 23

В одномерном массиве, состоящем из *N* вещественных элементов, вычислить:

1) произведение элементов массива с четными номерами, значения которых превышают 1;

2) сумму элементов массива, расположенных до первого и после последнего нулевого элементов.

Преобразовать массив таким образом, чтобы сначала располагались все положительные элементы, а потом все отрицательные. Выполнить сортировку каждой части массива методом прямого выбора по возрастанию.

Вариант 24

В одномерном массиве, состоящем из *N* вещественных элементов, вычислить:

1) сумму элементов массива с нечетными номерами, значения которых превышают 0;

2) сумму элементов массива, расположенных до первого и после последнего отрицательного элемента.

Удалить все элементы, модуль которых не превышает 1, и упорядочить элементы массива по возрастанию, используя алгоритм сортировки методом прямого обмена.

Вариант 25

В одномерном массиве, состоящем из *N* вещественных элементов, вычислить:

1) максимальный элемент массива;

2) сумму элементов массива, расположенных после последнего положительного элемента.

Заменить все элементы, модуль которых находится внутри отрезка [a, b] на число b и упорядочить элементы массива по возрастанию, используя алгоритм сортировки методом прямого обмена. Значение `a` и `b` вводит пользователь.

Вариант 26

В одномерном массиве, состоящем из *N* вещественных элементов, вычислить:

1) минимальный элемент массива;

2) сумму элементов массива, расположенных до первого и после последнего положительного элемента.

Преобразовать массив таким образом, чтобы сначала располагались все эле-

менты, больше 1, а потом – все остальные. Упорядочить каждую часть массива по возрастанию, используя алгоритм сортировки методом прямого обмена.

Вариант 27

В одномерном массиве, состоящем из *N* целых элементов, вычислить:

1) номер максимального элемента массива;

2) произведение элементов массива, расположенных после первого нулевого элемента.

Преобразовать массив таким образом, чтобы в первой его половине располагались элементы, стоявшие на нечетных позициях, а во второй половине – элементы, стоявшие на четных позициях. Упорядочить каждую часть массива по возрастанию, используя алгоритм сортировки методом прямого обмена.

Вариант 28

В одномерном массиве, состоящем из *N* вещественных элементов, вычислить:

1) номер минимального элемента массива;

2) сумму элементов массива, расположенных после первого отрицательного элемента.

Преобразовать массив таким образом, чтобы сначала располагались все элементы, модуль которых не превышает 1, а потом – все остальные. Упорядочить каждую часть массива по возрастанию, используя алгоритм сортировки методом вставки.

Вариант 29

В одномерном массиве, состоящем из *N* вещественных элементов, вычислить:

1) максимальный по модулю элемент массива;

2) сумму элементов массива, расположенных после второго положительного элемента.

Преобразовать массив таким образом, чтобы элементы, больше 1, располагались после всех остальных. Упорядочить каждую часть массива по убыванию, используя алгоритм сортировки методом прямого выбора.

Вариант 30

В одномерном массиве, состоящем из *N* целых элементов, вычислить:

1) минимальный по модулю элемент массива;

2) сумму модулей элементов массива, расположенных после второго элемента, равного нулю.

Преобразовать массив таким образом, чтобы в первой его половине располагались элементы, стоявшие на нечетных позициях, а во второй половине – элементы, стоявшие на четных позициях. Упорядочить каждую часть массива по возрастанию, используя алгоритм сортировки методом прямого обмена.

4.4 Порядок выполнения работы

4.4.1 Разработать структурную схему алгоритма решения задачи.

4.4.2 Написать соответствующую программу.

4.4.3 Составить тестовые примеры, следуя указаниям, изложенным в разделе 4.2.

4.4.4 Выполнить отладку программы для всех тестовых примеров.

4.5 Содержание отчета

Цель работы, вариант задания, структурная схема алгоритма, текст программы, тестовые примеры, выводы.

4.6 Контрольные вопросы

4.6.1 Как в языках C/C++ описываются одномерные массивы?
4.6.2 Каковы особенности инициализации массива при его объявлении?
4.6.3 Как осуществляется обращение к отдельным элементам массива?
4.6.4 Напишите фрагмент программы, выполняющий поиск заданного элемента в массиве.
4.6.5 Напишите фрагмент программы, выполняющий поиск минимального элемента в массиве.
4.6.6 Объясните алгоритм сортировки методом пузырька.
4.6.7 Объясните алгоритм сортировки методом вставки.
4.6.8 Объясните алгоритм сортировки методом прямого выбора.
4.6.9 Напишите программу сортировки методом пузырька.
4.6.10 Напишите программу сортировки методом вставки.
4.6.11Напишите программу сортировки методом прямого выбора.

ЛАБОРАТОРНАЯ РАБОТА №5
«ОБРАБОТКА ОДНОМЕРНЫХ ДИНАМИЧЕСКИХ МАССИВОВ С ПОМОЩЬЮ ФУНКЦИЙ»

5.1 Цель работы

Изучение особенностей представления и обработки одномерных массивов в языках С/С++ с учетом связи указателей и массивов. Получение практических навыков реализации алгоритмов обработки одномерных динамических массивов средствами языков С/С++. Исследование особенностей обработки одномерных динамических массивов.

5.2 Краткие теоретические сведения

5.2.1 Указатели

Указатели – это переменные, которые содержат в качестве своих значений *адреса памяти*. В общем случае переменная типа *указатель* объявляется следующим образом:

```
тип *переменная_указатель;
```

Такая запись означает, что `переменная_указатель` является указателем на объект типа `тип`. Так, объявление

```
int *countPtr,count=5;
```

объявляет переменную `countPtr` типа `int *` (т. е. указатель на целое число) и читается как «`countPtr` является указателем на целое число» или «`countPtr` указывает на объект типа `int`». Однако переменная `count` объявлена как целое число, но не как указатель на целое число. Смвол `*` в объявлении относится только к `countPtr`. *Каждая переменная, объявляемая как указатель, должна иметь перед собой знак звездочки* (`*`).

Указатели должны инициализироваться либо при своем объявлении, либо с помощью оператора присваивания. Указатель может получить в качестве начального значения `0`, `NULL` или адрес. Указатель с начальными значениями `0` или `NULL` ни на что не указывает и часто используется как *ограничитель в динамических структурах*. `NULL` – это *символическая константа*, определенная в головном файле `<iostream.h>`, а также в нескольких головных файлах стандартной библиотеки языка С.

Унарный оператор адресации `&` возвращает адрес своего операнда. Например, в результате выполнения инструкции

```
countPtr=&count;
```

адрес переменной `count` будет запомнен в указателе `countPtr`.

Оператор *****, называемый *оператором раскрытия ссылки* или *оператором разыменования (разадресации)*, возвращает значение объекта, на который указывает указатель. Например, инструкция

```
cout<<*countPtr<<endl;
```

выводит на экран значение переменной `count`, а именно `5`.

Два оператора языка C++ `new` и `delete` позволяют управлять временем жизни какой-либо переменной. Переменная может быть создана в любой точке программы с помощью оператора `new` и затем при необходимости уничтожена с помощью оператора `delete`.

В результате выполнения инструкции

```
countPtr=new int;
```

переменной-указателю `countPtr` присваивается адрес начала участка памяти размером `sizeof(int)` байт. Память выделяется *динамически* из специальной области, которая называется *куча (heap)*. Оператор `delete` освобождает ранее занятую память, возвращая ее в кучу.

Наряду с операторами `new` и `delete` в языках C/C++ существуют две функции для захвата/освобождения памяти в куче: `malloc` и `free` соответственно. Эти функции описаны в головном файле `<stdlib.h>`, а также в `<alloc.h>`. Функция `malloc(size)` запрашивает память размером `size` байт из кучи и возвращает указатель на первый байт этого участка. Функция `malloc` всегда возвращает указатель на тип `void`, поэтому для получения адреса объекта определенного типа необходимо выполнить соответствующее *преобразование типа*, например:

```
int *x;
x=(int*)malloc(sizeof(int));
```

Здесь преобразование типа `(int*)` приводит значение, возвращаемое функцией `malloc` к адресу указателя на целочисленную переменную.

Память, выделенную в куче с помощью функции `malloc`, следует освобождать с помощью функции `free`. Единственным аргументом функции `free` является указатель, ссылающийся на освобождаемую память.

При работе с указателями возможны ошибки. Рассмотрим фрагмент программы

```
int *p=new int, *q=new int;
*p=255, *q=127;
p=q, *p=63;
```

Присвоение `p=q` означает, что `p`, начиная с этого момента, указывает на ту же область памяти, что и `q`. Когда по адресу, содержащемуся в `p`, заносится значение `63` (для этого используется инструкция `*p=63;`), значение, на которое ссылается `q`, также будет равно `63` (`*q==63`). Кроме того, после присвоения `p=q` значение `*p==255` оказывается *потерянным*. Не существует доступа к этой области памяти. Она остается захваченной до конца выполнения программы. Такие явления называют *утечкой памяти*.

Если указатель является *локальной переменной*, т. е. описан в некотором блоке программы, как это имеет место во фрагменте

```
{     char *q1=new char;
     *q1='c';
}
```

то при выходе из блока он *перестанет существовать* и память, на которую он ссылается, окажется недоступной. Эта память не помечается как свободная и поэтому не может быть использована в дальнейшем.

Еще один источник ошибок – *неосвобождение памяти*, запрошенной ранее с помощью оператора `new` или функции `malloc`. Система не способна автоматически освобождать память в куче. Неосвобождение памяти может остаться незамеченным в небольших программах, однако часто приводит к отказам в серьезных разработках и является плохим стилем программирования.

5.2.2 Массивы и указатели

Имя массива является *константой-указателем* и содержит адрес области памяти, которую занимает набор последовательных ячеек, соответствующих массиву.

Декларация

```
double a[10];
```

определяет массив `a`, состоящий из десяти последовательных объектов с именами `a[0]`, `a[1]`, …, `a[9]`. Если `pa` есть указатель на `double`, т. е. определен как `double *pa`, то в результате присваивания

```
pa=a;
```

или

```
pa=&a[0];
```

`pa` будет указывать на нулевой элемент массива `a`; иначе говоря, `pa` будет содержать адрес элемента `a[0]`.

Если `pa` указывает на некоторый элемент массива, то `pa+1` *по определению* указывает на следующий элемент, `pa-1` указывает на предыдущий элемент, `pa+i` – на *i*-й элемент после `pa`, а `pa-i` – на *i*-й элемент перед `pa`. Таким образом, если `pa` указывает на `a[0]` (`pa==&a[0]`), то `*pa` есть содержимое `a[0]`, `*(pa+1)` есть содержимое `a[1]`, а `*(pa+i)` – содержимое `a[i]`. Сделанные замечания верны безотносительно к типу и размеру элементов массива `a`. Однако нельзя выходить за границы массива и тем самым ссылаться на несуществующие объекты.

Если `p` и `q` указывают на элементы *одного и того же* массива, то к ним можно применять операторы отношения `==`, `!=`, `<`, `<=`, `>`, `>=`. Например, отношение вида `p<q` истинно, если `p` указывает на «более ранний» элемент массива, чем `q`. Любой указатель всегда можно сравнить на равенство и неравенство с нулем.

Допускается также вычитание указателей. Например, если `p` и `q` ссылаются на элементы *одного и того же* массива и `p<q`, то `q-p+1` есть число элементов от `p` до `q` включительно.

Если до начала работы программы количество элементов в массиве неизвестно, то следует использовать *динамические массивы*. Память под них выделяется во *время выполнения* программы с помощью оператора `new` или функции `malloc`, а освобождается с помощью оператора `delete` или функции `free` соответственно, например:

```
#include <stdlib.h>
main
{
    int n;
    cout<<"Введите количество элементов: ", cin>>n;
    int *a=new int[n];
    double *b=(double*)malloc(n*sizeof(double));
    // обработка массивов a и b
    ..............................
    delete []a;
    free(b);
    return 0;
}
```

5.2.3 Пример программы обработки динамических массивов

Рассмотрим пример работы с динамическими массивами. Ниже представлен текст программы, которая в целочисленном массиве, не все элементы которого одинаковы, заменяет набор элементов, расположенных между максимальным и минимальным элементами массива (максимальный и минимальный элементы не включаются в набор), на единственный элемент, равный сумме положительных элементов в заменяемом наборе. После этого выполняется сортировка полученного массива методом прямого обмена.

```
#include <conio.h>
#include <iostream.h>
main()
// пример программы обработки динамических массивов
{int n, // кол-во эл-тов в исходном массиве
      m; // кол-во эл-тов в результирующем массиве
 int *a, // указатель на массив (исходный
        // и результирующий)
     *temp_a; // указатель на временный
              // (промежуточный) массив
 int i,j; // счетчики циклов
 int imax, // индекс макс. эл-та массива
     imin; // индекс мин. эл-та массива
 int ibeg, // индекс начала заменяемого набора эл-тов
     iend; // индекс конца заменяемого набора эл-тов
int s_pos; // сумма полож. эл-тов в заменяемом наборе
int temp; // буфер для сортировки методом
          //прямого обмена
    clrscr();
    //--------------------------------------

    // формирование исходного массива
```

```
  cout<<"Введите количество элементов массива: ", cin>>n;
     a=new int[n];
     cout<<"Введите "<<n<<" элемента(ов) массива: ";
     for(i=0;i<n;i++) cin>>a[i];
      cout<<"Исходный массив:"<<endl;
      for(i=0;i<n;i++) cout<<"a["<<i<<"]="<<a[i]<<" ";
      cout<<endl;
    // поиск индексов макс. и мин. эл-тов массива
     for(imax=imin=0,i=1;i<n;i++)
        {
            if(a[i]>a[imax]) imax=i;
            if(a[i]<a[imin]) imin=i;
        }
    cout<<"Максимальный элемент:a[" << imax << "]=" <<
    a[imax] << endl  <<"Минимальный элемент: a[" << imin <<
    "]=" << a[imin] << endl;
// поиск индексов начала и конца заменяемого набора эл-тов
    if(imax<imin) ibeg=imax+1, iend=imin-1;
    else ibeg=imin+1, iend=imax-1;
    cout<<"Заменяемый набор элементов:"<<endl;
     for(i=ibeg;i<=iend;i++)
        cout<<"a["<<i<<"]="<<a[i]<<" ";
        cout<<endl;
    // расчет суммы полож. эл-тов заменяемого набора
        for(i=ibeg,s_pos=0;i<=iend;i++)
            if(a[i]>0) s_pos+=a[i];
            cout<<"Сумма положительных элементов заменяемого
набора: "<<s_pos<<endl;
        temp_a=a; // адрес исходного массива
        m=n+ibeg-iend; // кол-во эл-тов
                       //в результирующем массиве
        a=new int[m]; // результирующий массив
    /*---- формирование результирующего массива -------*/
    // запись эл-тов, расположенных до начала
    // заменяемого набора
        for(i=0,j=0;i<ibeg;i++,j++) a[j]=temp_a[i];
        // запись суммы полож. эл-тов заменяемого набора
        a[j++]=s_pos;
        // запись эл-тов, расположенных после
        //конца заменяемого набора
        for(i=iend+1;i<n;i++,j++) a[j]=temp_a[i];
        // освобождение памяти из-под исходного массива
        delete []temp_a;

        // вывод на экран результирующего массива
```

```
    cout<<"Результирующий массив:"<<endl;
    for(i=0;i<m;i++) cout<<"a["<<i<<"]="<<a[i]<<" ";
    cout<<endl;
    // сортировка массива методом прямого обмена
    for(i=m-1;i;i--)
        for(j=0;j<i;j++)
            if(a[j]>a[j+1])
                temp=a[j],a[j]=a[j+1],a[j+1]=temp;
    // вывод на экран отсортированного массива
    cout<<"Отсортированный массив:"<<endl;
    for(i=0;i<m;i++) cout<<"a["<<i<<"]="<<a[i]<<" ";
    cout<<endl;
    cout<<"Нажмите любую клавишу...";
    getch();
    delete []a; // освобождение памяти
    return 0;
}
```

Исходный целочисленный массив формируется динамически, т. е. во время выполнения программы; при этом используется значение размера массива, введенное пользователем. После того, как введены элементы массива, происходит поиск индексов максимального и минимального элементов `imax` и `imin` соответственно.

Так как порядок расположения элементов в массиве заранее не известен, то сначала может следовать как максимальный (`imax<imin`), так и минимальный элемент (`imin<imax`). С учетом этого обстоятельства осуществляется поиск индексов начала и конца заменяемого набора `ibeg` и `iend` соответственно. Далее производится расчет суммы положительных элементов заменяемого набора, т. е. тех элементов исходного массива, индексы которых лежат в диапазоне от `ibeg` до `iend` включительно.

Затем динамически происходит создание результирующего массива. При этом адрес исходного массива запоминается, чтобы после того, как формирование результирующего массива будет окончено, освободить память, которую занимает исходный массив. Для вычисления размера результирующего массива необходимо из размера исходного массива (`n`) вычесть количество элементов в заменяемом наборе (`iend-ibeg+1`) и добавить единицу, соответствующую элементу, заменяющему набор:

```
n-(iend-ibeg+1)+1==n+ibeg-iend
```

Далее происходит формирование результирующего массива. Следует обратить внимание, что при копировании элементов из исходного массива в результирующий массив используются *разные* счетчики цикла, так как копируемым элементам соответствуют *различные* индексы в соответствующих массивах. После того как результирующий массив сформирован, память из-под исходного массива освобождается. В конце результирующий массив сортируется методом прямого обмена.

5.2.4 Функции

Функция – это группа операторов, вызываемая по имени и возвращающая в точку вызова предписанное значение. Формат простейшего заголовка функции: `<тип> <имя>(<список формальных параметров>)`

Например, заголовок функции `main` обычно имеет вид

```
int main();
```

Это означает, что никаких параметров функции не передается, а возвращает она одно значение типа `int`. Функция может и не возвращать значения, в этом случае должен быть указан тип `void`.

Имена формальных параметров при задании *прототипа функции* можно не указывать. Достаточно указать их тип: `double sin(double);`

Здесь записано, что функция имеет имя `sin`, вычисляет значение синуса типа `double` и для этого ей надо передать аргумент типа `double`.

Хороший стиль программирования требует, чтобы все, что передается в функцию и обратно, отражалось в ее заголовке.

Заголовок задает *объявление функции*. *Определение функции*, кроме заголовка, включает ее тело, т.е. операторы, которые выполняются при вызове функции, например:

```
int sum(int a,int b)
{    return a+b; // тело функции
}
```

Тело функции представляет собой блок, заключенный в фигурные скобки. Для возврата результата, вычисленного в функции, служит оператор `return`. После него указывается выражение, значение которого и передается в точку вызова функции. Функция может иметь несколько операторов возврата.

Для вызова функции указывают ее имя, а также передают ей значения *фактических параметров*:

```
<имя функции>(<список фактических параметров>);
```

Порядок следования аргументов в списке фактических параметров и их типы должны строго соответствовать *списку формальных параметров*. Пример обращения к определенной выше функции `sum`:

```
int summa,a=5;
summa=sum(a,5);
```

Рассмотрим *механизм передачи параметров* в функцию. Он весьма прост. Параметры передаются в функцию как *параметры-значения*. Иными словами, функция работает с копией значений, которые ей передаются. Кстати, именно поэтому на месте таких параметров можно при вызове задавать выражения, например: `summa=sum(a*10+5,a+3);`

Для передачи в функцию параметров способом *«параметр-переменная»* в языке С используют указатели. Определим функцию `swap`, осуществляющую обмен значениями двух переменных.

```
void swap(int *x,int *y)
{   int temp;
    temp=*x;
    *x=*y;
    *y=temp;
}
```

Здесь параметры x и y являются указателями и позволяют функции осуществлять доступ к ячейкам памяти вызвавшей ее программы и менять их значения местами. Чтобы функция swap выполняла желаемые действия, необходимо при вызове на место параметров x и y подставить адреса соответствующих ячеек памяти. Для этого используется операция взятия адреса &:

```
swap(&a,&b);
```

в этом случае функция swap поменяет местами значения переменных a и b.

В языке C++ для передачи параметров способом «параметр-переменная» можно использовать *ссылочные переменные*. *Ссылка* – это переменная, которая ассоциирована с другой переменной и содержит ее адрес:

```
int ivar=1234;
int &iref=ivar; // ссылка iref
```

Здесь iref – это ссылка, которой присваивается адрес переменной ivar. Ссылки должны инициализироваться при их объявлении.

Не существует операторов, непосредственно производящих действия над ссылками. Все операции выполняются над ассоциированными значениями. Так, оператор iref++ выполняет инкремент значения ivar.

Сами по себе ссылки применяются редко. Их обычно используют в качестве параметров функций. Так, функцию swap можно переписать в виде

```
void swap(int &x,int &y)
{   int temp;
    temp=x;
    x=y;
    y=temp;
}
```

При этом упрощается вызов функции: swap(a,b);

В этом случае x будет ссылаться на a, а y – на b, и функция swap выполнит обмен значениями a и b.

При определении функции входные данные обычно передаются по значению, а результаты ее работы – по ссылке или указателю.

Следует иметь ввиду, что возвращение ссылки или указателя из функции может привести к проблемам, если переменная, на которую делается ссылка, вышла из области видимости. Например,

```
int & func()
{  int x;
    x=5;
    return x;}
```

В этом случае попытка возвратить ссылку на локальную переменную приведет к ошибке.

Эффективность передачи в функцию адреса вместо самой переменной ощутима и в скорости работы, особенно если используются массивы.

Если в функцию требуется передать достаточно большой объект, однако его модификация не предусматривается, то используется *константный указатель* или ссылка:

```
const int * func(const int * number);
const int & func(const int & number);
```

Здесь функция `func` принимает и возвращает указатель или ссылку на константный объект типа `int`. Любая попытка модифицировать такой объект внутри функции вызовет сообщение компилятора об ошибке.

В заключение рассмотрим передачу массивов функциям. Пусть требуется написать функцию, суммирующую элементы массива. Предположим, что имеются следующие описания:

```
#define MAX 100;
int a[MAX];
```

Тогда можно объявить функцию со следующим прототипом:

```
int SumOfA(int a[MAX]);
```

Однако будет лучше оставить квадратные скобки пустыми и добавить второй аргумент, определяющий размер массива:

```
int SumOfA(int a[],int n);
```

Необходимо помнить, что в этом случае реально в функцию передается указатель на тип элемента массива. Таким образом, изменение любого элемента массива внутри функции повлияет и на оригинал. Поэтому лучше сразу передать в функцию указатель на нулевой элемент константного массива, например:

```
int SumOfA(const int *a,int n);
```

Аналогичным образом поступают и при передаче двумерных массивов в функцию. При передаче двумерного массива в функцию в списке формальных параметров обычно указывают только количество столбцов массива. Количество строк передают в виде дополнительного параметра:

```
int SumOfMatrix(int matrix[][10],int nrow);
```

Поскольку доступ к двумерному массиву можно получить с помощью указателя на указатель, то прототип функции можно объявить и так:

```
int SumOfMatrix(int **matrix,int nrow,int ncol);
```

Здесь `nrow` – количество строк матрицы, а `ncol` – количество столбцов.

5.2.5 Обработка массивов с помощью функций

Пусть требуется написать программу, которая создает одномерный динамический массив, считывает значения элементов массива с клавиатуры, находит сумму элементов массива.

Ниже приведен текст программы.

```
#include <stdio.h>
#include <stdlib.h>
// ------------------  прототипы функций
int summa(int *a, int n);// функция, суммирующая элементы
массива
void out_mas(int *a, int n);// функция, выполняющая вывод
int* in_mas(int *a, int n);// функция, выполняющая ввод
// -----------------   основная функция
int main(){
  int n, i;
 printf("Введите количество элементов массива:n->");
 scanf("%d",&n);
    int *a=in_mas(a,n);
        out_mas(a,n);
     int res=summa(a,n);
     printf("\n сумма равна %d",res);
    free []a;// освобождение памяти
return 0;
}
  //----------------------- подсчет суммы
int summa(int *a, int n){
 int s=0;
  for(int i=0;i<n;i++)
      s+=a[i];
    return s;
}
  //-------------------- выделение и заполнение массива
int* in_mas(int *x, int n){
  x=(int *) malloc(sizeof(int)*n); // выделение памяти
 printf("Введите элементы массива:");
     for(int i=0;i<n;i++) // ввод
        scanf("%d",&x[i]);
    return x;
}
 //----------------------- печать элементов массива
void out_mas(int *a, int n) {
 int i;
 printf("\nмассив\n");
 for(i=0;i<n;i++)
   printf(" %d",a[i]);
 return;
}
```

5.3 Варианты заданий

Даны натуральные числа n, p, q,причем $1<p<q<n$. Также дана последовательность чисел $a_1, a_2, \ldots, a_n$.

Требуется ввести с клавиатуры исходные данные, выполнить их обработку в соответствии с вариантом задания и вывести результаты обработки на экран.

Программу оформить в виде функций, выполняющих:

– выделение памяти и заполнения массива (с клавиатуры);

– обработку массива способом согласно варианту;

– сортировку массива способом согласно варианту (сортировать массив от p-го до q-го элемента, параметры p и q вводит пользователь с клавиатуры);

– вывод массива на экран.

Вызов функций осуществлять из интерактивного меню, при выходе из программы не забыть освободить память.

Вариант 1

Последовательность состоит из целых чисел. Необходимо найти количество четных элементов последовательности и определить значение наименьшего из них.

Упорядочить элементы последовательности a_p, a_{p+1}, …, a_q по убыванию, используя алгоритм сортировки методом вставки.

Вариант 2

Последовательность состоит из целых чисел. Необходимо найти сумму положительных четных элементов последовательности.

Упорядочить элементы последовательности a_p, a_{p+1}, …, a_q по возрастанию, используя алгоритм сортировки методом прямого обмена.

Вариант 3

Последовательность состоит из целых чисел. Необходимо найти количество отрицательных элементов последовательности, кратных 3 и 7.

Упорядочить элементы последовательности a_p, a_{p+1}, …, a_q по убыванию, используя алгоритм сортировки методом вставки.

Вариант 4

Последовательность состоит из целых чисел. Необходимо найти количество элементов последовательности таких, что верно $sin(a_i) < 0$, $i=1,2…n$.

Упорядочить элементы последовательности a_p, a_{p+1}, …, a_q по возрастанию, используя алгоритм сортировки методом вставки.

Вариант 5

Последовательность состоит из целых чисел. Необходимо найти количество элементов последовательности, отвечающих условию: $a_{i-1}<a_i>a_{i+1}$, при $i=2,3…n-1$.

Упорядочить элементы последовательности a_p, a_{p+1}, …, a_q по убыванию, используя алгоритм сортировки методом прямого выбора.

Вариант 6

Последовательность состоит из целых чисел. Необходимо найти количество нечетных элементов последовательности и определить значение наибольшего из них.

Упорядочить элементы последовательности a_p, a_{p+1}, …, a_q по возрастанию, используя алгоритм сортировки методом вставки.

Вариант 7

Последовательность состоит из вещественных чисел. Необходимо ввести два числа bи c. Все элементы последовательности, значения которых попадают в интервал $[b, c]$, заменить на 0.

Упорядочить элементы последовательности a_p, a_{p+1}, ..., a_q по убыванию, используя алгоритм сортировки методом прямого выбора.

Вариант 8

Последовательность состоит из вещественных чисел. Необходимо ввести число *b*. Определить элемент последовательности, наиболее близкий к числу *b*.

Упорядочить элементы последовательности a_p, a_{p+1}, ..., a_q по возрастанию, используя алгоритм сортировки методом прямого обмена.

Вариант 9

Последовательность состоит из вещественных чисел. Необходимо поменять местами элементы с четными и нечетными индексами (для n кратного двум).

Упорядочить элементы последовательности a_p, a_{p+1}, ..., a_q по убыванию, используя алгоритм сортировки методом вставки.

Вариант 10

Последовательность состоит из вещественных чисел. Необходимо найти количество элементов последовательности таких, что верно $cos(a_i) > 0$, $i=1,2…n$.

Упорядочить элементы последовательности a_p, a_{p+1}, ..., a_q по возрастанию, используя алгоритм сортировки методом вставки.

Вариант 11

Последовательность состоит из вещественных чисел. Необходимо найти произведение тех элементов последовательности, значения которых лежат на отрезке *[b, c]* (границы отрезка вводит пользователь).

Упорядочить элементы последовательности a_p, a_{p+1}, ..., a_q по убыванию, используя алгоритм сортировки методом прямого выбора.

Вариант 12

Последовательность состоит из вещественных чисел. Необходимо найти такое a_i, что выполнится следующее: $max[cos(a_1), cos(a_2)… cos(a_n)] = cos(a_i)$.

Упорядочить элементы последовательности a_p, a_{p+1}, ..., a_q по возрастанию, используя алгоритм сортировки методом прямого выбора.

Вариант 13

Последовательность состоит из целых чисел. Необходимо найти сумму отрицательных нечетных элементов последовательности.

Упорядочить элементы последовательности a_p, a_{p+1}, ..., a_q по убыванию, используя алгоритм сортировки методом прямого обмена.

Вариант 14

Последовательность состоит из целых чисел. Необходимо «перевернуть» последовательность (записать ее в обратном порядке).

Упорядочить элементы последовательности a_p, a_{p+1}, ..., a_q по возрастанию, используя алгоритм сортировки методом прямого выбора.

Вариант 15

Последовательность состоит из целых чисел. Необходимо найти сумму нечетных элементов последовательности, имеющих четные порядковые номера (индексы).

Упорядочить элементы последовательности a_p, a_{p+1}, ..., a_q по убыванию, используя алгоритм сортировки методом прямого обмена.

Вариант 16

Последовательность состоит из целых чисел. Необходимо найти количество элементов последовательности таких, что верно условие $a_i=(a_{i-1}+a_{i+1})/2$, $i=2,3…n-1$.

Упорядочить элементы последовательности a_p, a_{p+1}, …, a_q по возрастанию, используя алгоритм сортировки методом вставки.

Вариант 17

Последовательность состоит из целых чисел. Необходимо определить, есть ли в последовательности три положительных элемента, идущих подряд.

Упорядочить элементы последовательности a_p, a_{p+1}, …, a_q по убыванию, используя алгоритм сортировки методом прямого обмена.

Вариант 18

Последовательность состоит из целых чисел. Необходимо указать, каких элементов в последовательности больше: четных или нечетных, положительных или отрицательных.

Упорядочить элементы последовательности a_p, a_{p+1}, …, a_q по возрастанию, используя алгоритм сортировки методом прямого выбора.

Вариант 19

Последовательность состоит из вещественных чисел. Необходимо найти количество элементов последовательности, отвечающих условию: $a_{i-1}<a_i<a_{i+1}$, при $i=2,3…n-1$.

Упорядочить элементы последовательности a_p, a_{p+1}, …, a_q по убыванию, используя алгоритм сортировки методом прямого обмена.

Вариант 20

Последовательность состоит из вещественных чисел. Необходимо переместить элементы последовательности со значением 0 в ее начало.

Упорядочить элементы последовательности a_p, a_{p+1}, …, a_q по возрастанию, используя алгоритм сортировки методом прямого обмена.

Вариант 21

Последовательность состоит из вещественных чисел. Необходимо найти такое a_i, что выполнится следующее: $min[tg(a_1), tg(a_2)… tg(a_n)]= tg(a_i)$.

Упорядочить элементы последовательности a_p, a_{p+1}, …, a_q по убыванию, используя алгоритм сортировки методом прямого выбора.

Вариант 22

Последовательность состоит из вещественных чисел. Необходимо проверить, является ли последовательность упорядоченной по убыванию.

Упорядочить элементы последовательности a_p, a_{p+1}, …, a_q по возрастанию, используя алгоритм сортировки методом прямого обмена.

Вариант 23

Последовательность состоит из вещественных чисел. Необходимо вычислить значение выражения $\sqrt{a_1^2 + a_2^2 + \cdots + a_n^2}$.

Упорядочить элементы последовательности a_p, a_{p+1}, …, a_q по убыванию, используя алгоритм сортировки методом прямого выбора.

Вариант 24

Последовательность состоит из вещественных чисел. Необходимо найти число с наибольшей дробной частью.

Упорядочить элементы последовательности a_p, a_{p+1}, ..., a_q по возрастанию, используя алгоритм сортировки методом прямого выбора.

Вариант 25

Последовательность состоит из символов. Необходимо найти количество гласных букв в последовательности.

Упорядочить элементы последовательности a_p, a_{p+1}, ..., a_q по убыванию, используя алгоритм сортировки методом прямого выбора.

Вариант 26

Последовательность состоит из символов. Необходимо все заглавные буквы (если они есть) заменить на соответствующие строчные.

Упорядочить элементы последовательности a_p, a_{p+1}, ..., a_q по возрастанию, используя алгоритм сортировки методом вставки.

Вариант 27

Последовательность состоит из символов. Необходимо найти количество символов, не являющихся буквами.

Упорядочить элементы последовательности a_p, a_{p+1}, ..., a_q по убыванию, используя алгоритм сортировки методом вставки.

Вариант 28

Последовательность состоит из символов. Необходимо найти количество групп символов в последовательности (группа – это несколько одинаковых символов, расположенных рядом; например, в последовательности “aabcccaaddd” четыре группы символов).

Упорядочить элементы последовательности a_p, a_{p+1}, ..., a_q по возрастанию, используя алгоритм сортировки методом прямого выбора.

Вариант 29

Последовательность состоит из символов. Необходимо найти самую длинную последовательность цифр в исходном массиве.

Упорядочить элементы последовательности a_p, a_{p+1}, ..., a_q по убыванию, используя алгоритм сортировки методом прямого обмена.

Вариант 30

Последовательность состоит из символов. Необходимо разделить символы последовательности так, чтобы в левой ее части располагались гласные буквы, а в правой – согласные.

Упорядочить элементы последовательности a_p, a_{p+1}, ..., a_q по возрастанию, используя алгоритм сортировки методом прямого выбора.

5.4 Порядок выполнения работы

5.4.1 Проработать необходимый теоретический материал, опираясь на сведения и указания, представленные в предыдущем пункте и литературе [5,7,9,10].

5.4.2 Ответить на контрольные вопросы.

5.4.3 Внимательно изучить постановку задачи.

5.4.4 Разработать структурные схемы алгоритма для всех функций решения задачи.

5.4.5 Разработать *тестовые примеры (не менее двух)*.

5.4.6 Написать программу, решающую поставленную задачу. Программа *обязательно* должна содержать комментарии.

5.4.7 Выполнить *тестирование и отладку* написанной программы, используя разработанные тестовые примеры.

5.4.8 Подготовить отчет по работе.

5.5 Содержание отчета

Цель работы; постановка задачи и вариант задания; структурные схемы функций решения задачи; текст программы; результаты тестирования и отладки программы; выводы.

5.6 Контрольные вопросы

5.6.1 Как объявить переменную типа указатель?

5.6.2 Охарактеризуйте оператор адресации **`&`** и оператор раскрытия ссылки **`*`**.

5.6.3 Охарактеризуйте операторы **`new`** и **`delete`**.

5.6.4 Опишите функции **`malloc`** и **`free`**.

5.6.5 Перечислите наиболее распространенные ошибки, связанные с использованием указателей.

5.6.6 Охарактеризуйте связь между массивами и указателями.

5.6.7 Какие действия можно выполнять над указателями?

5.6.8 Что такое динамические массивы?

5.6.9 Как выделить память под динамический массив?

5.6.10 Приведите формат заголовка функции.

5.6.11 Что является телом функции?

5.6.12 В чем состоит механизм передачи параметров в функцию?

5.6.13 Каким образом функция возвращает значение?

5.6.14 Как записываются списки формальных и фактических параметров?

5.6.15 Как передаются параметры-значения?

5.6.16 Как передаются параметры-переменные?

5.6.17 Как передаются параметры-константы?

5.6.18 Что такое ссылочные переменные? В каких случаях их целесообразно использовать?

5.6.19 Что такое аргументы по умолчанию?

5.6.20 Как осуществляется передача массивов в функции?

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Грошев, А.С. Информатика [Электронный ресурс] : учебник / А.С. Грошев, П.В. Закляков. – Электрон. дан. – Москва : ДМК Пресс, 2019. – 672 с. – Режим доступа: https://e.lanbook.com/book/108131.

2. Вирт, Н. Алгоритмы и структуры данных. Новая версия для Оберона [Электронный ресурс] : учебное пособие / Н. Вирт. – Электрон. дан. – Москва : ДМК Пресс, 2010. – 272 с. – Режим доступа: https://e.lanbook.com/book/1261.

3. Подбельский, В.В. Курс программирования на языке Си [Электронный ресурс] : учебник / В.В. Подбельский, С.С. Фомин. – Электрон. дан. – Москва : ДМК Пресс, 2012. – 384 с. – Режим доступа: https://e.lanbook.com/book/4148.

4. Кудинов, Ю.И. Практикум по основам современной информатики [Электронный ресурс] : учебное пособие / Ю.И. Кудинов, Ф.Ф. Пащенко, А.Ю. Келина. – Электрон. дан. – Санкт-Петербург : Лань, 2011. – 352 с. – Режим доступа: https://e.lanbook.com/book/68471.

5. Солдатенко, И.С. Практическое введение в язык программирования Си [Электронный ресурс] : учебное пособие / И.С. Солдатенко, И.В. Попов. – Электрон. дан. – Санкт-Петербург : Лань, 2019. – 132 с. – Режим доступа: https://e.lanbook.com/book/109619.

Заказ № _____ от «_____» _____2019г. Тираж _____экз.
Изд-во СевГУ