

**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  
2019**

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. – Севастополь: Изд-во СевГУ, 2019. – 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 языке Паскаль с использованием текстовых и типизированных файлов, регулярных и комбинированных типов данных.

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.

Рецензент: Кожяев Е.А., к.т.н., доцент кафедры информационных технологий и компьютерных систем

## СОДЕРЖАНИЕ

Введение.....	4
1 Лабораторная работа №1 “ОСНОВЫ РАБОТЫ В ОПЕРАЦИОННОЙ СИСТЕМЕ WINDOWS”.....	5
2 Лабораторная работа №2 “Программирование линейных и разветвляющихся алгоритмов” .....	13
3 Лабораторная работа №3 “Программирование алгоритмов циклической структуры”.....	19
4 Лабораторная работа №4 “Программирование алгоритмов обработки одномерных массивов”.....	25
5 Лабораторная работа №5 “Обработка двумерных массивов с помощью процедур и функций” .....	36
6 Лабораторная работа №6 “Программирование операций над строками и текстовыми файлами”.....	49
7 Лабораторная работа №7 “Программирование операций над записями и типизированными файлами ”	60
8 Лабораторная работа №8 “Программирование линейных списков”.....	76
9 Лабораторная работа №9 “Программирование нелинейных структур данных”.....	85
10 Лабораторная работа №10 “Программирование с использованием объектов и модулей” .....	94
Библиографический список.....	103

## ВВЕДЕНИЕ

### Цели и задачи лабораторных работ

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

Студенты должны получить практические навыки работы в интегрированной среде программирования, навыки разработки программ, использующих регулярные и комбинированные типы данных, текстовые и типизированные файлы.

### 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 Паскаль 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.

# 1 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 строки, встроенной в Windows, можно двумя способами:

- через быстрое меню (правой кликом 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»
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*.

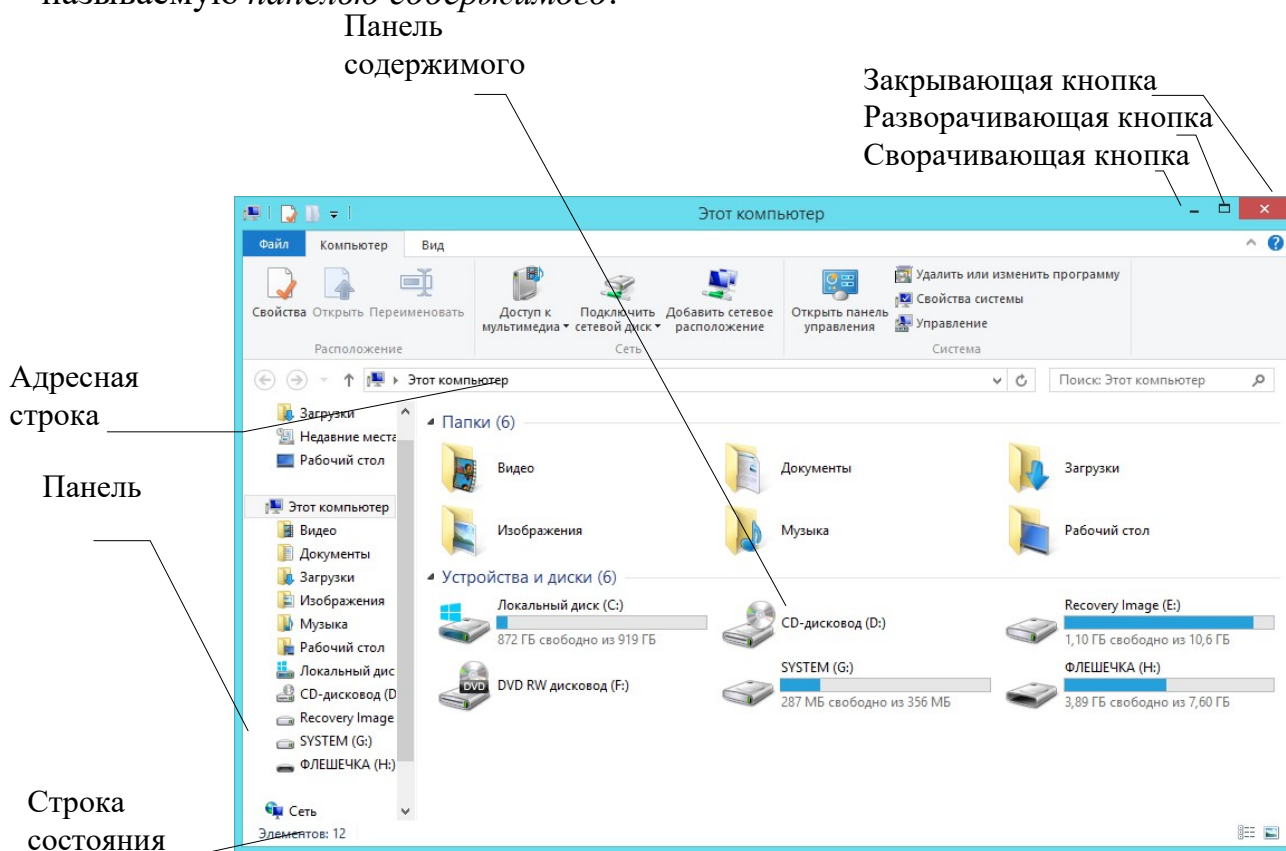


Рисунок 1.2 – Окно программы Проводник

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 того, 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».

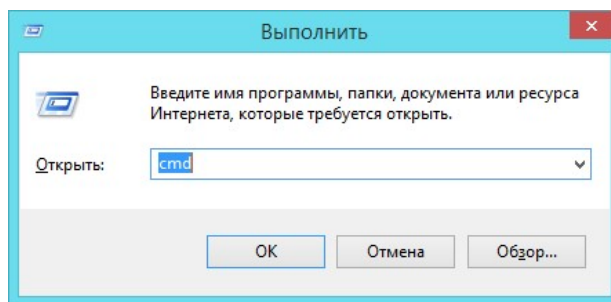


Рисунок 1.3 – Диалог «Выполнить»

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
<b>explorer</b>	открытие проводника Windows
<b>shutdown</b>	завершение работы компьютера
<b>logoff</b>	выход из системы
<b>shutdown -r</b>	перезагрузка
<b>control</b>	открытие панели управления
<b>taskmgr</b>	диспетчер задач
<b>osk</b>	запуск экранной клавиатуры
<b>desk.cpl</b>	окно свойств экрана
<b>controlkeyboard</b>	окно свойств клавиатуры
<b>controlmouse</b>	окно свойств мыши
<b>controlprinters</b>	окно свойств принтеров
<b>appwiz. cpl</b>	быстрый доступ в меню добавления/удаления программ
<b>cleanmgr</b>	запуск инструмента «Очистка диска»
<b>%appdata%</b>	открытие папки с данными установленных приложений
<b>cmd</b>	запуск командной строки
<b>regedit</b>	запуск редактора реестра
<b>msinfo32</b>	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 «С:» 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 «Jp1» под именем «ПерсональныеДанные.txt». Скопировать этот файл во вложенную папку «Jp2» и переименовать его как «Архив.txt».

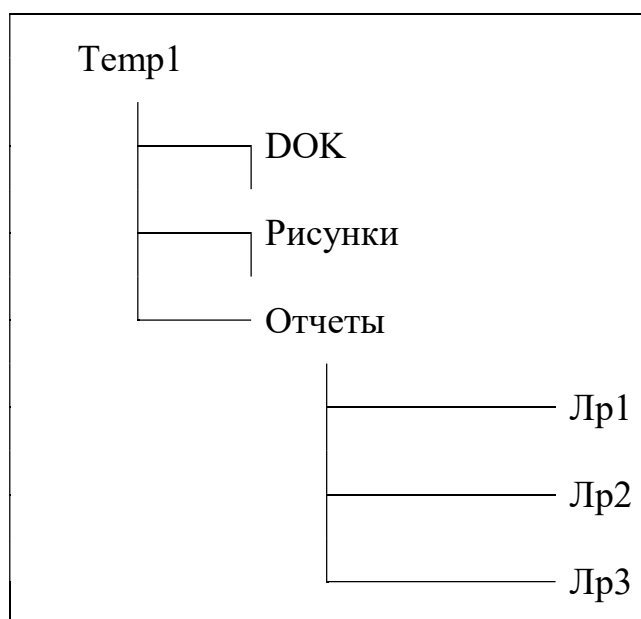


Рисунок 1.4 – Структура папок

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

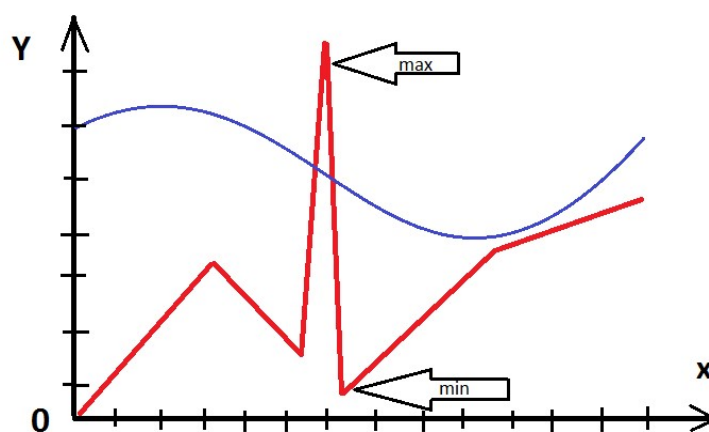


Рисунок 1.5 – Графики

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: формат, 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», 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».

## 2 ЛАБОРАТОРНАЯ РАБОТА №2 «ПРОГРАММИРОВАНИЕ ЛИНЕЙНЫХ И РАЗВЕТВЛЯЮЩИХСЯ АЛГОРИТМОВ»

### 2.1 Цель работы

Изучение основных типов данных, исследование математических функций языка Паскаль и простейших процедур ввода-вывода. Приобретение навыков разработки и отладки программ линейной и разветвляющейся структуры.

### 2.2 Основные сведения и методические указания

Если в программе все операторы выполняются последовательно, один за другим, то программа называется *линейной*.

Рассмотрим пример такой программы. Требуется написать программу для перевода значения температуры, заданной по шкале Цельсия, в значения, соответствующие шкалам Фаренгейта и Кельвина.

После того, как задача поставлена, необходимо составить алгоритм ее решения. Для этого достаточно записать формулы перевода температуры по Цельсию в каждую из упомянутых шкал:

$$T_F = 1,8 T_C + 32, T_K = T_C + 273,15, \quad (2.1)$$

где  $T_C$ ,  $T_F$ ,  $T_K$  – температуры по шкале Цельсия, Фаренгейта и Кельвина соответственно.

Перед написанием любой программы необходимо четко определить входные и выходные данные. В рассматриваемой задаче в качестве исходных данных выступает одно вещественное число, представляющее температуру по Цельсию, а результатом является другое вещественное число.

Алгоритмических сложностей решение этой задачи не представляет, поэтому сразу напишем текст программы:

```
Program Cels_to_other;
Var  Celsius, Fahrenheit, Kelvin: Real;
Begin
    Writeln('Соответствие между температурами');
    Writeln('Цельсия, Фаренгейта и Кельвина');
    Writeln;
    Writeln('Введите значение температуры по Цельсию');
    Readln(Celsius); Writeln;
    Fahrenheit := 1.8 * Celsius + 32;
    Kelvin := Celsius + 273.15;
    Writeln('C = ', Celsius);
    Writeln('F = ', Fahrenheit);
    Writeln('K = ', Kelvin);
    Writeln;
    Writeln('Нажмите <Enter>');
    Readln
End.
```

Здесь после слова **Var** приведено описание переменных программы. При описании любой переменной необходимо указать ее *тип*, чтобы компилятор знал, сколько выделить места в памяти, как интерпретировать значение переменной (то есть ее внутреннее представление), а также какие действия можно будет выполнять с этой переменной. Например, для вещественных чисел в памяти хранится мантисса и порядок, а целые числа представляются просто в двоичной форме, поэтому внутреннее представление одного и того же целого и вещественного числа будет различным. Имена переменным дает программист, исходя из их назначения. Имя не должно начинаться с цифры. Поскольку температура может принимать не только целые значения, но и вещественные, для переменных выбран вещественный тип **Real**.

Для того чтобы пользователь программы знал, в какой момент требуется ввести с клавиатуры данные, применяется так называемое приглашение к вводу:

**Writeln('Введите значение температуры по Цельсию')**

На экран выводится указанная в операторе вызова процедуры **Writeln** строка символов, и курсор переводится на следующую строку. В строке программы, содержащей оператор вызова процедуры **Readln**, выполняется ввод значения переменной **Celsius**.

Далее вычисляются температуры по Фаренгейту и Кельвину, а с помощью процедуры **Writeln** результаты выводятся на экран. При этом сначала выводится обозначение температуры, а затем ее значение. Например, если введено **Celsius = 20**, то

**Writeln('C = ', Celsius)**

выведет на экран **C = 2.0E+01**.

Здесь значение температуры представлено в форме с плавающей точкой. Последние два оператора программы позволяют пользователю задержать на экране окно для просмотра результатов, пока пользователь не нажмет клавишу **Enter**.

В линейной программе все операторы выполняются последовательно. Для того чтобы в зависимости от значений данных обеспечить выполнение разных последовательностей операторов, применяются *операторы ветвления*: условный оператор **if** и оператор выбора **case**.

Условный оператор языка Паскаль записывается в форме:

**If** <логическое выражение> **Then** <оператор1>  
  **Else** <оператор2>

Если <логическое выражение> истинно, то выполняется <оператор1>, иначе выполняется <оператор2>.

Обратите внимание на то, что в условном операторе после служебных слов **Then** и **Else** следует только один оператор. Если по условию задачи требуется в одной из ветвей условного оператора выполнить несколько операторов, то необходимо использовать составной оператор, который начинается словом **Begin** и завершается словом **End**:

**Begin**  
          <оператор 1>;  
          <оператор 2>;  
          ...

<оператор n>

End

Следует обратить внимание на то, что перед словом **Else** точка с запятой не ставится.

Для ситуаций, где имеется несколько альтернатив, можно использовать оператор **Case**. Этот оператор записывается в форме:

```
Case <выражение> of
    <список значений 1>: <оператор 1>;
    <список значений 2>: <оператор 2>;
    ...
    <список значений n>: <оператор n>
End
```

Здесь между зарезервированными словами **case** и **of** находится <выражение>, результат вычисления которого может равняться одному из значений, входящих в <список значений>. Данное выражение называют *селектором*. Выражение-селектор может иметь только значение ординального типа (**integer**, **char**, **boolean**, перечислимый тип, диапазонный тип). Конструкция <список значений> содержит возможные значения выражения-селектора, записанные через запятую.

Например:

```
Var i: Integer;
...
Case i of
    1: x := 1;
    2, 3: Begin
        x := 2;
        y := 3;
        End;
    4, 5: z := 4
End
```

Здесь выражение-селектор представлено переменной *i*. Если *i* равно 1, то выполняется оператор  $x:=1$ , если *i* равно 2 или 3, то выполняется составной оператор, если 4 или 5, то выполняется оператор  $z := 4$ .

Рассмотрим пример разветвляющейся программы. Пусть требуется вычислить функцию:

$$y = \begin{cases} 0, & x < -2 \\ -x-2, & -2 \leq x < -1 \\ x, & -1 \leq x < 1 \\ -x+2, & 1 \leq x < 2 \\ 0, & x \geq 2 \end{cases}.$$

При написании программы следует составить алгоритм ее решения: сначала в общем виде, а затем постепенно детализируя каждый шаг. Такой способ, называемый

«нисходящая разработка», позволяет создавать простые по структуре программы. Ниже приведено описание алгоритма в словесной форме. Запись алгоритма в словесной форме является рекомендуемой, поскольку позволяет облегчить процесс составления программы с использованием языка программирования.

1. Ввести значение аргумента  $x$ .
2. Определить, какому интервалу из области определения оно принадлежит.
3. Вычислить значение  $y$  по соответствующей формуле.
4. Вывести значения.

После этого нетрудно составить текст программы:

```
Program Calculate_function;
Var  x, y: Real;
Begin
    Writeln('Введите значение аргумента');
    Readln(x);
    If x < -2 then y := 0 else
        If (x >= -2) and (x < -1) then y := -x - 2 else
            If (x >= -1) and (x < 1) then y := x else
                If (x >= 1) and (x < 2) then y := -x + 2 else      y := 0;
    Writeln('Для x = ', x, ' значение функции y = ', y);
    Writeln('Нажмите <Enter>'); Readln
End.
```

Тестовые примеры этой программы должны включать, по крайней мере, по одному значению аргумента из каждого интервала. Особое внимание следует обратить на проверку граничных условий.

При подготовке к лабораторной работе следует внимательно изучить стандартные типы данных, элементарные математические функции, особенности стандартного ввода-вывода в языке Паскаль. Следует также изучить приоритет и порядок вычисления выражений, понятие логического выражения, условный оператор и оператор выбора.

### 2.3 Варианты заданий

Составить структурную схему алгоритма и написать на языке Паскаль программу вычисления функции  $z = f(x)$ . Варианты функций приведены ниже. Значения параметров  $a$ ,  $b$  и аргумента  $x$  вводятся с клавиатуры. Результаты вычислений выводятся на дисплей в форме с плавающей точкой.

$$\begin{array}{ll}
 1) \ z = \begin{cases} \sin(x), & \text{если } x \leq a \\ \cos(x), & \text{если } a < x < b \\ \tan(x), & \text{если } x \geq b \end{cases} & 2) \ z = \begin{cases} e^x - \sin(x), & \text{если } x \leq a \\ \cos(x) + |x|, & \text{если } a < x < b \\ x^5, & \text{если } x \geq b \end{cases} \\
 3) \ z = \begin{cases} e^x, & \text{если } x \leq a \\ e^x + \cos(x), & \text{если } a < x < b \\ \tan(x), & \text{если } x \geq b \end{cases} & 4) \ z = \begin{cases} 1.7 \cdot \sin(x), & \text{если } x \leq a \\ \cos(x) + x^2, & \text{если } a < x < b \\ x^5, & \text{если } x \geq b \end{cases}
 \end{array}$$



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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

## 2.4 Порядок выполнения работы

2.4.1 Разработать структурную схему алгоритма решения задачи по заданному варианту.

2.4.2 Написать программу, вычисляющую значение функции.

2.4.3 Разработать тестовые примеры.

2.4.4 Выполнить отладку и провести исследование (тестирование) каждой ветви программы.

2.4.5 Проверить значение функции в граничных точках.

2.4.6 Подготовить отчет по работе, приложив результаты вычислений для всех тестов.

## 2.5 Содержание отчета

Цель работы, постановка задачи, структурная схема алгоритма, текст программы с комментариями, описание тестов и результатов тестирования программы, выводы.

## 2.6 Контрольные вопросы

2.6.1 Охарактеризуйте целый тип данных языка Паскаль.

2.6.2 Охарактеризуйте вещественный тип данных.

2.6.3 Охарактеризуйте литерный тип данных.

2.6.4 Охарактеризуйте логический тип данных.

2.6.5 Приведите примеры операторов, выполняющих ввод и вывод в простейшей форме.

2.6.6 Назовите и охарактеризуйте виды операторов присваивания в языке Паскаль.

2.6.7 Что понимают под условным выражением?

2.6.8 Укажите приоритет операций языка Паскаль и порядок вычисления выражений.

2.6.9 Перечислите действия, реализуемые при выполнении условного оператора.

2.6.10 Какие действия выполняются оператором перехода?

2.6.11 Как организовать разветвление вычислений на три ветви?

2.6.12 Приведите пример использования оператора выбора.

2.6.13 Зачем при отладке программы тестировать все ветви?

### 3 ЛАБОРАТОРНАЯ РАБОТА №3 «ПРОГРАММИРОВАНИЕ АЛГОРИТМОВ ЦИКЛИЧЕСКОЙ СТРУКТУРЫ»

#### 3.1 Цель работы

Получить навыки программирования итерационных циклических алгоритмов, исследовать зависимость объёма вычислений от точности.

#### 3.2 Основные сведения и методические указания

Цикл представляет собой последовательность операторов, которая выполняется многократно. В языке Паскаль имеется три разновидности операторов цикла – *цикл с параметром*, *цикл с предусловием* и *цикл с постусловием*.

Оператор цикла с параметром записывается в одной из следующих форм:

For <переменная>:=<выражение1> To <выражение2> Do <оператор>;

For <переменная>:=<выражение1> Downto <выражение2> Do <оператор>;

Обратите внимание на то, что *переменная* цикла, *выражение 1* и *выражение 2* должны иметь значения ординальных типов. Оператор, записанный после слова Do, выполняется многократно для каждого нового значения переменной цикла. Причём очередное значение переменной вычисляется как следующее или предыдущее по порядку значение (т.е. с помощью функции SUCC() и PRED()).

Цикл с предусловием в языке Pascal записывается в следующем виде:

While <условное выражение> Do <оператор>;

Выход из цикла происходит, когда условное выражение получает значение False.

Оператор цикла с постусловием записывается в виде:

Repeat       <оператор1>;

              <оператор2>;

.....

              <оператор N>

Until <условное выражение>;

В этом случае выход из цикла происходит, когда условное выражение получает значение True. Следует обратить внимание на то, что оператор в цикле While ни разу не выполнится, если условное выражение с самого начала будет иметь значение False. В цикле Repeat-Until операторы выполняются хотя бы один раз независимо от значения условного выражения. Для обеспечения выхода из указанных циклов необходимо, чтобы операторы, находящиеся в теле цикла, воздействовали на условное выражение и через конечное число шагов меняли его значение на противоположное.

Например:

CondExpr:=1.0;

Summa:=0;

While CondExpr > 0.03 do

Begin

Summa:= Summa + CondExpr;

```

    i:=i+1;
    CondExpr:=1/i
end;

```

Здесь начальное значение переменной цикла равно 1,0 и условное выражение `CondExpr>0.03` имеет значение `True`. На каждом шаге выполнения цикла переменная `CondExpr` уменьшается за счёт увеличения значения переменной `i`. Таким образом, через конечное число шагов условное выражение получит значение `False` и цикл завершится. В рассмотренном выше примере вычисляется сумма членов ряда  $1 + 1/2 + 1/3 + \dots + 1/i$  с точностью до члена ряда меньшего, чем 0.03.

При использовании вложенных циклов следует иметь ввиду, что затраты процессорного времени на обработку такой конструкции могут зависеть от порядка следования вложенных циклов. Например, вложенный цикл с параметром

```

For j:=1 to 100000 do
  For k:=1 to 1000 do
    a:=1;

```

выполняется примерно на 10% дольше, чем следующий цикл:

```

For j:=1 to 1000 do
  For k:=1 to 100000 do
    a:=1;

```

Объясняется это тем, что инициализация цикла, т.е. обработка процессором его заголовка с целью определения начального и конечного значения счётчика, требует времени. В первом случае один раз инициализируется внешний цикл и 100000 раз внутренний. Во втором случае таких инициализаций будет  $1 + 1000$ .

Можно сделать вывод, что при программировании вложенных циклов, по возможности, следует делать цикл с наибольшим числом повторений самым внутренним, а цикл с наименьшим числом повторений – самым внешним.

Тема оптимального программирования циклов тесно связана с понятием инварианта цикла.

*Инвариантом цикла* называется фрагмент кода цикла, который никак не зависит от управляющей переменной цикла. Современные компиляторы часто определяют наличие таких фрагментов кода и выполняют их автоматическую оптимизацию. Инвариантные фрагменты цикла следует вычислять вне цикла.

Рассмотрим пример вычисления значения функции  $\operatorname{ch}(x)$  (гиперболический косинус) с помощью бесконечного ряда с точностью  $E$  по формуле:

$$\operatorname{ch}(x) = 1 + \frac{x^2}{2!} + \frac{x^4}{4!} + \dots + \frac{x^{2n}}{2n!} + \dots \quad (3.1)$$

Этот ряд сходится при  $|x| < \infty$ . Для достижения заданной точности требуется суммировать члены ряда, абсолютная величина которых больше  $E$ . Для сходящегося ряда модуль члена ряда  $C_n$  при увеличении  $n$  стремится к нулю. При некотором  $n$  неравенство  $|C_n| \geq E$  перестаёт выполняться и вычисления прекращаются.

Общий алгоритм решения данной задачи очевиден: требуется задать начальное значение суммы ряда, а затем многократно вычислять очередной член ряда и добав-

лять его к ранее найденной сумме. Вычисления заканчиваются, когда абсолютная величина очередного элемента ряда станет меньше заданной точности.

Прямое вычисление ряда по приведённой выше общей формуле, когда  $x$  возводится в степень, вычисляется факториал, а затем числитель делится на знаменатель, имеет два недостатка, которые делают этот способ непригодным. Первый недостаток – большая погрешность вычислений. При возведении в степень и вычислении факториала можно получить большие числа, при делении которых друг на друга происходит потеря точности, поскольку количество значащих цифр, хранимых в ячейке памяти, ограничено. Вторым недостатком связан с эффективностью вычислений: как легко заметить, при вычислении очередного элемента ряда предыдущий уже известен, поэтому вычислять каждый член ряда нерационально.

Для уменьшения количества выполняемых действий следует воспользоваться следующей рекуррентной формулой получения последующего члена ряда через предыдущий:

$$C_{n+1} = C_n T, \quad (3.2)$$

где  $T$  – некоторый множитель,

$$T = \frac{C_{n+1}}{C_n} = \frac{2n!x^{2(n+1)}}{x^{2n}(2(n+1))!} = \frac{x^2}{(2n+1)(2n+2)} \quad (3.3)$$

Ниже приведён текст программы с комментариями.

```

Program GiperCosinus;
Const MaxIter = 500; {Максимальное число итераций}
Var ch, y: real;
    x, eps: real;
    done: boolean;
    n: integer;
Begin
  Writeln('Введите аргумент и точность:');
  Readln(x, eps);
  ch:=1; {первый член ряда}
  y:=ch; {начальное значение аргумента}
  n:=0; done:=true;
  While abs(ch)>eps do
    Begin
      ch:=ch*(x*x/((2*n+1)*(2*n+2))); {очередной член ряда}
      n:=n+1;
      y:=y+ch;
      If n>MaxIter then
        Begin
          Writeln('Ряд расходится!');
          done:=false;
          Break;
        End;
    End;
  End;

```

```

If done then
  Begin
    Writeln('Значение функции ', y, ' для x=', x);
    Writeln('Вычислено после ', n, ' итераций');
  End;
End.

```

Первый член ряда равен 1, поэтому чтобы при первом проходе цикла значение второго члена вычислялось правильно,  $n$  должно быть равно 0. Максимально допустимое количество итераций удобно задать с помощью именованной константы. Для аварийного выхода из цикла применяется оператор **break**, который выполняет выход на первый после цикла оператор.

Поскольку выход как в случае аварийного, так и в случае нормального завершения цикла происходит в одну и ту же точку программы, вводится булева переменная **done**, которая предотвращает печать значения функции после выхода из цикла в случае, когда точность не достигнута. Создание подобных переменных-«флагов», принимающих значение «истина» в случае успешного окончания вычислений и «ложь» в противном случае, является распространённым приёмом программирования.

### 3.3 Варианты заданий

Вычислить и вывести на экран в виде таблицы значения функции, заданной с помощью ряда, на интервале от  $X_{\text{нач}}$  до  $X_{\text{кон}}$  с шагом  $dX$  и точностью  $E$ . Таблицу снабдить заголовком и шапкой. Строка таблицы должна содержать значение аргумента, значение функции и количество просуммированных членов ряда.

$$1. \ln \frac{x+1}{x-1} = 2 \sum_{n=0}^{\infty} \frac{1}{(2n+1)x^{2n+1}} = 2\left(\frac{1}{x} + \frac{1}{3x^3} + \frac{1}{5x^5} + \dots\right), \quad |x| > 1$$

$$2. e^{-x} = \sum_{n=0}^{\infty} \frac{(-1)^n \cdot x^n}{n!} = 1 - x + \frac{x^2}{2!} - \frac{x^3}{3!} + \frac{x^4}{4!} - \dots, \quad |x| < \infty$$

$$3. Si(x) = \int_0^x \frac{\sin(x)}{x} dx = x - \frac{1}{3!} \frac{x^3}{3} + \frac{1}{5!} \frac{x^5}{5} - \dots, \quad |x| < \infty$$

$$4. \ln(x+1) = \sum_{n=0}^{\infty} \frac{(-1)^n \cdot x^{n+1}}{n+1} = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} - \dots, \quad -1 < x \leq 1$$

$$5. \ln \frac{1+x}{1-x} = 2 \sum_{n=0}^{\infty} \frac{x^{2n+1}}{2n+1} = 2\left(x + \frac{x^3}{3} + \frac{x^5}{5} + \dots\right), \quad |x| < 1$$

$$6. \ln(1-x) = -\sum_{n=1}^{\infty} \frac{x^n}{n} = -(x + \frac{x^2}{2} + \frac{x^3}{3} + \dots), \quad -1 \leq x < 1$$

$$7. \operatorname{arcctg}(x) = \frac{\pi}{2} + \sum_{n=0}^{\infty} \frac{(-1)^{n+1} \cdot x^{2n+1}}{2n+1} = \frac{\pi}{2} - x + \frac{x^3}{3} - \frac{x^5}{5} - \dots, \quad |x| < 1$$

$$8. \arctg(x) = \frac{\pi}{2} + \sum_{n=0}^{\infty} \frac{(-1)^{n+1}}{(2n+1) \cdot x^{2n+1}} = \frac{\pi}{2} - \frac{1}{x} + \frac{1}{3x^3} - \frac{1}{5x^5} + \dots, \quad x > 1$$

$$9. \arctg(x) = \sum_{n=0}^{\infty} \frac{(-1)^n \cdot x^{2n+1}}{(2n+1)} = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \dots, \quad |x| < 1$$

$$10. \arth(x) = \sum_{n=0}^{\infty} \frac{x^{2n+1}}{2n+1} = x + \frac{x^3}{3} + \frac{x^5}{5} + \frac{x^7}{7} + \dots, \quad |x| \leq 1$$

$$11. \arcth(x) = \sum_{n=0}^{\infty} \frac{1}{(2n+1) \cdot x^{2n+1}} = \frac{1}{x} + \frac{1}{3x^3} + \frac{1}{5x^5} + \dots, \quad |x| \geq 1$$

$$12. \arctg(x) = -\frac{\pi}{2} + \sum_{n=0}^{\infty} \frac{(-1)^{n+1}}{(2n+1) \cdot x^{2n+1}} = -\frac{\pi}{2} - \frac{1}{x} + \frac{1}{3x^3} - \frac{1}{5x^5} + \dots, \quad |x| \leq 1$$

$$13. e^{-x^2} = \sum_{n=0}^{\infty} \frac{(-1)^n \cdot x^{2n}}{n!} = 1 - x^2 + \frac{x^4}{2!} - \frac{x^6}{3!} + \frac{x^8}{4!} - \dots, \quad |x| < \infty$$

$$14. S = -\frac{(2x)^2}{2} + \frac{(2x)^4}{4!} + \dots + (-1)^n \frac{(2x)^{2n}}{(2n)!} + \dots, \quad |x| < \infty$$

$$15. \frac{\sin(x)}{x} = \sum_{n=0}^{\infty} \frac{(-1)^n \cdot x^{2n}}{(2n+1)!} = 1 - \frac{x^2}{3!} + \frac{x^4}{5!} - \frac{x^6}{7!} - \dots, \quad |x| < \infty$$

$$16. \ln(x) = 2 \sum_{n=0}^{\infty} \frac{(x-1)^{2n+1}}{(2n+1) \cdot (x+1)^{2n+1}} = 2 \left( \frac{x-1}{x+1} + \frac{(x-1)^3}{3 \cdot (x+1)^3} + \frac{(x-1)^5}{5 \cdot (x+1)^5} + \dots \right), \quad x > 0$$

$$17. \ln(x) = \sum_{n=0}^{\infty} \frac{(-1)^n \cdot (x-1)^{n+1}}{n+1} = (x-1) - \frac{(x-1)^2}{2} + \frac{(x-1)^3}{3} - \dots, \quad 0 < x \leq 2$$

$$18. \ln(x) = \sum_{n=1}^{\infty} \frac{(x-1)^n}{nx^n} = \frac{x-1}{x} + \frac{(x-1)^2}{2x^2} + \frac{(x-1)^3}{3x^3} + \dots, \quad x > \frac{1}{2}$$

$$19. \arcsin(x) = x + \sum_{n=1}^{\infty} \frac{1 \cdot 3 \cdot \dots \cdot (2n-1) \cdot x^{2n+1}}{2 \cdot 4 \cdot \dots \cdot 2n \cdot (2n+1)} = x + \frac{x^3}{2 \cdot 3} + \frac{1 \cdot 3 \cdot x^5}{2 \cdot 4 \cdot 5} + \frac{1 \cdot 3 \cdot 5 \cdot x^7}{2 \cdot 4 \cdot 6 \cdot 7} + \frac{1 \cdot 3 \cdot 5 \cdot 7 \cdot x^9}{2 \cdot 4 \cdot 6 \cdot 8 \cdot 9} + \dots, \quad |x| < 1$$

$$20. \arccos(x) = \frac{\pi}{2} - \left( x + \sum_{n=1}^{\infty} \frac{1 \cdot 3 \cdot \dots \cdot (2n-1) \cdot x^{2n+1}}{2 \cdot 4 \cdot \dots \cdot 2n \cdot (2n+1)} \right) = \frac{\pi}{2} - \left( x + \frac{x^3}{2 \cdot 3} + \frac{1 \cdot 3 \cdot x^5}{2 \cdot 4 \cdot 5} + \frac{1 \cdot 3 \cdot 5 \cdot x^7}{2 \cdot 4 \cdot 6 \cdot 7} + \frac{1 \cdot 3 \cdot 5 \cdot 7 \cdot x^9}{2 \cdot 4 \cdot 6 \cdot 8 \cdot 9} + \dots \right),$$

$$|x| < 1$$

$$21. e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \dots, \quad |x| < \infty$$

$$22. \ln|\sin x| = -\ln 2 - \cos 2x - \frac{\cos 4x}{2} - \dots - \frac{\cos 2nx}{n} - \dots, \quad 0 < x^2 < \pi^2$$

$$23. S = -\frac{x^4}{3!4!} + \frac{2!x^6}{5!6!} - \frac{3!x^8}{7!8!} + \dots + (-1)^n \frac{n!x^{2n+2}}{(2n+1)!(2n+2)!} + \dots, \quad |x| < \infty$$

$$24. a^x = 1 + \frac{x \ln a}{1!} + \frac{(x \ln a)^2}{2!} + \dots + \frac{(x \ln a)^n}{n!} + \dots, \quad |x| < \infty$$

$$25. \sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \dots, \quad |x| < \infty$$

$$26. \operatorname{arsh}(x) = x - \frac{1}{2} \frac{x^3}{3} + \frac{1}{2} \frac{3}{4} \frac{x^5}{5} - \frac{1}{2} \frac{3}{4} \frac{5}{6} \frac{x^7}{7} + \dots, \quad |x| < 1$$

$$27. \operatorname{arctg}(x) = \frac{\pi}{2} - \frac{1}{x} + \frac{1}{3x^3} - \frac{1}{5x^5} + \frac{1}{7x^7} - \dots, \quad x > 1$$

$$28. \cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \dots + \frac{(-1)^{n+1} x^{2n}}{(2n)!} - \dots, \quad |x| < \infty$$

$$29. S = \frac{x}{1!} - \frac{x^2}{2!} + \frac{x^3}{3!} - \dots + \frac{(-1)^{n+1} x^n}{n!} - \dots, \quad -1 < x \leq 1$$

$$30. S = x + \frac{x^3}{3!} + \frac{x^5}{5!} + \frac{x^7}{7!} + \dots, \quad |x| < \infty$$

### 3.4 Порядок выполнения работы

3.4.1 Исследовать функции и определить область допустимых значений аргумента  $X$ , выбрать  $X_{\text{нач}}$ ,  $X_{\text{кон}}$  и шаг  $dX$ .

3.4.2 Разработать структурную схему алгоритма решения задачи в соответствии с заданным вариантом.

3.4.3 Написать программу и выполнить её отладку.

3.4.4 Выполнить вычисления для трёх различных значений точности.

3.4.5 Выявить зависимость количества просуммированных членов ряда от точности.

3.4.6 Проанализировать область значений функции. Насколько это соответствует ожидаемым значениям?

### 3.5 Содержание отчета

Цель работы, описание математического метода решения задачи, структурная схема алгоритма, текст программы, анализ результатов вычислений, выводы.

### 3.6 Контрольные вопросы

3.6.1 Укажите необходимые элементы любого цикла.

3.6.2 Запишите общий формат оператора цикла с параметром.

3.6.3 Приведите пример использования оператора цикла с параметром.

3.6.4 Как с помощью оператора цикла с параметром вывести коды всех строчных литер?

3.6.5 Запишите и объясните общий формат оператора цикла с предусловием.

3.6.6 Запишите и объясните общий формат цикла оператора с постусловием.

3.6.7 Что называют инвариантом цикла?

3.6.8 Приведите примеры использования операторов цикла с постусловием и предусловием.

3.6.9 Какие условия должны выполняться, чтобы циклы `While` и `Repeat-Until` не превратились в бесконечные циклы?



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

### 4.1 Цель работы

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

### 4.2 Краткие теоретические сведения

#### 4.2.1 Описание массивов

Массив представляет собой набор однотипных данных, имеющий общее для всех своих элементов имя.

Тип *массив* относится к группе структурных типов. Элементы массива пронумерованы, и обратиться к каждому из них можно, указав один или несколько *индексов*. *Вектор* – это пример массива, в котором элементы нумеруются одним индексом. Если речь идет о хранении в массиве *таблицы* значений (матрицы), то его элементы нумеруются двумя индексами.

Задание массива (регулярного типа) в языке *Pascal* выполняют с помощью следующей конструкции:

Array [*<тип индекса>*] of [*<тип элемента>*]

Тип элемента массива может быть любым, в том числе и массивом. Это позволяет задавать многомерные массивы. Тип индекса массива может быть либо ограниченным, либо перечисленным, либо литерным, либо логическим. Наиболее часто используют ограниченный тип. Например,

Type

vector = array [1..12] of real;  
spisok = array ['a'..'z'] of integer;

Var X: vector;

Y: spisok;

Здесь в разделе типов (следует после слова *Type*) выполнено задание двух новых типов – *vector* и *spisok*. Каждый из них описывает некоторый одномерный массив. Причём в первом случае индекс задан ограничением на множестве целых чисел, а во втором – на множестве литер. В разделе переменных (следует после слова *Var*) объявлены два массива *X* и *Y* соответствующих типов. Объявление переменных *X* и *Y* массивами приводит к выделению необходимых объёмов памяти для хранения значений элементов массивов.

При обращении к элементу массива индекс массива указывается в квадратных скобках после имени массива. Например: *X*[1], *X*[2], ... *X*[12] или *Y*['a'], *Y*['b'], ... , *Y*['z'].

#### 4.2.2 Поиск в массивах

Напишем программу, которая для целочисленного массива из 100 элементов определяет, сколько положительных элементов располагается между его максималь-

ным 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:

6	-8	15	9	-1	3	5	-10	12	2
МАКС			+		+	+	МИН		

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:

```

Program Demo_Array;
Const n=10;
Var  a: array [1..n] of integer;
      i, imax, imin, count: integer;
      ibeg, iend: integer;
Begin
  {Ввод значений элементов массива}
  writeln ('Введите ', n, ' элементов массива');
  for i:=1 to n do readln(a[i]);
  imax:=1; imin:=1; {Принимаем за максимум и минимум первый элемент}
  {Выполняем поиск максимального и минимального элемента}
  for i:=1 to n do
    begin  if a[i]>a[imax] then imax:=i;
           if a[i]<a[imin] then imin:=i;
    end;
  {отладочная печать}
  writeln ('max=', a[imax], ' min=', a[imin]);
  {определяем границы просмотра массива для поиска положительных
  элементов, находящихся между максимальным и минимальным элементами}
  if imax<imin then ibeg:=imax
    else ibeg:=imin;
  if imax<imin then iend:=imin
    else iend:=imax;
  {отладочная печать} writeln ('ibeg=', ibeg, ' iend=', iend);
  {просматриваем элементы массива}
  count:=0;
  for i:=ibeg+1 to iend-1 do

```

```

        if a[i]>0 then count:=count+1;
        writeln('Количество положительных элементов ', count);
    end.

```

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) *a[imin]* расположен левее *a[imax]*;
- 2) *a[imin]* расположен правее *a[imax]*;
- 3) *a[imin]* и *a[imax]* совпадают.

Последняя ситуация имеет место, когда в массиве все элементы имеют одно и то же значение. Желательно также проверить, как работает программа, если *a[min]* и *a[max]* 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 4.1.

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 4.1 – Сортировка методом пузырька

Ниже представлен фрагмент программы сортировки массива методом «пузырька».

```

Var  i, j: integer;
     a: array[1..n] of integer;
     x: integer;
begin
  for j:=n downto 2 do
    for i:=1 to j-1 do      {сравнение двух соседних элементов}
      if (a[i]>a[i+1]) then
        begin x:=a[i]; {перестановка элементов}
              a[i]:=a[i+1] ;
              a[i+1]:=x;
        end
      end;
    end;
  end;

```

#### *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 4.2 – Сортировка массива методом вставки

Ниже представлен фрагмент программы сортировки массива методом вставки.

```

Var  i, j, x: integer;
     a: array[0..n] of integer;
begin
  for i:=2 to n do
    begin x:=a[i] ; a[0]:=x; j:=i;
          while x<a[j-1] do
            begin a[j] :=a[j-1] ; j:=j-1; end;
          a[j] :=x;
    end;
  end;
end;

```

### 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 4.3 – Сортировка методом прямого выбора

Фрагмент соответствующей процедуры представлен ниже:

```

Var i, j, k: integer;
a: array[1..n] of integer;  x: integer;
begin
  for i:=1 to n-1 do
    begin
      k:=i; x:=a[i];
      for j:=i+1 to n do {поиск минимального элемента}
        if a[j]<x then
          begin  k:=j;
                  x:=a[k];
          end;
      a[k] :=a[i]; {перестановка элементов}
      a[i] :=x;
    end;
  end;
end;

```

### 4.3 Варианты заданий

#### 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.

Удалить все элементы, модуль которых не превышает 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.

Удалить все элементы, модуль которых находится внутри отрезка  $[a, b]$  и упорядочить элементы массива по возрастанию, используя алгоритм сортировки методом вставки.

### **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.

### **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.

**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.

**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  $C$ ;
- 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.

**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  $C$ ;
- 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.

Удалить все элементы, модуль которых находится внутри отрезка  $[a, b]$  и упорядочить элементы массива по возрастанию, используя алгоритм сортировки методом прямого обмена.

### **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 элементы, 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.

4.4.5 Для алгоритма сортировки исследовать зависимость количества операций сравнения и пересылок данных от размера массива. Для этого следует задать значение константы  $n$ , определяющей число элементов массива «с запасом», а фактическое количество введенных элементов запомнить в переменной, которая затем участвует в организации циклов по массиву, задавая фактическую верхнюю границу индекса.

Например:

```
Count  n=1000;
Var    a: array [1..n] of real;
       i, Kol: integer;
Begin
    Writeln ('введите кол-во элементов');
    Readln (Kol);
    If Kol > n then
        Begin('превышение размеров массива');
        Exit {выход из программы}
    end;
    {ввод значений массива}
    For i:=1 to kol do Readln (a[i]);
    ...
```

В этом фрагменте программы переменная Kol используется для хранения фактического количества введенных элементов массива.

Для того чтобы подсчитать количество операций сравнения и пересылок данных, в тексте программы сортировки массива следует добавить соответствующие счётчики. Результаты исследований представить в виде графиков.

#### 4.5 Содержание отчета

Цель работы, описание алгоритма решения задачи, структурная схема алгоритма, текст программы, описание тестовых примеров, анализ результатов вычислений, зависимости количества операций пересылок и сравнения данных в ходе сортировки от размера массива, выводы.

#### 4.6 Контрольные вопросы

4.6.1 Приведите общую форму задания регулярного типа в языке Pascal.

4.6.2 Какие типы индексов допустимы при задании массивов?

4.6.3 Как осуществляется обращение к отдельным элементам массива?

4.6.4 Приведите примеры задания массивов для различных типов индексов.

4.6.5 Напишите фрагмент программы, выполняющий поиск заданного элемента в массиве.

4.6.6 Что такое «поиск с барьером» в массиве?

4.6.7 Объясните алгоритмы сортировки методом пузырька, вставки, прямого выбора.

4.6.8 Напишите программы сортировки методом пузырька, вставки, прямого выбора.

## 5 ЛАБОРАТОРНАЯ РАБОТА №5 «ОБРАБОТКА ДВУМЕРНЫХ МАССИВОВ С ПОМОЩЬЮ ПРОЦЕДУР И ФУНКЦИЙ»

### 5.1 Цель работы

Изучить основные принципы обработки двумерных массивов, получить навыки разработки программ блочной структуры, исследовать способы передачи параметров в процедуры и функции.

### 5.2 Краткие теоретические сведения

#### 5.2.1 Обработка двумерных массивов

При объявлении двумерного массива (матрицы) необходимо задать два индекса. Это можно сделать по-разному. Два варианта описания приводятся ниже:

Var a: array[1..3,1..3] of Real;

Var a: array[1..3] of array [1..3] of Real;

Эти описания эквивалентны. Обращаться к элементам получившегося массива можно по-разному:  $a[3,2]$  или  $a[3][2]$ . Здесь первый индекс соответствует номеру строки массива, а второй – номеру столбца. Массив хранится в непрерывной области памяти ЭВМ по строкам:

$a_{11}$	$a_{12}$	$a_{13}$	$a_{21}$	$a_{22}$	$a_{23}$	$a_{31}$	$a_{32}$	$a_{33}$
← 1-ая строка →			← 2-ая строка →			← 3-ая строка →		

Рисунок 5.1 – Хранение двумерного массива

Строки массива в памяти ЭВМ ничем не отделены одна от другой, то есть прямоугольной матрицей двумерный массив является только в форме визуального представления. В памяти сначала располагаются первая строка, затем вторая и т.д.

Напишем программу, которая для целочисленной матрицы  $10 \times 20$  определяет среднее арифметическое ее элементов и количество положительных элементов в каждой строке.

Алгоритм решения этой задачи очевиден. Для вычисления среднего арифметического элементов массива требуется найти их общую сумму, после чего разделить ее на количество элементов. Порядок просмотра массива роли не играет. Определение положительных элементов каждой строки требует просмотра матрицы по строкам. Обе величины вычисляются при одном просмотре матрицы.

```

Program Demo_matrix;
uses crt;
Const nrow=10; ncol=20;
Var  a: array[1..nrow,1..ncol] of integer;
      n, i, j: integer;
      s: real;

begin
clrscr;
Writeln('Введите элементы массива:');
  For i:=1 to nrow do
    For j:=1 to ncol do
      Readln(a[i,j]);
    s:=0; {сумма элементов}
    For i:=1 to nrow do
      begin n:=0; {счетчик положительных элементов}
        For j:=1 to ncol do
          begin s:=s+a[i,j];
            if a[i,j]>0 then n:=n+1;
          end;
        Writeln('В строке ', i, ' количество положительных элементов = ', n);
      end;
    if nrow*ncol=0 then s:=0 else
      s:=s/nrow*ncol; Writeln('Среднее арифметическое =', s);
    readkey; {ожидание нажатия любой клавиши}
  end.

```

Здесь размерности массива заданы именованными константами `nrow` и `ncol`, что позволяет легко их изменять. Для упрощения отладки рекомендуется задать небольшие значения этих констант. При вычислении количества положительных элементов для каждой строки выполняются однотипные действия: обнуление счетчика `n`, просмотр каждого элемента строки и сравнение его с нулем, при необходимости увеличение счетчика на единицу, а после окончания вычислений – вывод результирующего значения счетчика.

### 5.2.2 Процедуры и функции

Процедуры и функции называют подпрограммами. Применение подпрограмм дает возможность уменьшать число повторений одной и той же последовательности операторов, а также разбивать программу на отдельные подпрограммы. В программе описание процедур и функций должно располагаться между разделами переменных и операторов. Каждая процедура или функция определяется только один раз, но может использоваться многократно. Структура процедур и функций аналогична структуре полной программы на языке Паскаль. В процедурах и функциях могут быть описаны собственные метки, константы, типы, переменные и другие процедуры и функции.

Описание каждой процедуры начинается с заголовка, в котором задается имя процедуры и список формальных параметров с указанием их типов. Процедура может быть без параметров; тогда в заголовке указывается только ее имя. С помощью

параметров осуществляется передача исходных данных в процедуру, а также передача результатов обратно в вызывающую ее программу. Общая форма записи заголовка процедуры:

Procedure <имя> (<список формальных параметров>);

В списке формальных параметров должны быть перечислены имена формальных параметров и их типы, например:

Procedure SB (a: Real; b: Integer; c: Char).

Как видно из примера, параметры в списке отделяются точкой с запятой. *Список формальных параметров* может включать в себя параметры-значения, параметры-переменные (перед ними должно стоять служебное слово **Var**) и параметры-константы (перед ними стоит служебное слово **Const**). После заголовка процедуры следуют ее разделы в том же порядке, что и в программе.

Вызов и выполнение процедуры осуществляется при помощи оператора процедуры:

<имя процедуры>(<список фактических параметров>).

Элементы списка фактических параметров отделяются друг от друга запятой. Между формальными и фактическими параметрами должно быть полное соответствие, т.е. формальных и фактических параметров должно быть одинаковое количество, порядок следования фактических и формальных параметров должен быть один и тот же, тип каждого фактического параметра должен совпадать с типом соответствующего ему формального параметра.

При вызове процедуры происходит передача параметров, при этом параметры-значения передаются по значению, а параметры-переменные – по ссылке. Основное отличие этих способов передачи параметров заключается в том, что присваивание значений формальному параметру-переменной внутри процедуры одновременно выполняется и для соответствующего фактического параметра, так как процедуре передается ссылка (адрес) на область памяти, где располагается фактический параметр. Все действия над формальным параметром-переменной представляют собой действия над фактическим параметром. Параметры-переменные обычно используют для возврата результатов работы процедуры в вызвавшую ее программу.

В случае параметров-значений выполняется подстановка (копирование) значений фактических параметров в формальные параметры. Поэтому параметры, через которые в процедуру передаются исходные данные, обычно являются параметрами-значениями. В процедуре можно изменять формальные параметры-значения, однако соответствующие им фактические параметры останутся без изменений. При вызове процедуры на место параметров-значений можно подставлять выражения.

В случае параметра-константы в подпрограмму также передается адрес области памяти, в которой располагается переменная или вычисляемое значение. Однако компилятор блокирует любые присваивания параметру-константе нового значения в теле подпрограммы.

При выборе вида формальных параметров следует учитывать следующее обстоятельство. Как уже говорилось, при объявлении параметра-значения осуществляется копирование фактического параметра во временную память. Если этим параметром будет массив большой размерности, то существенные затраты времени и

памяти на копирование при многократных обращениях к подпрограмме можно минимизировать, объявив этот параметр параметром-константой. Параметр-константа не копируется во временную область памяти, что сокращает затраты времени на вызов подпрограммы, однако любые изменения в теле подпрограммы невозможны. Если необходимо допустить изменения массива, то следует передавать его как параметр-переменную.

Следует также обратить внимание на то, что в списке формальных параметров может быть указан только стандартный или ранее объявленный тип. Поэтому *нельзя*, например, объявить следующую процедуру:

Procedure S (a: array[1..10] of Real);

так как в списке параметров фактически объявляется ограниченный тип 1..10. В этом случае необходимо первоначально описать тип, соответствующий массиву. Например:

Type vector = array[1..10] of Real;  
Procedure S( Var a: vector);

Процедуры могут возвращать результат в основную программу не только при помощи параметров-переменных, но и непосредственно, изменяя *глобальные переменные*. Переменные, описанные в основной программе, являются глобальными по отношению к внутренним процедурам и функциям. Переменные, описанные внутри процедур и функций, называются *локальными*. Такие переменные создаются при каждом входе в процедуру и уничтожаются при выходе из этой процедуры, т.е. локальные переменные существуют только при выполнении процедуры и недоступны основной программе.

Описание функции в основном аналогично описанию процедуры. Однако имеются некоторые отличия. Результатом работы функции является некоторое значение, возвращаемое в точку вызова функции. Тип результата задается в заголовке функции:

Function <имя функции> (<список формальных параметров>):<тип результата>;

Если функция изменяет значения формальных параметров-переменных или значение глобальных переменных, то говорят, что функция имеет побочный эффект. Применение таких функций нежелательно. Среди входящих в функцию операторов должен обязательно присутствовать хотя бы один оператор присваивания, в левой части которого стоит имя данной функции. Этот оператор и определяет значение, возвращаемое функцией:

<имя функции>:=<результат>;

Ниже приведен пример функции, которая проверяет, является ли квадратная матрица симметричной. (Напомним, что матрица называется симметричной, если элементы, расположенные относительно главной диагонали, равны, т.е.  $A_{ij} = A_{ji}$ , где  $A_{ij}$  – элементы матрицы  $A$ .)

```
...
Const n=100;
Type matrix = array[1..n] of Integer;
```

```

...
Function Symmetric (Var A: matrix): Boolean;
Var i, k: integer;
begin
  Symmetric:=True;
  For j:=1 to n-1 do
    For k:=j+1 to n do
      If A[j,k]<>A[k,j] Then
        begin  Symmetric:=False;
              Exit; {выход из функции}
        end;
    end;
  end;
end;

```

Задача решается путем перебора элементов матрицы и проверкой условия симметричности. При этом проверка не затрагивает элементы главной диагонали. Для проверки симметричности достаточно перебрать значения индексов, соответствующие элементам матрицы, расположенным ниже (или выше) главной диагонали.

### 5.2.3 Пример использования процедур и функций

Пусть требуется написать программу, которая упорядочивает строки прямоугольной целочисленной матрицы по возрастанию сумм их элементов. Начинать решение задачи необходимо с четкого описания того, что является ее исходными данными и результатами, и каким образом они будут представлены в программе.

*Исходные данные.* Размерность матрицы будем задавать с помощью символических констант. В качестве типа значений элементов матрицы будем использовать тип **Integer**.

*Результаты.* Результатом является та же матрица, но упорядоченная. Это значит, что не следует отводить для результата новую область памяти, а необходимо упорядочить матрицу на том же месте.

*Промежуточные величины.* Кроме конечных результатов, в любой программе есть промежуточные, а также служебные переменные. Следует выбрать их тип и способ хранения. Очевидно, что если требуется упорядочить матрицу по возрастанию сумм элементов ее строк, эти суммы необходимо вычислить и где-то хранить. Поскольку все они потребуются при упорядочивании, их запишем в массив, количество элементов которого соответствует количеству строк матрицы, а  $i$ -ый элемент содержит сумму элементов  $i$ -ой строки. Сумма элементов строки может превысить диапазон значений, допустимых для отдельного элемента строки, поэтому для элементов этого массива необходимо выбрать тип **LongInt**.

После того, как выбраны структуры данных, можно приступить к разработке алгоритма, поскольку алгоритм зависит от того, каким образом представлены данные.

*Алгоритм работы программы.* Для сортировки строк воспользуемся алгоритмом прямого выбора (см. лабораторную работу №4). При этом будем одновременно с перестановкой элементов массива выполнять обмен двух строк матрицы. Вычисления в данном случае состоят из 2-х шагов: формирование сумм элементов каждой строки и упорядочивание матрицы. Упорядочивание состоит в выборе наименьшего



элемента и обмена с первым из рассматриваемых. Разработанные алгоритмы полезно представить в виде структурной схемы. Функционально завершение части алгоритма отделяется пустой строкой и/или комментарием. Не следует стремиться написать всю программу сразу. Сначала рекомендуется написать и отладить фрагмент, содержащий ввод исходных данных. Затем целесообразно перейти к следующему фрагменту алгоритма.

Ниже приведен текст программы сортировки строк матрицы по возрастанию сумм элементов:

```

Program Sort_matrix;
uses crt;
Const nrow=5; ncol=3;
Type matrix=array[1..nrow,1..ncol] of Integer;
vector=array[1..nrow] of LongInt;
Var      a:matrix;      v:vector;
{----- процедура ввода исходной матрицы -----}
Procedure Vvod (Var a:matrix);
Var i, j: Integer;
Begin
  Writeln('Введите элементы массива:');
  For i:=1 to nrow do
    For j:=1 to ncol do
      Readln(a[i,j]);
    End;
  End;
{-----процедура вычисления суммы элементов строк -----}
Procedure SummaStrok(Const a:matrix; Var v:vector);
Var i, j: Integer;
Begin
  For i:=1 to nrow do
    Begin  v[i]:=0;
           For j:=1 to ncol do  v[i]:=v[i]+a[i,j];
        End;
  End;
{-----процедура контрольной печати-----}
Procedure Vyvod (Const a:matrix; Const v:vector);
Var i, j:Integer;
Begin
  For i:=1 to nrow do
    Begin  For j:=1 to ncol do Write(a[i,j], ' ');
           Writeln;
        End;
    For i:=1 to nrow do Write(v[i], ' ');
  Writeln;
End;
{----процедура сортировки строк матрицы методом прямого выбора-----}
Procedure Sort(Var a:matrix; Var v:vector);

```

```

Var  buf_sum: LongInt; min, buf_a: Integer;  i, j: Integer;
Begin
  For i:=1 to nrow-1 do
    Begin
      min:=i;
      for j:=i+1 to nrow do
        if v[j]<v[min] then min:=j;
      buf_sum:=v[i];
      v[i]:=v[min];
      v[min]:=buf_sum;
      for j:=1 to ncol do
        begin
          buf_a:=a[i,j];
          a[i,j]:=a[min,j];
          a[min,j]:=buf_a;
        end;
      End;
    End;
  End;
  {-----основная программа-----}
Begin
  Vvod(a); {Ввод матрицы}
  SummaStrok(a,v); {вычисление суммы элементов строк}
  Vyvod(a,v); {контрольная печать}
  Sort(a,v); {сортировка строк по возрастанию сумм элементов}
  Vyvod(a,v); {вывод результатов}
  Readkey;
End.

```

В процедуре **Sort** используются две буферные переменные: **buf\_sum**, через которую осуществляется обмен двух значений сумм (имеет такой же тип, что и сумма), **buf\_a** для обмена значений элементов массива (того же типа, что и элементы массива).

### 5.3 Варианты заданий

Каждый пункт нижеприведенного задания оформить в виде функции (процедуры). Все необходимые данные для функций (процедур) должны передаваться им в качестве параметров. Ввод-вывод данных и результатов также организовать с помощью соответствующих процедур.

#### Вариант 1

Дана целочисленная прямоугольная матрица. Определить:

- 1) количество строк, не содержащих ни одного нулевого элемента;
- 2) встречается ли более одного раза максимальное из чисел в заданной матрице.

#### Вариант 2

Дана целочисленная прямоугольная матрица. Определить количество столбцов, не содержащих ни одного нулевого элемента.

Характеристикой строки целочисленной матрицы назовем сумму ее положи-

тельных четных элементов. Переставляя строки заданной матрицы, расположить их в соответствии с ростом характеристик.

### Вариант 3

Дана целочисленная прямоугольная матрица. Определить:

- 1) количество столбцов, содержащих хотя бы один нулевой элемент;
- 2) номер строки, в которой находится самая длинная серия одинаковых элементов.

### Вариант 4

Дана целочисленная квадратная матрица. Определить:

- 1) произведение элементов в тех строках, которые не содержат отрицательных элементов;
- 2) максимум среди сумм элементов диагоналей, параллельных главной диагонали матрицы.

### Вариант 5

Дана целочисленная квадратная матрица. Определить:

- 1) сумму элементов в тех столбцах, которые не содержат отрицательных элементов;
- 2) минимум среди сумм модулей элементов диагоналей, параллельных главной диагонали матрицы.

### Вариант 6

Дана целочисленная прямоугольная матрица. Определить:

- 1) сумму элементов в тех строках, которые содержат хотя бы один отрицательный элемент;
- 2) номера строк и столбцов всех седловых точек матрицы.

*Примечание.* Матрица  $A$  имеет седловую точку  $a_{ij}$ , если  $a_{ij}$  является минимальным элементом в  $i$ -й строке и максимальным в  $j$ -м столбце.

### Вариант 7

Найти сумму элементов в тех строках, которые содержат хотя бы один отрицательный элемент.

Для заданной матрицы найти такие  $k$ , что  $k$ -я строка матрицы совпадает с  $k$ -м столбцом.

### Вариант 8

Найти сумму элементов в тех столбцах, которые содержат хотя бы один отрицательный элемент.

Характеристикой столбца целочисленной матрицы назовем сумму модулей его отрицательных нечетных элементов. Переставляя столбцы заданной матрицы, расположить их в соответствии с ростом характеристик.

### Вариант 9

Найти сумму модулей элементов, расположенных ниже главной диагонали.

Определить, является ли заданная квадратная матрица  $n \times n$  симметричной, относительно главной диагонали.

### Вариант 10

Дана целочисленная квадратная матрица. Определить:

- 1) номер строки с максимальной суммой;

2) произведение модулей элементов, расположенных выше главной диагонали.

### **Вариант 11**

Найти сумму модулей элементов, расположенных ниже главной диагонали.

Дана целочисленная квадратная матрица. Поменять местами  $k$ -ю строку с  $k$ -м столбцом.

### **Вариант 12**

Найти номер первой из строк, содержащих хотя бы один положительный элемент.

Уплотнить заданную матрицу, удаляя из нее строки и столбцы, заполненные нулями.

### **Вариант 13**

Найти сумму элементов в тех строках, которые содержат хотя бы один отрицательный элемент.

Осуществить циклический сдвиг элементов прямоугольной матрицы на  $n$  элементов вправо или влево (в зависимости от введенного режима),  $n$  может быть больше количества элементов в строке или столбце.

### **Вариант 14**

Найти сумму элементов в тех столбцах, которые содержат хотя бы один отрицательный элемент.

Осуществить циклический сдвиг элементов прямоугольной матрицы на  $n$  элементов вниз или вверх (в зависимости от введенного режима),  $n$  может быть больше количества элементов в строке или столбце.

### **Вариант 15**

Дана целочисленная прямоугольная матрица. Определить номер первого из столбцов, содержащих хотя бы один нулевой элемент.

Характеристикой строки целочисленной матрицы назовем сумму ее отрицательных четных элементов. Переставляя строки заданной матрицы, расположить их в соответствии с убыванием характеристик.

### **Вариант 16**

Найти номер первого из столбцов, не содержащих ни одного отрицательного элемента.

Упорядочить строки целочисленной прямоугольной матрицы по возрастанию количества одинаковых элементов в каждой строке.

### **Вариант 17**

Найти номер первой из строк, не содержащих ни одного положительного элемента.

Путем перестановки элементов квадратной вещественной матрицы добиться того, чтобы ее максимальный элемент находился в левом верхнем углу, следующий по величине – в позиции (2,2), следующий по величине – в позиции (3,3) и т.д., заполнив таким образом всю главную диагональ.

### **Вариант 18**

Дана целочисленная прямоугольная матрица. Определить:

- 1) количество строк, содержащих хотя бы один нулевой элемент;
- 2) номер столбца, в котором находится самая длинная серия одинаковых элементов.

**Вариант 19**

Дана целочисленная квадратная матрица. Определить:

- 1) сумму элементов в тех строках, которые не содержат отрицательных элементов;
- 2) минимум среди сумм элементов диагоналей, параллельных главной диагонали матрицы.

**Вариант 20**

Дана целочисленная прямоугольная матрица. Определить:

- 1) количество отрицательных элементов в тех строках, которые содержат хотя бы один нулевой элемент;
- 2) номера строк и столбцов всех седловых точек матрицы.

*Примечание.* Матрица  $A$  имеет седловую точку  $a_{ij}$ , если  $a_{ij}$  является минимальным элементом в  $i$ -й строке и максимальным в  $j$ -м столбце.

**Вариант 21**

Дана прямоугольная матрица. Выполнить:

- 1) поиск позиций всех седловых точек матрицы;
- 2) зеркальную перестановку столбцов матрицы относительно вертикальной оси;

*Примечание.* Позицией элемента матрицы называется упорядоченная пара  $(i, j)$ , где  $i$  – номер строки элемента,  $j$  – номер столбца. Матрица  $A$  имеет седловую точку  $a_{ij}$ , если  $a_{ij}$  является минимальным элементом в  $i$ -й строке и максимальным в  $j$ -м столбце.

**Вариант 22**

Дана прямоугольная матрица. Выполнить:

- 1) поиск позиций всех локальных максимумов матрицы;
- 2) зеркальную перестановку строк матрицы относительно горизонтальной оси;

*Примечание.* Позицией элемента матрицы называется упорядоченная пара  $(i, j)$ , где  $i$  – номер строки элемента,  $j$  – номер столбца. Соседями элемента  $a_{ij}$  в матрице  $A$  назовем элементы  $a_{kl}$ , для которых  $i-1 \leq k \leq i+1$ ,  $j-1 \leq l \leq j+1$ . Элемент матрицы называется локальным максимумом, если он строго больше всех имеющихся у него соседей.

**Вариант 23**

Дана прямоугольная матрица. Выполнить:

- 1) поиск позиций всех локальных минимумов матрицы;
- 2) зеркальную перестановку столбцов матрицы относительно вертикальной оси;

*Примечание.* Позицией элемента матрицы называется упорядоченная пара  $(i, j)$ , где  $i$  – номер строки элемента,  $j$  – номер столбца. Соседями элемента  $a_{ij}$  в матрице  $A$  назовем элементы  $a_{kl}$ , для которых  $i-1 \leq k \leq i+1$ ,  $j-1 \leq l \leq j+1$ . Эле-

мент матрицы называется локальным минимумом, если он строго меньше всех имеющих у него соседей.

#### Вариант 24

Дана квадратная матрица. Выполнить поворот этой матрицы на  $90 \times k$  градусов, где  $k$  – целое число. Осуществить поиск позиций максимального и минимального элементов матрицы.

*Примечание.* Позицией элемента матрицы называется упорядоченная пара  $(i, j)$ , где  $i$  – номер строки элемента,  $j$  – номер столбца.

#### Вариант 25

Определить, является ли заданная квадратная матрица симметрической и тёплицевой.

*Примечание.* Матрица  $A$  является симметрической, если  $A^T = A$ . Матрица называется тёплицевой (матрица Тёплица), если все ее элементы, расположенные на каждой диагонали, равны.

#### Вариант 26

Определить, является ли заданная квадратная матрица антисимметрической и тёплицевой.

*Примечание.* Матрица  $A$  является антисимметрической, если  $A^T = -A$ . Матрица называется тёплицевой (матрица Тёплица), если все ее элементы, расположенные на каждой диагонали, равны.

#### Вариант 27

Определить, является ли заданная квадратная матрица  $n \times n$  матрицей Адамара порядка  $n$ .

*Примечание 1.* Матрица Адамара  $H$  порядка  $n$  – это квадратная матрица размера  $n \times n$ , составленная из чисел 1 и  $-1$ , столбцы которой ортогональны, так что справедливо соотношение  $H^T H = nI$ , где  $I$  – единичная квадратная матрица размера  $n \times n$ . Матрица Адамара размера  $n \times n$  при  $n > 2$  существует, только если  $n$  делится на 4 без остатка. Ниже представлен пример матрицы Адамара 4-го порядка:

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix}.$$

*Примечание 2.* Ортогональность векторов одинаковой длины означает равенство нулю скалярного произведения последних.

#### Вариант 28

Определить, является ли заданная квадратная матрица размера  $n \times n$  состоящей из целых чисел от 1 до  $n^2$ , встречающихся по одному разу. Определить позиции минимального и максимального элементов этой матрицы.

*Примечание.* Позицией элемента матрицы называется упорядоченная пара  $(i, j)$ , где  $i$  – номер строки элемента,  $j$  – номер столбца.

### Вариант 29

Определить, является ли заданная квадратная матрица размера  $n \times n$ ,  $n \geq 3$ , магическим квадратом. Определить позиции минимального и максимального элементов этой матрицы.

*Примечание 1.* Магический квадрат порядка  $n$  ( $n \geq 3$ ) представляет собой квадратную матрицу  $n \times n$ , состоящую из целых чисел от 1 до  $n^2$ , в которой суммы элементов по строкам, столбцам и главным диагоналям равны одному и тому же числу. Ниже представлен магический квадрат 4-го порядка:

$$\begin{pmatrix} 16 & 2 & 3 & 13 \\ 5 & 11 & 10 & 8 \\ 9 & 7 & 6 & 12 \\ 4 & 14 & 15 & 1 \end{pmatrix}.$$

*Примечание 2.* Позицией элемента матрицы называется упорядоченная пара  $(i, j)$ , где  $i$  – номер строки элемента,  $j$  – номер столбца.

### Вариант 30

Выполнить построение матрицы Паскаля порядка  $n$ . Повернуть построенную матрицу на 90 градусов против часовой стрелки.

*Примечание.* Матрица Паскаля порядка  $n$  представляет собой симметрическую положительно определенную квадратную матрицу размера  $n \times n$  с целочисленными элементами, взятыми из треугольника Паскаля. Ниже представлена матрица Паскаля 4-го порядка:

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 \\ 1 & 3 & 6 & 10 \\ 1 & 4 & 10 & 20 \end{pmatrix}.$$

## 5.4 Порядок выполнения работы

5.4.1 Выбрать типы данных для хранения исходных данных, промежуточных величин и результатов.

5.4.2 Разработать алгоритм решения задачи в соответствии с вариантом.

5.4.3 Составить программу, оформив ввод данных, вывод результатов, необходимые вычисления в виде функций или процедур.

5.4.4 Разработать тестовые примеры, которые проверяют все ветви алгоритма и возможные диапазоны данных. В качестве тестового примера рекомендуется ввести значения элементов массива, близкие к максимально возможным. Дополнительно рекомендуется проверить работу программы, когда массив состоит из одной или двух строк (столбцов), поскольку многие ошибки при написании циклов связаны с неверным указанием их граничных значений.

5.4.5 Выполнить отладку программы.

5.4.6 Получить результаты для всех вариантов тестовых примеров.

## 5.5 Содержание отчета

Цель работы, вариант задания, описание алгоритма решения задачи на естественном языке, структурные схемы алгоритмов всех процедур (функций) и основной программы, описание тестовых примеров, текст программы, анализ результатов вычислений, выводы.

## 5.6 Контрольные вопросы

5.6.1 Приведите пример задания двумерного массива в языке Паскаль.

5.6.2 Как выполняется обращение к элементам двумерного массива?

5.6.3 Как хранится в памяти ЭВМ двумерный массив?

5.6.4 Приведите общий формат описания процедуры без параметров и с параметрами.

5.6.5 Приведите общий формат описания функции без параметров и с параметрами.

5.6.6 Каким образом функция возвращает значение?

5.6.7 Как записываются списки формальных и фактических параметров?

5.6.8 Как передаются параметры-значения?

5.6.9 Как передаются параметры-переменные?

5.6.10 Как передаются параметры-константы?

5.6.11 Что понимают под глобальными и локальными переменными?

5.6.12 Как передать в процедуру значения, используя механизм глобальных параметров?

5.6.13 Как передать в процедуру или функцию массив?



## 6 ЛАБОРАТОРНАЯ РАБОТА №6

### “ПРОГРАММИРОВАНИЕ ОПЕРАЦИЙ НАД СТРОКАМИ И ТЕКСТОВЫМИ ФАЙЛАМИ”

#### 6.1 Цель работы

Изучение основных операций над строками и файлами, программирование операций обработки строк текстовых файлов, исследование свойств файловых переменных.

#### 6.2 Краткие теоретические сведения

##### 6.2.1 Множества

В задачах обработки строк часто используют множества. *Множеством* в языке Pascal называется конечный набор однотипных данных. От массива множество отличается отсутствием индексации его элементов. При описании множества используется зарезервированное слово **set**:

Type <имя типа> = set of <базовый тип>;

Базовым типом множества может быть любой упорядоченный простой тип, состоящий не более чем из 256 элементов. Пример описания множества:

Type Bukvi = set of char;

Допустимыми значениями множественного типа данных являются все возможные подмножества, составленные из значений базового типа. Если переменная типа “множество” описана как **set of 1..3**, то она может принимать значения из следующего набора: {(1, 2, 3), (1,2), (1, 3), (2, 3), (1), (2), (3), 0}.

Значения переменных множественного типа задаются с помощью конструктора, состоящего из квадратных скобок, в которых перечисляются элементы множества или указывается диапазон их значений. Примеры:

Const

```
alphabet = ['A'..'Z', 'a'..'z'];
empty = [];
digits = [0..9];
```

Отсутствие элементов в квадратных скобках означает пустое множество. Зарезервированное слово **in** используется для определения принадлежности элемента множеству:

Var ch: char;

If ch in alphabet then ...

Над множествами языка Pascal можно выполнять те же операции, что и над математическими множествами, то есть находить их объединение, пересечение, разность. Если **S1** и **S2** – константы или переменные множественного типа, то **S1 + S2** будет их объединением, **S1 \* S2** – пересечением, **S1 - S2** – разностью. К множествам могут применяться операции отношений: **=** (равенство), **<>** (неравенство), **<=** (является подмножеством), **>=** (является надмножеством).

Чтобы добавить во множество какой-нибудь элемент, следует объединить его с множеством, состоящим из единственного элемента

$S := S + [ch];$

или использовать процедуру `Include(S, ch)`. Для исключения элемента из множества применяется процедура `Exclude`. У этой процедуры первый параметр означает множество, а второй – исключаемый элемент.

Рассмотрим пример. Пусть требуется создать множество из литер, найти в нем элемент с минимальным кодом, распечатать все элементы множества в алфавитном порядке. Выход из процедуры создания множества будем осуществлять вводом символа '\*'. Работу программы организуем с использованием процедур, вызываемых из пунктов меню.

```

Program DemoSet;
Uses Crt; { Указание имени модуля, содержащего процедуру ClrScr}
Var S: Set of 'A' .. 'Z'; { описание множества }
    C: Char;
    Number: Integer;
Procedure Konstr; {Процедура создания множества}
begin
    S:=[];
    Writeln('Вводите буквы от A до Z');
    Readln(C);
    While C<> '*' Do {Вводим литеры во множество, до тех пор пока
                        не будет введен символ '*'}
        begin
            S:=S+[C]; {Добавление элемента во множество с помощью
                        операции объединения двух множеств S и [C]}
            Readln(C)
        end
    end;
Procedure Smin; {Процедура поиска литеры с минимальным
                  кодом, входящей во множество S}
begin
    C:='A';
    While not (C in S) Do
        C:=Succ(C);
    Writeln('Литера с минимал. кодом, входящая в множество S ', C);
    Writeln('Нажмите клавишу Enter'); Readkey;
end;
Procedure Rasp; { Процедура печати элементов множества S в
                  алфавитном порядке}
begin
    Writeln('Литеры, входящие в множество:');
    For C:='A' To 'z' Do
        If C in S Then Writeln(C);

```

```

Writeln('Нажмите клавишу Enter'); Readkey;
end;
{=====основная программа=====}
begin
{бесконечный цикл, обеспечивающий вывод на экран Пунктов меню}
  While True Do
    Begin
      ClrScr; { очистка экрана }
      Writeln('1-Формирование множества');
      Writeln('2-Поиск элемента с мин. кодом');
      Writeln('3-Печать множества');
      Writeln('4-Выход');
      Writeln('-----');
      Writeln('Введите номер пункта меню');
      Readln(Number);
      Case Number of { вызов необходимой процедуры по номеру}
        1:Konstr;
        2:Smin;
        3:Rasp;
        4:Exit { Встроенная процедура выхода}
      End
    End.

```

### 6.2.2 Строковый тип

*Строка* представляет собой последовательность литер и в стандартном языке Pascal описывается в виде упакованного одномерного массива, компоненты которого есть литеры, а тип индекса задается диапазоном  $1..N$ , где  $N > 1$ :

```
Var stroka: packed array[1..N] of char;
```

В языке Turbo Pascal строковый тип может быть также описан с помощью зарезервированного слова **String**. Пример описания строковой переменной в языке Turbo Pascal:

```
Var stroka1: string[10];
```

Целое число в квадратных скобках задает длину строки. Максимально допустимая длина строки типа **string** составляет 255 символов и назначается строковой переменной по умолчанию, если длина не указана. В примере резервируется для переменной **stroka1** одиннадцать байтов памяти. Дополнительный байт содержит фактическую длину строковой переменной. Длина строки равна значению нулевого элемента массива, т.е. **stroka1[0]**.

В языке Pascal имеется набор процедур и функций для работы со строками. Строке можно присваивать значение строковой константы:

```
stroka1:='ABC';
```

К строкам можно применять операцию *конкатенации*, которая обозначается знаком "+". Конкатенация – это объединение строк:

```
stroka2:= stroka1 + 'DEF';
```

Результатом такой последовательности операторов будет строка 'ABCDEF'. Длина строковой переменной *stroka2* должна задаваться с учетом суммарной длины слагаемых. Строковая константа в исходном тексте программы должна размещаться в пределах одной строки текста. Чтобы задать длинное строковое значение, занимающее в тексте программы несколько строк, можно воспользоваться операцией конкатенации. Операцию слияния строк *str1* и *str2* выполняет и функция `Concat(str1, str2)`.

Функция `Length(str)` определяет длину аргумента *str* строкового типа.

Процедура `Delete(str, start, nrem)` используется для того, чтобы удалить *nrem* символов в строковой переменной *str*, начиная с позиции *start*.

Имеется также процедура `Insert`, предназначенная для вставки одной строки в другую. Чтобы вставить строку (константу или значение строковой переменной) *instr* в строковую переменную *str*, начиная с позиции *start*, данную процедуру следует вызвать следующим образом:

`Insert(instr, str, start);`

Функция `Copy(str1, start, n)` копирует *n* символов строки *str1*, начиная с позиции *start*. Это строковое значение можно присвоить другой строковой переменной.

Функция `Pos(substr, str)` определяет начальную позицию подстроки *substr* в строке *str*. Например, результат следующего вызова равен 5:

`Pos(' F(x) ', ' LetF(x) = 2x')`

Если подстрока не встретилась в строке, значение `Pos` равно нулю.

Имеются две процедуры, преобразующие числовые значения в строковые и обратно – `Str` и `Val`. Процедура `Str` должна вызываться следующим образом:

`Str(num, strnum);`

Здесь *num* – значение числового типа, а *strnum* – переменная строкового типа. Процедура присваивает переменной *strnum* строковое значение, представляющее собой символьное представление значения переменной *num*. Процедура `Val` выполняет обратное преобразование. Обращение к ней имеет вид:

`Val(strnum, num, errcode);`

Третий параметр в этой процедуре равен нулю при успешном выполнении преобразования. В том случае, когда первый параметр содержит символы, недопустимые при записи числа, значение параметра *errcode* по окончании работы процедуры будет равно номеру позиции с ошибочно заданным символом.

Две строки можно сравнивать при помощи операций `>`, `<`, `<=`, `>=`. При этом сравниваются коды символов, начиная с первых символов строк.

### 6.2.3 Файловый тип

*Файловый тип* языка Паскаль представляет собой последовательность одно-типных элементов. Число элементов в определении файлового типа, в отличие от массива, не фиксируется. В зависимости от типа элементов файлы подразделяются на *текстовые и бинарные*. Текстовые файлы предназначены для хранения текстов (например, текстов программ на языке Pascal), а бинарные используются для хранения данных определенных типов.

*Текстовый файл* содержит последовательность символов, организованных в строки. Каждая строка файла заканчивается *меткой конца строки*, состоящей из

пары управляющих символов: возврат каретки  $CR = \#13$  и перевод строки  $LF = \#10$ . Заканчивается файл меткой конец файла (управляющий символ  $\#26$ ).

Для описания текстовых файлов в языке Pascal используется стандартный файловый тип `text`. Прежде чем приступить к операциям над текстовыми файлами, необходимо описать файловую переменную типа `text`:

```
Var in_file: text;
```

Процедура `Assign` связывает имя внешнего файла (файла, который физически расположен на жестком диске) с файловой переменной. Например:

```
Assign(in_file, 'C:\work\ivanov\my_file');
```

Здесь `my_file` имя внешнего файла (физический файл), а `in_file` – файловая переменная (логический файл), используемая в программе в качестве параметра процедур работы с файлами.

После установления связи внешнего файла с файловой переменной, файл надо открыть для записи или чтения. При открытии файла выполняются необходимые системные операции, подготавливающие файл к записи или считыванию информации. Текстовый файл `in_file` открывается процедурой `Reset(in_file)` только для чтения, а процедурой `Rewrite(in_file)` – только для записи. После открытия файла процедурой `Rewrite` файл считается пустым. Если файл с таким именем уже существовал, то данные, хранившиеся ранее в этом файле, становятся недоступными. Если требуется открыть существующий файл в режиме записи с возможностью дописывания данных в конец, то используют процедуру `Append(<имя файловой переменной>)`. По завершении обработки файла программой он должен быть закрыт. После закрытия файла связанный с ним внешний файл обновляется, а файловая переменная может быть связана с другим внешним файлом. Закрывается файл с помощью процедуры `Close(<имя файловой переменной>)`.

Доступ к текстовому файлу организуется последовательно, то есть программа не может в любой момент времени считать из него произвольную порцию информации или произвести запись в произвольное место файла. Файл представляет собой линейную последовательность элементов, каждый из которых имеет свой номер. Указатель файла указывает на текущую позицию файла, из которой выполняется чтение значений или запись. Указатель файла при считывании очередного элемента файла перемещается к следующему элементу.

К файловым переменным применима стандартная функция `Eof(<файловая переменная>)`, которая принимает значение `TRUE`, если указатель файла указывает на конец файла, и значение `FALSE` в ином случае.

Для записи значений в файл или чтения из него используются соответственно процедуры:

```
Write(<файловая переменная>, <список вывода>);  
Read(<файловая переменная>, <список ввода>).
```

Например: `Read(in_file, X, Y)`. Здесь список ввода представлен переменными `X` и `Y`, которым присваиваются значения двух очередных элементов из файла, связанного с файловой переменной `in_file`.

Запись признака конца строки в текстовый файл выполняется процедурой `WriteLn(<файловая переменная>)`.

Для обнаружения признака конца строки при чтении текстового файла используется встроенная функция `Eoln(<файловая переменная>)`, принимающая значение `TRUE`, если указатель текущей позиции указывает на метку конец строки и значение `FALSE` в противном случае. Процедура `Readln(<файловая переменная>, <список ввода>)` пропускает все символы до начала следующей строки текстового файла (т.е. переводит указатель на новую строку), а затем присваивает значения в соответствии со списком ввода.

Текстовый файл может использоваться для хранения числовых значений. При считывании таких значений из файла или их записи в файл происходит автоматическое преобразование из числового формата в символьный и наоборот.

При обращениях к стандартным функциям и процедурам ввода-вывода автоматически производится проверка на наличие ошибок. При обнаружении ошибки программа прекращает работу и выводит на экран соответствующее сообщение. С помощью директив компилятора `{I+}` и `{I-}` автоматическую проверку ошибок ввода-вывода можно включить или выключить. Если автоматическая проверка отключена, ошибки ввода-вывода, возникающие при работе программы, не приводят к ее останову. Встроенная функция `IOResult` возвращает код ошибки. Нулевое значение кода ошибки означает нормальное завершение операции ввода-вывода.

#### 6.2.4 Пример программы обработки строк файлов

Пусть требуется написать программу, которая считывает заданный текстовый файл и выводит номера строк, в которых содержится введенное пользователем слово. Длина строки текста не превышает 80 символов. В программировании *словом* принято называть последовательность символов, ограниченную разделителями. В качестве разделителей используются пробелы, запятые, точки, двоеточия, точки с запятой. Текст не содержит переносов слов.

В предыдущих лабораторных работах был рассмотрен общий порядок действий при создании программы. Будем придерживаться его и впредь.

##### Исходные данные и результаты.

Исходные данные:

1 Текстовый файл неизвестного размера, состоящий из строк длиной не более 80 символов. Поскольку по условию переносы отсутствуют, можно ограничиться поиском слова в каждой строке текста отдельно. Для ее хранения введем строковую переменную длиной 80 символов.

2 Слово-образец, вводимое для поиска с клавиатуры. Для его хранения также выделим строку длиной 80 символов.

Результатом работы программы является номер строки, в которую входит слово. Представим его в программе целым значением.

Для хранения длины строки будем использовать именованную константу, а для хранения фактического количества символов в строке – переменную целого типа. Для работы с файлом потребуется файловая переменная соответствующего типа.

##### Алгоритм решения задачи.

1 Построчно считывать текст из файла.

2 Просматривать каждую строку и искать в ней первое вхождение слова.

При каждом обнаружении слова печатать номер строки и саму строку.

Детализируем второй пункт алгоритма. Для обнаружения слова организуем цикл просмотра символов считанной строки. Если очередной символ строки не является разделителем, то добавляем его в строку, которая предназначена для хранения слова, считываемого из текущей строки текста. Когда очередной символ строки разделитель, то это означает, что в строке выделено очередное слово и его можно сравнить со словом-образцом.

### **Программа и тестовые примеры.**

Текст программы с комментариями приведен ниже.

```

Program search_word_in_file;
Const
    delimiters = [' ','.',',',';',':',';']; {разделители слов}
    len=80; {максимальная длина строки файла}
Var   in_file : text;
      pattern_word,           {введенное слово-образец}
      word_intext,            {слово, найденное в тексте}
      stroka :string[len];    {считываемая строка файла}
      k, n, strokalength : word;
      file_name : string[20];
Begin
    WriteLn( 'Введите имя текстового файла');WriteLn;
    ReadLn(file_name);
    WriteLn('Введите образец для поиска');WriteLn;
    ReadLn(pattern_word);
    {Отключение автоматической проверки ошибок ввода-вывода}
    {$I-}
    Assign(in_file, file_name);
    Reset(in_file);
    If IOResult > 0 then Halt; {Если ошибка открытия файла, то останов}
    {чтение строк до конца файла}
    n := 0;
    while not Eof(in_file) do
    begin
        Inc(n); {увеличить номер строки}
        ReadLn(in_file, stroka);
        strokalength := Length(stroka); {запомнить длину текущей строки}
        {выделение слова из текущей строки}
        k := 1;
        word_intext := '';
        while k <= strokalength do
        begin
            if (stroka[k] in delimiters) or (k=strokalength) then
            begin {обнаружен конец слова}
                if k=strokalength then {включение последней буквы в слово}
                    word_intext := word_intext+stroka[k];
            end
        end
    end

```

```

{проверка совпадения с образцом}
if word_intext = pattern_word then
begin
    WriteLn('Слово ', pattern_word,' найдено в строке ', n:4, ':');
    WriteLn(stroka);
    Break; {слово-образец обнаружено, прервать поиск}
end;
word_intext := "";
end
else {если не конец слова, то добавить букву в слово}
    word_intext := word_intext+stroka[k];
Inc(k);
end;
end;
Close(in_file);
WriteLn('Для завершения работы нажмите <Enter>'); Readkey;
end.

```

Перечень символов-разделителей содержится в константе множественного типа **delimiters**. После того как пользователем заданы имя текстового файла (в данном случае путь доступа к файлу должен быть указан полностью, вместе с расширением), а также слово для поиска, указанный файл открывается, и строки текста последовательно считываются из него. В каждой считанной строке выделяются слова, которые сравниваются с образцом для поиска. Это делается до тех пор, пока не будет найдено искомое слово, либо будут перебраны все слова, принадлежащие текущей строке. Все необходимые для этого операции выполняются во внутреннем цикле **while...do** и условных операторах. Проверка прекращается, если обнаружено первое появление слова в строке. В этом случае на печать выводятся номер строки и сама строка.

Для тестирования программы требуется создать файл с текстом, в котором заданное слово встречается:

- в начале строки;
- в конце строки;
- в середине строки;
- несколько раз в одной строке;
- как часть других слов, находящаяся в начале, середине и конце этих слов;
- между различными разделителями.

Длина хотя бы одной из строк должна быть равна 80 символам. Для тестирования программы следует выполнить ее, по крайней мере, два раза: введя с клавиатуры слово, содержащееся в файле, и слово, которого в нем нет.

### 6.3 Варианты заданий

#### Вариант 1

Написать программу, которая считывает из текстового файла три предложения и выводит их в обратном порядке.

#### Вариант 2

Написать программу, которая считывает текст из файла и выводит на экран



только предложения, содержащие введенное с клавиатуры слово.

### **Вариант 3**

Написать программу, которая считывает текст из файла и выводит на экран только строки, содержащие двузначные числа.

### **Вариант 4**

Написать программу, которая считывает английский текст из файла и выводит на экран слова, начинающиеся с гласных букв.

### **Вариант 5**

Написать программу, которая считывает текст из файла и выводит его на экран, меняя местами каждые два соседних слова.

### **Вариант 6**

Написать программу, которая считывает текст из файла и выводит на экран только предложения, не содержащие запятых.

### **Вариант 7**

Написать программу, которая считывает текст из файла и определяет, сколько в нем слов, состоящих не более чем из четырех букв.

### **Вариант 8**

Написать программу, которая считывает текст из файла и выводит на экран только цитаты, то есть предложения, заключенные в кавычки.

### **Вариант 9**

Написать программу, которая считывает текст из файла и выводит на экран только предложения, состоящие из заданного количества слов.

### **Вариант 10**

Написать программу, которая считывает английский текст из файла и выводит на экран слова текста, начинающиеся и оканчивающиеся на гласные буквы.

### **Вариант 11**

Написать программу, которая считывает текст из файла и выводит на экран только строки, не содержащие двузначных чисел.

### **Вариант 12**

Написать программу, которая считывает текст из файла и выводит на экран только предложения, начинающиеся с тире, перед которым могут находиться только пробельные символы.

### **Вариант 13**

Написать программу, которая считывает английский текст из файла и выводит его на экран, заменив каждую первую букву слов, начинающихся с гласной буквы, на прописную.

### **Вариант 14**

Написать программу, которая считывает текст из файла и выводит его на экран, заменив цифры от 0 до 9 на слова “ноль”, “один”, ... “девять”, начиная каждое предложение с новой строки.

### **Вариант 15**

Написать программу, которая считывает текст из файла, находит самое длинное слово и определяет, сколько раз оно встретилось в тексте.

### **Вариант 16**

Написать программу, которая считывает текст из файла и выводит на экран

сначала вопросительные, а затем восклицательные предложения.

#### **Вариант 17**

Написать программу, которая считывает текст из файла и выводит его на экран, после каждого предложения добавляя, сколько раз встретилось в нем введенное с клавиатуры слово.

#### **Вариант 18**

Написать программу, которая считывает текст из файла и выводит на экран все его предложения в обратном порядке.

#### **Вариант 19**

Написать программу, которая считывает текст из файла и выводит на экран сначала предложения, начинающиеся с однобуквенных слов, а затем все остальные.

#### **Вариант 20**

Написать программу, которая считывает текст из файла и вычисляет количество открытых и закрытых скобок. Дописать вычисленные значения в конец каждой строки. Результаты записать в новый файл.

#### **Вариант 21**

Написать программу, которая считывает текст из файла и выводит на экран предложения, содержащие максимальное количество знаков пунктуации.

#### **Вариант 22**

Даны два текстовых файла. Удалить из этих файлов строки, которые имеют одинаковые номера, но сами не являются одинаковыми. Результаты записать в новый файл.

#### **Вариант 23**

В каждой строке текстового файла найти наиболее длинную последовательность цифр. Значение ее длины преобразовать в строку, которую записать в начале строки файла. Результаты записать в новый файл.

#### **Вариант 24**

Даны два текстовых файла. Создать третий файл из символов, которые записаны в позициях с одинаковыми номерами, но различаются между собой.

#### **Вариант 25**

Дан текстовый файл с программой. Исключить комментарии в тексте Pascal-программы.

#### **Вариант 26**

Написать программу, которая подсчитывает количество пустых строк в текстовом файле.

#### **Вариант 27**

Написать программу, которая находит максимальную длину строки текстового файла и печатает эту строку.

#### **Вариант 28**

Пусть текстовый файл разбит на непустые строки. Написать программу для подсчета числа строк, которые начинаются и оканчиваются одной и той же литерой.

#### **Вариант 29**

Пусть текстовый файл разбит на непустые строки. Написать программу для подсчета числа строк, которые состоят из одинаковых литер.

### Вариант 30

Написать программу, переписывающую содержимое текстового файла `t2` в текстовый файл `t1` (с сохранением деления на строки).

#### 6.4 Порядок выполнения работы

6.4.1 Выбрать способ представления исходных данных задачи и результатов.

6.4.2 Разработать алгоритм решения задачи, разбив его при необходимости на отдельные процедуры или функции.

6.4.3 Разработать программу на языке **Pascal**.

6.4.4 Разработать тестовые примеры, следуя указаниям, изложенным в разделе 6.2.

6.4.5 Выполнить отладку программы

6.4.6 Исследовать работоспособность программы для различных режимов ее использования.

6.4.7 Используя окно просмотра **Watch**, выяснить значение файловой переменной до и после выполнения процедуры **Assign**, а также до и после выполнения процедур **Reset** и **Rewrite**.

#### 6.5 Содержание отчета

Цель работы, вариант задания, структурная схема алгоритма решения задачи с описанием, текст программы с комментариями, тестовые примеры, результаты вычислений, значения файловой переменной на разных этапах выполнения программы, выводы.

#### 6.6 Контрольные вопросы

6.6.1 Какие типы данных используются в качестве базовых при построении множественных типов? Приведите пример описания множественного типа

6.6.2 Какие операции определены над множественными типами и их приоритет?

6.6.3 Приведите примеры выполнения операций над множественными типами.

6.6.4 Напишите программу создания множества из букв латинского алфавита.

6.6.5 Чем отличается текущая длина строки от ее общей длины?

6.6.6 Как хранятся в памяти строки типа **string**?

6.6.7 Как инициализируется строка при ее объявлении?

6.6.8 Напишите примеры ввода и вывода строк.

6.6.9 Какие библиотечные функции имеются для работы со строками в языке **TurboPascal**?

6.6.10 Объясните, что означают следующие термины: логический и физический файл, текстовый и бинарный файл, последовательный и прямой метод доступа.

6.6.11 Напишите примеры открытия и закрытия файлов.

6.6.12 Как выполняется запись данных в файл последовательного доступа, чтение из файла? Как обнаружить конец файла? Как обнаружить конец строки текстового файла?

6.6.13 Как связать файловую переменную с файлом расположенным на внешнем устройстве?

## 7 ЛАБОРАТОРНАЯ РАБОТА №7

### “ПРОГРАММИРОВАНИЕ ОПЕРАЦИЙ НАД ЗАПИСЯМИ И ТИПИЗИРОВАННЫМИ ФАЙЛАМИ”

#### 7.1 Цель работы

Исследование способов обработки логически связанных данных различных типов. Создание программ, использующих записи и типизированные файлы.

#### 7.2 Краткие теоретические сведения

##### 7.2.1 Записи

*Запись* (комбинированный тип) – это структурированный тип данных, содержащий из нескольких компонент (полей). В отличие от массивов компоненты записи могут иметь разные типы, и доступ к ним осуществляется не по индексу, а по имени поля. При определении записи задаются имя и тип каждого поля. Описание записи начинается служебным словом **record** и заканчивается словом **end**. В качестве примера рассмотрим запись, содержащую анкетные данные студента:

```
Type Student=record
    First_name: string[15];
    Last_name: string[15];
    birthday: record
        day: 1..31;
        month: 1..12;
        year: integer
    end;
end;
Var S: Student;
    Grappa: array[1..30] of Student;
```

К каждому компоненту записи можно обратиться, используя имя переменной и имя поля, разделенные точкой. Такие имена в программах называются *составными именами*. Например,

```
S.First_name:= 'ИВАНОВ';
```

Поле с именем **birthday** – тоже запись и состоит из трех полей. Доступ к этим полям осуществляется добавлением через точку имени соответствующего поля:

```
S.birthday.day:=15;
S.birthday.month:=4;
S.birthday.year:=1978;
```

Приведенные операторы присваивания можно записать короче, если использовать оператор присоединения:

```
With < имя переменной типа запись> Do  Begin <операторы> End
```

В этом случае в операторах вместо составных имен указываются только имена полей:

```

With S.birthday Do
  Begin day:=15;
        month:=4;
        year:=1978
End;

```

Записи могут входить в качестве компонентов в другие переменные. Например, переменная **Gruppa** – это массив, состоящий из 30 записей. Оператор присваивания нового значения полю **month** пятого элемента массива имеет вид **Gruppa[5].birthday.month:=12;**

Легко видеть, что к каждой записи группы доступ осуществляется при помощи индекса.

Приведенный ниже фрагмент программы подсчитывает число студентов, родившихся в 1978 году:

```

K:=0;
For i:=1 To 30 Do
  With Gruppa[i].birthday Do
    If year=1978 then K:=K+1;
  Writeln(' число родившихся в 1978 году = ', K:2);

```

При определении записи в неё можно включать вариантную часть. Например, опишем запись **person**, предназначенную для хранения данных либо о преподавателе, либо о студенте. Такая запись будет содержать фиксированную часть, в которой будет указана фамилия персоны и адрес, а также вариантную часть, в которой для преподавателя будет указываться должность (**post**) и количество трудов, а для студента средний балл и группа.

```

Type person=record
  FIO: string;
  Address: string;
  Case who: (teacher, student) of
    teacher: (post: string; number: integer);
    student: ( mark: real; gruppa: string)
  End;

```

Вариантная часть записи начинается словом **case**. После ее окончания в записи не могут появляться никакие другие поля, поэтому слово **Case** не закрывается отдельным служебным словом **End**. За словом **case** следует *поле признака* с указанием его типа – это поле **who**. Если это поле примет значение **teacher**, то рассматриваемая запись будет включать поля, соответствующие первому варианту, иначе – второму. Тип поля признака должен быть перечислимым.

Для переменной **P** типа **person** можно записать следующие операторы присваивания:

```

P.FIO:= 'СТОЛЯРОВ';
P.address:= 'ул. Тенистая д.5 кв. 7';
P.who:= teacher;
P.post:= 'docent';
P.number:= 52;

```

Задав значение поля признака **who**, следует присваивать новые значения только полям того варианта, которые помечены соответствующей константой выбора.

### 7.2.2 Типизированные и нетипизированные файлы

Бинарные файлы в языке Turbo Pascal подразделяются на *типизированные* и *нетипизированные*. Бинарные файлы допускают прямой доступ к элементам файла, то есть в соответствии с их номерами. Описание типизированного файла имеет вид:

```
Var Typed_file : file of <тип>;
```

Здесь <тип> может быть любым типом за исключением файлового. Элементами типизированного файла являются значения указанного типа.

При работе с типизированными файлами используются процедуры **Assign**, **Reset**, **Rewrite**, **Read** и **Write**. Текстовый файл, открытый процедурой **Reset**, доступен только для чтения, а типизированный файл доступен как для чтения, так и для записи. Особенностью процедур **Read** и **Write**, применяемых к типизированным файлам является то, что их параметры должны принадлежать заданному типу. Например:

```
Var F: file of integer;
    X: integer;
    ...
    Read(F,X);
```

В языке Turbo Pascal имеются специальные процедуры и функции для работы с типизированными файлами. Функция **FilePos(F)**, где **F** — типизированная файловая переменная, возвращает номер текущего элемента файла. Функция **FileSize(F)** возвращает количество элементов файла. Процедура **Seek(F, N)** перемещает указатель текущего элемента в позицию **N**. Применяя последовательно процедуры **Seek(F, N)** и **Read(F, X)**, можно из файла считывать элементы в произвольном порядке.

Чтобы эффективно выполнять операции ввода-вывода для внешних файлов, в языке Pascal, целесообразно использовать нетипизированные файлы, так как при работе с ними можно применять быстрые дисковые операции низкого уровня. Нетипизированные файлы рассматриваются как последовательность байтов. Описание нетипизированной файловой переменной имеет вид

```
Var Untyped_file : file;
```

Такая файловая переменная связывается с внешним файлом обычным образом. В числе параметров процедур **Reset** и **Rewrite** для нетипизированных файлов, кроме файловой переменной, имеется необязательный второй параметр типа **Word**, например:

```
Reset (untyped_file, n);
```

Дополнительный параметр **n** описывает размер индивидуальной записи в файле в байтах. Если параметр **n** отсутствует, его значение по умолчанию принимается равным 128.

При работе с нетипизированными файлами для чтения и записи значений используются процедуры **BlockRead** и **BlockWrite**. Эти процедуры требуют организации в памяти буфера (область памяти, предназначенная для временного хранения данных). В качестве буфера обычно используются массивы.

### 7.2.3 Пример программы обработки записей типизированного файла

Имеются записи о сотрудниках, содержащие фамилию и инициалы, год рождения и оклад сотрудника. Требуется написать программу с использованием процедур и функций, которая:

- вводит записи о сотрудниках с клавиатуры в типизированный файл и упорядочивает его по размеру оклада;
- осуществляет поиск сотрудников в файле по фамилии и вычисляет средний оклад этих сотрудников;
- отображает содержимое типизированного файла.

#### Исходные данные и результаты.

Сведения об одном сотруднике будем представлять в виде записи. При этом фамилию и инициалы представим строкой из пятнадцати символов, год рождения – целым типом, оклад – вещественным типом. Поскольку количество записей о сотрудниках не оговорено, будем хранить все сведения в типизированном файле, а не массиве.

В результате работы программы требуется вывести на экран требуемые элементы файла. Так как результаты представляют собой выборку из исходных данных, дополнительная память для них не отводится. Кроме того, необходимо подсчитать средний оклад для найденных сотрудников. Для этого введем переменную вещественного типа.

#### Алгоритм решения задачи.

1. Ввести с клавиатуры сведения о сотрудниках в типизированный файл. При этом будем прекращать ввод, если пользователь введет символ '\*’.
2. Выполнить сортировку файла по окладу, используя возможность прямого доступа к элементам типизированного файла. В качестве метода сортировки будем использовать метод прямого обмена (пузырьковую сортировку);
3. Обеспечить вывод сведений о сотруднике:
  - а) ввести с клавиатуры фамилию;
  - б) выполнить поиск сотрудника в файле по фамилии;
  - с) увеличить суммарный оклад и счетчик количества сотрудников;
  - д) вывести сведения о сотруднике или сообщение об его отсутствии;
  - е) вывести средний оклад.
4. Отобразить содержимое типизированного файла.

Каждый из указанных пунктов алгоритма представим в форме процедуры, которые можно будет вызывать из основной программы в произвольном порядке. При этом контроль над порядком вызова процедур полностью возлагается на пользователя.

Программа и тестовые примеры

Program Demo\_TypedFiles;

Uses Crt; {подключить модуль управления клавиатуры и монитора}

Type person=record {запись о сотруднике}

FIO:string[15]; {фамилия и инициалы}

Year:integer; {год рождения}

Salary:real; {оклад}

```

End;
Var f: file of person;      {файл с записями о сотрудниках}
    Elem1,Elem2: person; {два элемента файла типа запись}
{=====создание файла=====}
Procedure Create_file;
Begin
{Установить указатель в позицию, определяемую размером файла. Если
создается новый файл, то в начало. Если записи добавляются в суще-
ствующий файл, то в конец}
Seek(f,FileSize(f));
    Writeln('Вводите сведения о сотрудниках. ');
    Writeln('Для выхода вместо фамилии напечатайте символ *');
    While True Do
    Begin
        Write('Введите фамилию и инициалы (обязательно)');
        Readln(Elem1.FIO);
        if Elem1.FIO='*' Then Break; {Выход, если введена *}
        Write('Введите год рождения'); Readln(Elem1.Year);
        Writeln('Введите оклад'); Readln(Elem1.Salary);
        Write(f,Elem1)
    End
End;
{=====сортировка файла=====}
Procedure Sort_file;      {Метод пузырька}
Var i, j: integer;
Begin
Seek(f,0); {Указатель файла установить в начало}
For i:=filesize(f)-1 downto 1 do {цикл, задающий число проходов}
For j:=0 to i-1 do {сравнивать элементы от 0-го до (i-1)-го}
    Begin Seek(f,j); {Указатель установить на j-ый элемент}
        Read(f,Elem1,Elem2); {Прочитать две записи}
        If Elem1.Salary>Elem2.Salary then {Сравнить оклады}
            Begin Seek(f,j); {Вернуть указатель в j-ую позицию}
                Write(f,Elem2,Elem1); {Поменять местами записи}
            End
    End
End;
End;
{=====вывод содержимого файла=====}
Procedure Print_file;
Begin Seek(f,0); {Указатель файла установить в начало}
    Writeln('Фамилия':15, 'Год':5, 'Зарплата':10);
    While not eof(f) do {Просмотр всех записей до конца файла}
    Begin
        Read(f,Elem1);

```



```

        Writeln(Elem1.FIO:15, Elem1.Year:5, Elem1.Salary:10:2);
    End
    Readkey; {Ожидание нажатия клавиши Enter}
End;
{====поиск сотрудников и вычисление ср. оклада=====}
Procedure Search_persons;
Var   Found: boolean; {признак успеха поиска}
      S: string[15];   {строка с фамилией}
      N_persons: integer; {количество найденных сотрудников}
      SummaSalary: Real; {суммарный оклад сотрудников}
Begin N_persons:=0; SummaSalary:=0;
While True Do
Begin Writeln('Введите фамилию или *');
      Readln(s);
      If s='*' Then Break; {Прерывание поиска, если введена *}
      Found:=False;
      Seek(f,0);           {Указатель файла установить в начало}
{Пока не конец файла и пока не найден сотрудник с введенной
фамилией продолжать поиск}
      While not eof(f) and not Found Do
      Begin
      Read(f,Elem1); {Прочитать очередную запись из файла}
      If (Pos(s,Elem1.FIO)<>0) and {Есть ли в FIO фамилия?}
         (Elem1.FIO[Length(s)+1]=' ') {Есть ли после нее пробел?}
      Then Begin
      {Вывод сведений о найденном сотруднике, увеличение счетчика
найденных сотрудников и вычисление суммарного оклада}
        Writeln(Elem1.FIO:15, Elem1.Year:5, Elem1.Salary:10:2);
        N_persons:=N_persons+1;
        SummaSalary:=SummaSalary+Elem1.Salary;
        Found:=True; {Сотрудник найден, прервать цикл}
      End;
      End;
      {Если не найден}
      If not Found Then Writeln('Такого сотрудника нет');
      End;
      If N_persons<>0 Then
      Writeln('Средний оклад=', SummaSalary/N_persons);
      Readkey; {Ожидание нажатия клавиши Enter для фиксации окна
просмотра}
      End;
{=====основная программа=====}
Begin
Assign(f,'person.dat'); {Установить связь логич. и физич. файлов}
{$I-}                   {отключить проверку ошибок ввода-вывода}

```

```

Reset(f); {открыть файл для чтения и добавления записей}
{$I+}     {включить проверку ошибок ввода-вывода}
IfI OResult=0 Then {Если файл успешно открыт, то}
    writeln('Добавление записей в существующий файл')
else {Если файла нет, то создать новый файл и открыть его}
begin Rewrite(f); {Создание и открытие файла для чтения/записи}
    writeln('Запись в новый файл');
end;
readkey; {Ожидание нажатия клавиши Enter }
{бесконечный цикл, обеспечивающий вывод на экран пунктов меню}
While True Do
Begin
ClrScr; { очистка экрана }
Writeln('1-Создание файла');
Writeln('2-Сортировка файла');
Writeln('3-Вывод содержимого файла');
Writeln('4-Поиск сотрудников и вычисление ср. оклада');
Writeln('5-Выход');
Writeln('-----');
Writeln('Введите номер пункта меню');
Readln(Number);
Case Number Of {Вызов необходимой процедуры по номеру}
    1:Create_file; {Вызов процедуры создания файла}
    2:Sort_file; {Вызов процедуры сортировки файла}
    3:Print_file; {Вызов процедуры просмотра файла}
    5:Search_persons; {Вызов процедуры поиска сотрудника}
    4:Exit {Встроенная процедура выхода}
End
End
End.

```

В тексте основной программы осуществляется открытие файла либо с помощью процедуры **Reset**, если файл существует, либо с помощью процедуры **Rewrite**, если файл **'person.dat'** на диске не обнаружен. Это позволяет один раз ввести исходные данные в файл, а затем его многократно использовать. Если потребуется начать работу с новым файлом, то старый файл следует удалить с диска средствами операционной системы.

Все необходимые действия с файлом выполняются с помощью процедур, вызываемых из пунктов меню, отображаемых на экране. При заполнении файла требуется обязательно вводить инициалы, которые должны отделяться от фамилии пробелом. Фамилия должна начинаться с первой позиции каждой строки. В дальнейшем это свойство используется в процедуре поиска сотрудников.

Рассмотрим подробнее процедуру **Search\_persons**. Здесь цикл поиска сотрудников по фамилии организован как бесконечный цикл, выход из которого выполняется при вводе вместо фамилии символа **"\*"**. В процедуре введена переменная **Found**

для того, чтобы после окончания цикла поиска сотрудника в файле было известно, завершился ли он успешно. При этом проверка совпадения фамилии сотрудника производится в два этапа. Сначала с помощью функции `Pos` проверяется, входит ли подстрока, представляющая фамилию, в строку, содержащую фамилию и инициалы. Если входит, то функция `Pos` вернет ненулевое значение. После этого проверяется, есть ли непосредственно после фамилии пробел (если пробела нет, то искомая фамилия является частью другой, и эта запись нам не подходит).

При отладке программы возможны проблемы при поиске фамилий, заданных русскими буквами. Это обусловлено разной кодировкой букв русского алфавита в операционных системах MSDOS (кодировка ASCII) и Windows (кодировка ANSI). Для того чтобы при работе программы нормально отображались русские сообщения, рекомендуется исходный текст программы подготовить с помощью редактора Far, выбрав кодировку DOS нажатием клавиши F8. Для ввода фамилий при работе программы на русском языке потребуется установить на компьютер драйвер клавиатуры, допускающий необходимые переключения в режиме DOS.

### 7.3 Варианты заданий

#### Вариант 1

Описать структуру с именем `STUDENT`, содержащую следующие поля:

- фамилия и инициалы;
- номер группы;
- успеваемость (массив из пяти элементов).

Написать программу, выполняющую следующие действия с помощью процедур или функций:

- ввод с клавиатуры данных в файл, состоящий из структур типа `STUDENT`; записи должны быть упорядочены по возрастанию номера группы;
- чтение данных из этого файла, корректировку данных в файле по номеру записи;
- вывод на дисплей фамилий и номеров групп для всех студентов, включенных в массив, если средний балл студента больше 4.0; если таких студентов нет, вывести соответствующее сообщение.

#### Вариант 2

Описать структуру с именем `STUDENT`, содержащую следующие поля:

- фамилия и инициалы;
- номер группы;
- успеваемость (массив из пяти элементов).

Написать программу, выполняющую следующие действия с помощью процедур или функций:

- ввод с клавиатуры данных в файл, состоящий из структур типа `STUDENT`; записи должны быть упорядочены по возрастанию среднего балла;
- чтение данных из этого файла;
- вывод на дисплей фамилий и номеров групп для всех студентов, имеющих оценки 4 и 5; если таких студентов нет, вывести соответствующее сообщение.

#### Вариант 3

Описать структуру с именем `STUDENT`, содержащую следующие поля:

- фамилия и инициалы;
- номер группы;
- успеваемость (массив из пяти элементов).

Написать программу, выполняющую следующие действия с помощью процедур или функций:

- ввод с клавиатуры данных в файл, состоящий из структур типа STUDENT; записи должны быть упорядочены по алфавиту;
- чтение данных из этого файла;
- вывод на дисплей фамилий и номеров групп для всех студентов, имеющих хотя бы одну оценку 2; если таких студентов нет, вывести соответствующее сообщение.

#### **Вариант 4**

Описать структуру с именем AEROFLOT, содержащую следующие поля:

- название пункта назначения рейса;
- номер рейса;
- тип самолета.

Написать программу, выполняющую следующие действия с помощью процедур или функций:

- ввод с клавиатуры данных в файл, состоящий из элементов типа AEROFLOT; записи должны быть упорядочены по возрастанию номера рейса;
- чтение данных из этого файла;
- вывод на экран номеров рейсов и типов самолетов, вылетающих в пункт назначения, название которого совпало с названием, введенным с клавиатуры; если таких рейсов нет, выдать на дисплей соответствующее сообщение

#### **Вариант 5**

Описать структуру с именем AEROFLOT, содержащую следующие поля:

- название пункта назначения рейса;
- номер рейса;
- тип самолета.

Написать программу, выполняющую следующие действия с помощью процедур или функций:

- ввод с клавиатуры данных в файл, состоящий из элементов типа AEROFLOT; записи должны быть размещены в алфавитном порядке по названиям пунктов назначения;
- чтение данных из этого файла;
- вывод на экран пунктов назначения и номеров рейсов, обслуживаемых самолетом, тип которого введен с клавиатуры; если таких рейсов нет, выдать на дисплей соответствующее сообщение.

#### **Вариант 6**

Описать структуру с именем WORKER, содержащую следующие поля:

- фамилия и инициалы работника;
- название занимаемой должности;
- год поступления на работу.

Написать программу, выполняющую следующие действия с помощью процедур или функций:

- ввод с клавиатуры данных в файл, состоящий из структур типа WORKER; записи должны быть размещены по алфавиту;
- чтение данных из этого файла;
- вывод на дисплей фамилий работников, чей стаж работы в организации превышает значение, введенное с клавиатуры; если таких работников нет, вывести на дисплей соответствующее сообщение.

### **Вариант 7**

Описать структуру с именем TRAIN, содержащую следующие поля:

- название пункта назначения;
- номер поезда;
- время отправления.

Написать программу, выполняющую следующие действия с помощью процедур или функций:

- ввод с клавиатуры данных в файл, состоящий из элементов типа TRAIN; записи должны быть размещены в алфавитном порядке по названиям пунктов назначения;
- чтение данных из этого файла;
- вывод на экран информации о поездах, отправляющихся после введенного с клавиатуры времени; если таких поездов нет, выдать на дисплей соответствующее сообщение.

### **Вариант 8**

Описать структуру с именем TRAIN, содержащую следующие поля:

- название пункта назначения;
- номер поезда;
- время отправления.

Написать программу, выполняющую следующие действия с помощью процедур или функций:

- ввод с клавиатуры данных в файл, состоящий из элементов типа TRAIN; записи должны быть упорядочены по времени отправления поезда;
- чтение данных из этого файла;
- вывод на экран информации о поездах, направляющихся в пункт, название которого введено с клавиатуры; если таких поездов нет, выдать на дисплей соответствующее сообщение.

### **Вариант 9**

Описать структуру с именем TRAIN, содержащую следующие поля:

- название пункта назначения;
- номер поезда;
- время отправления.

Написать программу, выполняющую следующие действия с помощью процедур или функций:

- ввод с клавиатуры данных в файл, состоящий из элементов типа TRAIN; записи должны быть упорядочены по номерам поездов;
- чтение данных из этого файла;
- вывод на экран информации о поезде, номер которого введен с клавиатуры; если таких поездов нет, выдать на дисплей соответствующее сообщение.

**Вариант 10**

Описать структуру с именем MARSH, содержащую следующие поля:

- название начального пункта маршрута;
- название конечного пункта маршрута;
- номер маршрута.

Написать программу, выполняющую следующие действия с помощью процедур или функций:

- ввод с клавиатуры данных в файл, состоящий из элементов типа MARSH; записи должны быть упорядочены по номерам маршрутов;
- чтение данных из этого файла;
- вывод на экран информации о маршруте, номер которого введен с клавиатуры; если таких маршрутов нет, выдать на дисплей соответствующее сообщение.

**Вариант 11**

Описать структуру с именем MARSH, содержащую следующие поля:

- название начального пункта маршрута;
- название конечного пункта маршрута;
- номер маршрута.

Написать программу, выполняющую следующие действия с помощью процедур или функций:

- ввод с клавиатуры данных в файл, состоящий из элементов типа MARSH; записи должны быть упорядочены по номерам маршрутов;
- чтение данных из этого файла;
- вывод на экран информации о маршрутах, которые начинаются или оканчиваются в пункте, название которого введено с клавиатуры; если таких маршрутов нет, выдать на дисплей соответствующее сообщение.

**Вариант 12**

Описать структуру с именем NOTE, содержащую следующие поля:

- фамилия, имя;
- номер телефона;
- дата рождения (массив из трех чисел).

Написать программу, выполняющую следующие действия с помощью процедур или функций:

- ввод с клавиатуры данных в файл, состоящий из элементов типа NOTE; записи должны быть упорядочены по датам рождения;
- чтение данных из этого файла;
- вывод на экран информации о человеке, номер телефона которого введен с клавиатуры; если такого нет, выдать на дисплей соответствующее сообщение.

**Вариант 13**

Описать структуру с именем NOTE, содержащую следующие поля:

- фамилия, имя;
- номер телефона;
- дата рождения (массив из трех чисел).

Написать программу, выполняющую следующие действия с помощью процедур или функций:

- ввод с клавиатуры данных в файл, состоящий из элементов типа NOTE; записи

должны быть размещены по алфавиту;

- чтение данных из этого файла;
- вывод на экран информации о людях, чьи дни рождения приходятся на месяц, значение которого введено с клавиатуры; если таких нет, выдать на дисплей соответствующее сообщение.

#### **Вариант 14**

Описать структуру с именем NOTE, содержащую следующие поля:

- фамилия, имя;
- номер телефона;
- дата рождения (массив из трех чисел).

Написать программу, выполняющую следующие действия с помощью процедур или функций:

- ввод с клавиатуры данных в файл, состоящий из элементов типа NOTE; записи должны быть упорядочены по трем первым цифрам номера телефона;
- чтение данных из этого файла;
- вывод на экран информации о человеке, чья фамилия введена с клавиатуры; если такого нет, выдать на дисплей соответствующее сообщение.

#### **Вариант 15**

Описать структуру с именем ZNAK, содержащую следующие поля:

- фамилия, имя;
- знак Зодиака;
- дата рождения (массив из трех чисел).

Написать программу, выполняющую следующие действия с помощью процедур или функций:

- ввод с клавиатуры данных в файл, состоящий из элементов типа ZNAK; записи должны быть упорядочены по датам рождения;
- чтение данных из этого файла;
- вывод на экран информации о человеке, чья фамилия введена с клавиатуры; если такого нет, выдать на дисплей соответствующее сообщение.

#### **Вариант 16**

Описать структуру с именем ZNAK, содержащую следующие поля:

- фамилия, имя;
- знак Зодиака;
- дата рождения (массив из трех чисел).

Написать программу, выполняющую следующие действия с помощью процедур или функций:

- ввод с клавиатуры данных в файл, состоящий из элементов типа ZNAK; записи должны быть упорядочены по датам рождения;
- чтение данных из этого файла;
- вывод на экран информации о людях, родившихся под знаком, название которого введено с клавиатуры; если таких нет, выдать на дисплей соответствующее сообщение.

#### **Вариант 17**

Описать структуру с именем ZNAK, содержащую следующие поля:

- фамилия, имя;

- знак Зодиака;
- дата рождения (массив из трех чисел).

Написать программу, выполняющую следующие действия с помощью процедур или функций:

- ввод с клавиатуры данных в файл, состоящий из элементов типа ZNAK (записи должны быть упорядочены по знакам Зодиака);
- чтение данных из этого файла;
- вывод на экран информации о людях, родившихся в месяц, значение которого введено с клавиатуры; если таких нет, выдать на дисплей соответствующее сообщение.

### **Вариант 18**

Описать структуру с именем PRICE, содержащую следующие поля:

- название товара;
- название магазина, в котором продается товар;
- стоимость товара в грн.

Написать программу, выполняющую следующие действия с помощью процедур или функций:

- ввод с клавиатуры данных в файл, состоящий из элементов типа PRICE; записи должны быть размещены в алфавитном порядке по названиям товаров;
- чтение данных из этого файла;
- вывод на экран информации о товаре, название которого введено с клавиатуры; если таких товаров нет, выдать на дисплей соответствующее сообщение

### **Вариант 19**

Описать структуру с именем PRICE, содержащую следующие поля:

- название товара;
- название магазина, в котором продается товар;
- стоимость товара в грн.

Написать программу, выполняющую следующие действия с помощью процедур или функций:

- ввод с клавиатуры данных в файл, состоящий из элементов типа PRICE; записи должны быть размещены в алфавитном порядке по названиям магазинов;
- чтение данных из этого файла;
- вывод на экран информации о товарах, продающихся в магазине, название которого введено с клавиатуры; если такого магазина нет, выдать на дисплей соответствующее сообщение.

### **Вариант 20**

Описать структуру с именем ORDER, содержащую следующие поля:

- расчетный счет плательщика;
- расчетный счет получателя;
- перечисляемая сумма в грн.

Написать программу, выполняющую следующие действия с помощью процедур или функций:

- ввод с клавиатуры данных в файл, состоящий из элементов типа ORDER; записи должны быть размещены в алфавитном порядке по расчетным счетам плательщиков;



- чтение данных из этого файла;
- вывод на экран информации о сумме, снятой с расчетного счета плательщика, введенного с клавиатуры; если такого расчетного счета нет, выдать на дисплей соответствующее сообщение.

### **Вариант 21**

Описать структуру с именем STUDENT, содержащую следующие поля:

- номер;
- фамилия и имя;
- год рождения;
- год поступления в университет;
- структура OCENKI, содержащая четыре поля: физика, математика, программирование, история;

Написать программу, выполняющую следующие действия с помощью процедур или функций:

- ввод с клавиатуры данных в файл, состоящий из структур типа STUDENT; записи должны быть упорядочены по алфавиту;
- чтение данных из этого файла;
- вывод на дисплей анкетных данных студентов отличников; если таких студентов нет, вывести соответствующее сообщение.

### **Вариант 22**

Для структуры, указанной в варианте 21, написать программу, выполняющую следующие действия с помощью процедур или функций:

- ввод с клавиатуры данных в файл, состоящий из структур типа STUDENT; записи должны быть упорядочены номеру;
- чтение данных из этого файла;
- вывод на дисплей анкетных данных студентов, получивших одну оценку 3; если таких студентов нет, вывести соответствующее сообщение.

### **Вариант 23**

Для структуры, указанной в варианте 21, написать программу, выполняющую следующие действия с помощью процедур или функций:

- ввод с клавиатуры данных в файл, состоящий из структур типа STUDENT; записи должны быть упорядочены по фамилиям;
- чтение данных из этого файла;
- вывод на дисплей анкетных данных студентов, получивших все двойки; если таких студентов нет, вывести соответствующее сообщение.

### **Вариант 24**

Для структуры, указанной в варианте 21, написать программу, выполняющую следующие действия с помощью процедур или функций:

- ввод с клавиатуры данных в файл, состоящий из структур типа STUDENT; записи должны быть упорядочены по номеру;
- чтение данных из этого файла;
- вывод на дисплей анкетных данных студентов, получивших все пятерки; если таких студентов нет, вывести соответствующее сообщение.

### **Вариант 25**

Для структуры, указанной в варианте 21, написать программу, выполняющую

следующие действия с помощью процедур или функций:

- ввод с клавиатуры данных в файл, состоящий из структур типа STUDENT; записи должны быть упорядочены по алфавиту;
- чтение данных из этого файла;
- вывод на дисплей анкетных данных студентов, получивших одну оценку 4, а все остальные – 5; если таких студентов нет, вывести соответствующее сообщение.

#### **Вариант 26**

Для структуры, указанной в варианте 21, написать программу, выполняющую следующие действия с помощью процедур или функций:

- ввод с клавиатуры данных в файл, состоящий из структур типа STUDENT; записи должны быть упорядочены по алфавиту;
- чтение данных из этого файла;
- вывод на дисплей фамилий студентов, которые начинаются с литеры 'А', и их оценки; если таких студентов нет, вывести соответствующее сообщение.

#### **Вариант 27**

Для структуры, указанной в варианте 21, написать программу, выполняющую следующие действия с помощью процедур или функций:

- ввод с клавиатуры данных в файл, состоящий из структур типа STUDENT; записи должны быть упорядочены по номеру;
- чтение данных из этого файла;
- вывод на дисплей фамилий студентов, которые начинаются с литеры 'Б', и год их рождения; если таких студентов нет, вывести соответствующее сообщение.

#### **Вариант 28**

Для структуры, указанной в варианте 21, написать программу, выполняющую следующие действия с помощью процедур или функций:

- ввод с клавиатуры данных в файл, состоящий из структур типа STUDENT; записи должны быть упорядочены по фамилиям;
- чтение данных из этого файла;
- вывод на дисплей фамилий студентов, которые начинаются с литеры 'Б' или 'Г', и год их поступления; если таких студентов нет, вывести соответствующее сообщение.

#### **Вариант 29**

Для структуры, указанной в варианте 21, написать программу, выполняющую следующие действия с помощью процедур или функций:

- ввод с клавиатуры данных в файл, состоящий из структур типа STUDENT; записи должны быть упорядочены по фамилиям;
- чтение данных из этого файла;
- вывод на дисплей фамилий студентов, имеющих средний балл выше среднего балла группы; если таких студентов нет, вывести соответствующее сообщение.

#### **Вариант 30**

Для структуры, указанной в варианте 21, написать программу, выполняющую следующие действия с помощью процедур или функций:

- ввод с клавиатуры данных в файл, состоящий из структур типа STUDENT; записи должны быть упорядочены по номеру;

- чтение данных из этого файла;
- вывод на дисплей фамилий и года рождения студентов, не получивших ни одной тройки; если таких студентов нет, вывести соответствующее сообщение.

#### **7.4 Порядок выполнения работы**

7.4.1 Выбрать способ представления исходных данных задачи и результатов.

7.4.2 Разработать алгоритм решения задачи, разбив его на отдельные процедуры и функции.

7.4.3 Разработать программу на языке **Pascal**.

7.4.4 Разработать тестовые примеры, следуя указаниям раздела 7.3.

7.4.5 Выполнить отладку программы.

7.4.6 Исследовать работоспособность программы в различных режимах ее использования.

#### **7.5 Содержание отчета**

Цель работы, вариант задания, структурная схема алгоритма решения задачи с описанием, текст программы с комментариями, тестовые примеры, результаты вычислений, выводы.

#### **7.6 Контрольные вопросы**

7.6.1 Что понимают под комбинированным типом данных?

7.6.2 Приведите пример описания комбинированного типа.

7.6.3 Как осуществляется обращение к полям комбинированного типа?

7.6.4 Для чего применяют оператор присоединения?

7.6.5 Приведите пример использования оператора присоединения.

7.6.6 Приведите пример описания вариантных записей.

7.6.7 Как обратиться к полям вариантов?

7.6.8 Как выполняется описание пустых полей вариантов?

7.6.9 Чем отличаются последовательный доступ к компонентам файла и прямой доступ?

7.6.10 В чем состоит различие между типизированными и нетипизированными файлами?

7.6.11 Назовите и объясните специальные функции и процедуры для работы с типизированными файлами.

7.6.12 Как записать и считать значения из нетипизированного файла?

## 8 ЛАБОРАТОРНАЯ РАБОТА №8 “ПРОГРАММИРОВАНИЕ ЛИНЕЙНЫХ СПИСКОВ”

### 8.1 Цель работы

Исследование списковых структур данных и приобретение навыков разработки и отладки программ, использующих динамическую память. Исследование особенностей использования переменных ссылочного типа.

### 8.2 Краткие теоретические сведения

#### 8.2.1 Ссылочный тип

В предыдущих лабораторных работах рассматривались *статические переменные*. Память под такие переменные отводится заранее на этапе компиляции программы. Часть возникает необходимость в использовании переменных, которые создаются и уничтожаются в процессе выполнения программы. Такие переменные называют *динамическими*, а память, которая для них выделяется – *динамической памятью*. Для работы с динамической памятью используют *переменные ссылочного типа (указатели)*. Задание ссылочного типа выполняется следующим образом:

<задание ссылочного типа>::=^<имя типа>

Например:

Type

IntPtr=^Integer;

Massive=Array[1..10] of Real;

Var

P:IntPtr;

Rabmas:^Massiv;

Здесь ссылочная переменная **P** – это указатель, значением которого является адрес области памяти, в которой хранится целое значение. Ссылочная переменная **RABMAS** – указатель на область памяти, где хранится массив.

В языке Turbo Pascal различают *типизированные* и *нетипизированные* указатели. Приведенные выше указатели могут ссылаться лишь на данные соответствующего типа и называются типизированными. Нетипизированный указатель не связывается с каким-либо типом данных и описывается как переменная типа **Pointer**:

Var Ukaz:Pointer;

Такой указатель используют для ссылки на данные, тип и структура которых меняются во время выполнения программы.

После описания ссылочной переменной выделяется область динамической памяти, на которую она будет ссылаться. Для этого используют процедуру **New(<имя ссылочной переменной>)**, которая выделяет необходимый объем памяти и присваивает переменной ссылочного типа значение, соответствующее начальному адресу выделенной памяти.

Для того чтобы записать значение в выделенную память, используют переменную с указателем

<переменная с указателем>::=<ссылочная переменная>^

Например:

```
New(P); P^:=5;
```

В этом случае в область динамической памяти, на которую ссылается указатель P, записывается значение 5 (рисунок 8.1). На рисунке символом "\*" обозначено (в общем случае неизвестное программисту) значение ссылочной переменной P.

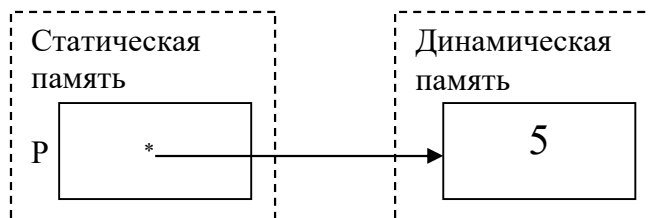


Рисунок 8.1 – Указатель и динамическая память

Для уничтожения объектов в динамической памяти применяется процедура `Dispose(<имя ссылочной переменной>)`. Данная процедура освобождает область динамической памяти, на которую ссылается параметр процедуры, после чего эта область становится доступной для дальнейшего распределения между другими динамическими переменными.

Для работы с нетипизированными указателями в языке Turbo Pascal используются процедуры `GetMem` и `FreeMem`. Процедура `GetMem` выделяет область динамической памяти заданного объема, а процедура `FreeMem` освобождает эту область памяти. Вызов этих процедур выполняется следующим образом:

```
GetMem(<имя ссылочной переменной>, <объем памяти>);
```

```
FreeMem (<имя ссылочной переменной>, <объем памяти>);
```

Здесь ссылочная переменная должна иметь тип `Pointer`, а объем памяти не должен превышать 65521 байт.

### 8.2.2 Списковые структуры данных

Из динамических структур в программах наиболее часто используют различные линейные списки, стеки, очереди, деревья. Они различаются способами связи отдельных элементов и допустимыми операциями над ними.

Линейные структуры данных предназначены для отображения линейных связей. Линейные структуры данных могут быть последовательными и списковыми. *Последовательной* называется структура, в которой логический порядок следования элементов задается их физическим порядком. Примером последовательной структуры данных является массив. *Списковой* называется такая структура данных, при которой логический порядок следования элементов задается путем отсылок, то есть каждый элемент списка, кроме последнего, содержит указатель на следующий элемент (или предыдущий). Доступ к первому элементу списка выполняется с помощью специального указателя – указателя на вершину (голову) списка.

*Односвязным линейным списком* называют список, в котором предыдущий элемент ссылается на следующий. *Двусвязный линейный список* – это список, в котором предыдущий элемент ссылается на следующий, а следующий – на предыдущий. *Односвязный циклический список* – это односвязный линейный список, в котором последний элемент ссылается на первый. *Стек* – это односвязный список, в котором

компоненты добавляются и удаляются только со стороны вершины списка. *Очередь* – это односвязный список, в котором компоненты добавляются в конец списка, а удаляются со стороны вершины списка.

Для задания списковых структур необходимо определить элемент списка в виде записи, в состав которой входит поле-указатель на элемент этого же типа. Для этого в языке Pascal разрешено определять ссылочные типы на еще не описанные типы значений:

```

Type
  Data=<тип данных элемента списка>;
  Ukaz=^Element;
  Element=Record
    D:Data;
    Next:Ukaz { поле-указатель на элемент}
  End;

```

Определив ссылочный тип *Ukaz*, можно с его помощью построить следующий связный однонаправленный список (рисунок 8.2).

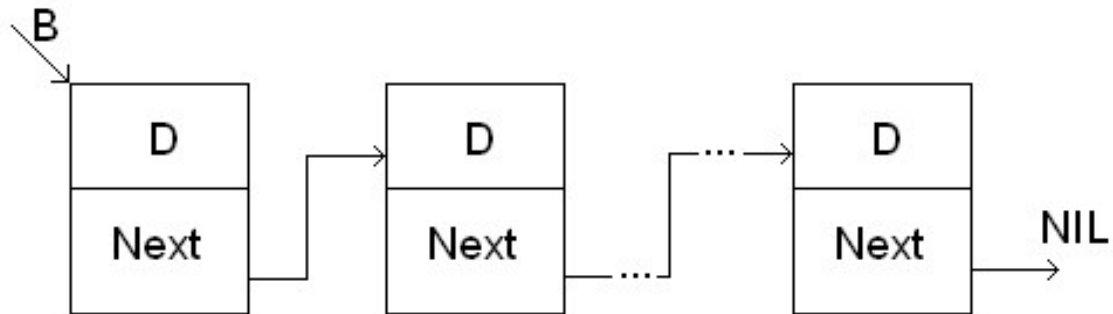


Рисунок 8.2 – Связный однонаправленный список

Здесь *B* является указателем на вершину списка. Каждый элемент списка содержит поле данных *D* и поле *Next* – указатель следующего элемента. Поле *Next* последнего элемента содержит пустую ссылку *NIL*.

Если *Temp* – переменная ссылочного типа, указывающая на элемент списка, предшествующий удаляемому, то чтобы исключить элемент из списка достаточно выполнить следующий оператор:

$Temp^{Next} := Temp^{Next}^{Next};$

Чтобы вставить элемент в произвольное место списка, кроме начала, необходимо изменить значения двух указателей. Пусть ссылочная переменная *Place* указывает на элемент, после которого необходимо вставить новый элемент. Если новый элемент адресуется указателем *NewE*, то операция вставки реализуется операторами:

$NewE^{Next} := Place^{Next}; \quad Place^{Next} := NewE;$

При добавлении элемента в начало списка или исключении начального элемента необходимо использовать указатель вершины списка *B*. Операция удаления первого элемента списка реализуется одним оператором

$B := B^{Next};$

Операция добавления в начало списка – двумя операторами:

$NewE^{Next} := B; \quad B := NewE;$

### 8.2.3 Пример программы работы с очередью

Рассмотрим программу, выполняющую организацию очереди, добавление элемента в очередь, удаление элемента из очереди. Очередь – это линейный список, добавление элементов в который выполняется с одной стороны, а исключение – с другой. При этом используются специальные указатели начала и конца очереди. В примере элементами очереди являются записи, содержащие сведения о грузополучателе. Каждая запись содержит следующие поля: фамилия, имя, отчество грузополучателя, адрес.

```

Program DemoQuery;
Uses Crt;
Type
  Data= Record {Описание записи о грузополучателе}
    FIO:string[15];
    Adr:string[30]
  End;
  Ukaz=^Query; {Описание указателя на элемент очереди}
  Query=Record
    Inf:Data;
    Next:Ukaz
  End;
Var NewE, Left, Right, Temp:Ukaz; {Указатели}
    Z:Data; {Запись, добавляемая в очередь}
    Key:Char; {номер пункта меню программы}
Procedure Org; {Процедура организации очереди}
Begin
  Writeln('Выполняется процедура организации очереди');
  Writeln('Для выхода из процедуры вводите символ * ');
  Writeln('===== ');
  Writeln('Введите фамилию и инициалы грузополучателя');
  Readln(Z.Fio);
  If Z.Fio='*' Then Exit; {Выход из процедуры при вводе '*'}
  Writeln('Введите адрес грузополучателя ');
  Readln(Z.Adr);
  New(NewE);           {Создание нового элемента      }
  NewE^.Inf.Fio:=Z.Fio; {Заполнение его полей          }
  NewE^.Inf.Adr:=Z.Adr; ;
  NewE^.Next:=nil;
  Right:=NewE; { Right - указатель хвоста очереди }
  Left:=NewE; { Left - указатель головы очереди }
  While True Do {Повторение этих же действий      }
  Begin
    Writeln('Введите фамилию и инициалы грузополучателя');
    Readln(Z.Fio);
    If Z.Fio='*' Then Exit;

```

```

    Writeln('Введите адрес грузополучателя');
    Readln(Z.Adr);
    New(NewE);
    NewE^.Inf.Fio:=Z.Fio;
    NewE^.Inf.Adr:=Z.Adr;
    NewE^.Next:=Nil;
    Right^.Next:=NewE; {Связь с предыдущим элементом}
    Right:=NewE        {Перемещение указателя хвоста очереди}
End
End;

Procedure Dob; {Добавление элемента в конец очереди}
Begin
    Writeln('Введите фамилию и инициалы грузополучателя');
    Readln(Z.Fio);
    If Z.Fio='' Then Exit;
    Writeln('Введите адрес грузополучателя');
    Readln(Z.Adr);
    New(NewE);
    NewE^.Inf.Fio:=Z.Fio;
    NewE^.Inf.Adr:=Z.Adr;
    NewE^.Next:=nil;
    If Right=Nil Then      {Если добавляется первый элемент, то}
        Left:=NewE        {инициализировать указатель головы}
    Else
        Right^.Next:=NewE; {иначе добавить в конец очереди}
    Right:=NewE;
End;

Procedure Udal; {Процедура исключения элемента}
Begin
    Writeln('Исключается головной элемент очереди');
    Writeln('Нажмите клавишу Enter');
    Readln;
    If Left<>Nil Then      {Если очередь не пустая, то}
    Begin
        Temp:=Left;      {запоминаем указатель на голову очереди}
        Left:=Left^.Next; {указатель головы смещаем на 2-ой элемент}
        Dispose(Temp);    {освобождаем память от головного элемента}
        If Left=Nil Then   {Если удалили последний элемент, то}
            Right:=Nil;    {указатель на конец очереди равен Nil}
    End
End;

Procedure Prosmotr; {Процедура просмотра очереди}
Var i:integer;      {Просмотр выполняется от головы к хвосту}
Begin

```



```

Writeln('Очередь содержит следующие элементы:');
Temp:=Left;
i:=1;
While Temp<>nil Do
Begin
  Writeln(i, ' ',Temp^.Inf.Fio,' ',Temp^.Inf.Adr);
  Temp:=Temp^.Next;
  i:=i+1;
End;
Writeln('Нажмите клавишу Enter');
Readkey;
End;
{=====основная программа=====}
begin
  Right:=Nil;   Left:=Nil;
  Repeat
    ClrScr; { очистка экрана }
  {вывод на экран пунктов меню}
    Writeln('1-Организация очереди');
    Writeln('2-Добавление элемента в очередь');
    Writeln('3-Удаление элемента из очереди');
    Writeln('4-Просмотр очереди');
    Writeln('5-Выход');
    Writeln('-----');
    Writeln('Нажмите клавишу от 1 до 5');
    Key:=ReadKey; {считывание кода нажатой клавиши}
    Case Key Of { вызов необходимой процедуры по номеру}
      '1':Org;
      '2':Dob;
      '3':Udal;
      '4':Prosmotr;
    End
  Until Key='5' {выход из программы}
End.

```

### 8.3 Варианты заданий

Представить одну из приведенных ниже таблиц в виде линейного списка L, элементами которого являются строки таблицы. Написать процедуры организации, добавления элемента в список, исключения элемента из списка, удаления списка, просмотра списка, а также одну из процедур в соответствии с вариантом, приведенным ниже.

Значения и количество записей в таблице студент выбирает самостоятельно. Исходные данные после организации списка должны сохраняться в файле и при повторном запуске программы считываться из файла. Количество строк таблицы не задается.

**Таблица 8.1 – Ведомость**

N	Фамилия	Имя	Отчество	Оценки		
				Математика	История	Физика

**Таблица 8.2 – Расписание**

N Поезда	Станция отправления	Станция назначения	Время отправления	Время прибытия	Стоимость билета
----------	---------------------	--------------------	-------------------	----------------	------------------

**Таблица 8.3 – Анкета**

N	ФИО	Год рождения	Пол	Семейное состояние	Количество детей	Оклад
---	-----	--------------	-----	--------------------	------------------	-------

**Вариант 1** Таблица 8.1. Процедуру, которая вставляет в начало списка L новый элемент E.

**Вариант 2** Таблица 8.2. Процедуру, которая вставляет в начало списка L новый элемент E.

**Вариант 3** Таблица 8.3. Процедуру, которая вставляет в начало списка L новый элемент E.

**Вариант 4** Таблица 8.1. Процедуру, которая вставляет в конец списка L новый элемент E.

**Вариант 5** Таблица 8.2. Процедуру, которая вставляет в конец списка L новый элемент E.

**Вариант 6** Таблица 8.3. Процедуру, которая вставляет в конец списка L новый элемент E.

**Вариант 7** Таблица 8.1. Процедуру, которая вставляет новый элемент E после первого элемента непустого списка L.

**Вариант 8** Таблица 8.2. Процедуру, которая вставляет новый элемент E после первого элемента непустого списка L.

**Вариант 9** Таблица 8.3. Процедуру, которая вставляет новый элемент E после первого элемента непустого списка L.

**Вариант 10** Таблица 8.1. Процедуру, которая вставляет в список L новый элемент E1 за каждым вхождением элемента E.

**Вариант 11** Таблица 8.2. Процедуру, которая вставляет в список L новый элемент E1 за каждым вхождением элемента E.

**Вариант 12** Таблица 8.3. Процедуру, которая вставляет в список L новый элемент E1 за каждым вхождением элемента E.

**Вариант 13** Таблица 8.1. Процедуру, которая вставляет в непустой список L пару новых элементов E1 и E2 перед его последним элементом.

**Вариант 14** Таблица 8.2. Процедуру, которая вставляет в непустой список L пару новых элементов E1 и E2 перед его последним элементом.

**Вариант 15** Таблица 8.3. Процедуру, которая вставляет в непустой список L пару новых элементов E1 и E2 перед его последним элементом.

**Вариант 16** Таблица 8.1. Процедуру, которая вставляет в непустой список L, элементы которого упорядочены по возрастанию значений одного из полей таблицы, новый элемент E так, чтобы сохранилась упорядоченность.

**Вариант 17** Таблица 8.2. Процедуру, которая вставляет в непустой список L, элементы которого упорядочены по возрастанию значений одного из полей таблицы, новый элемент E так, чтобы сохранилась упорядоченность.

**Вариант 18** Таблица 8.3. Процедуру, которая вставляет в непустой список L, элементы которого упорядочены по возрастанию значений одного из полей таблицы, новый элемент E так, чтобы сохранилась упорядоченность

**Вариант 19** Таблица 8.1. Процедуру, которая удаляет из непустого списка L последний элемент.

**Вариант 20** Таблица 8.2. Процедуру, которая удаляет из непустого списка L последний элемент.

**Вариант 21** Таблица 8.3. Процедуру, которая удаляет из непустого списка L последний элемент.

**Вариант 22** Таблица 8.1. Процедуру, которая удаляет из непустого списка L второй элемент, если такой есть.

**Вариант 23** Таблица 8.2. Процедуру, которая удаляет из непустого списка L второй элемент, если такой есть.

**Вариант 24** Таблица 8.3. Процедуру, которая удаляет из непустого списка L второй элемент, если такой есть.

**Вариант 25** Таблица 8.1. Процедуру, которая печатает фамилии отличников из непустого списка L, если таких нет, то выводит на экран соответствующее сообщение.

**Вариант 26** Таблица 8.2. Функцию, подсчитывающую среднюю стоимость билета.

**Вариант 27** Таблица 8.3. Функцию, вычисляющую средний оклад женщин из непустого списка L.

**Вариант 28** Таблица 8.1. Функцию, которая проверяет, есть ли в списке L хотя бы два элемента с равными значениями второго поля.

**Вариант 29** Таблица 8.2. Функцию, которая проверяет, есть ли в списке L хотя бы два элемента с равными значениями второго поля.

**Вариант 30** Таблица 8.3. Функцию, которая проверяет, есть ли в списке L хотя бы два элемента с равными значениями третьего поля.

## **8.4 Порядок выполнения работы**

8.4.1 Выбрать вид линейного списка, подходящий для решения задачи.

8.4.2 Разработать алгоритм решения задачи, разбив его на отдельные процедуры и функции, как указано в варианте задания.

8.4.3 Разработать программу на языке Pascal.

8.4.4 Разработать тестовые примеры, которые предусматривают проверку корректности работы программы для пустых списков.

8.4.5 Выполнить отладку программы.

8.4.6 Исследовать результаты работы программы для различных режимов ее использования.

### **8.5 Содержание отчета**

Цель работы, постановка задачи, структурная схема алгоритма, текст программы с комментариями, описание тестов и результатов тестирования программы, выводы.

### **8.6 Контрольные вопросы**

8.6.1 Чем различаются статические и динамические переменных?

8.6.2 Как осуществляется задание ссылочных переменных?

8.6.3 Для чего используется переменная с указателем?

8.6.4 Какое назначение процедур **New** и **Dispose**, **GetMem** и **FreeMem**?

8.6.5 Какие операции определены над ссылочными переменными?

8.6.6 Какие операции определены над переменными с указателем?

8.6.7 Что понимают под линейной структурой данных?

8.6.8 Что называется последовательной структурой и списковой структурой данных?

8.6.9 Определите различные виды линейных односвязных списков.

8.6.10 Как описать элемент списковой структуры на языке Паскаль?

8.6.11 Напишите на языке Паскаль процедуры для работы с однонаправленными списками:

- создания списка;
- добавления элемента в список;
- удаления элемента из списка.

## 9 ЛАБОРАТОРНАЯ РАБОТА №9

### “ПРОГРАММИРОВАНИЕ НЕЛИНЕЙНЫХ СТРУКТУР ДАННЫХ”

#### 9.1 Цель работы

Исследование нелинейных структур данных и приобретение навыков разработки и отладки программ, использующих древовидные структуры. Исследование особенностей работы с поисковыми бинарными деревьями.

#### 9.2 Краткие теоретические сведения

##### 9.2.1 Бинарные деревья и алгоритмы их обработки

Нелинейные структуры предназначены для отображения связей подчиненности элементов данных. Примером нелинейных структур являются иерархические древовидные структуры. В представлении данных наибольшее распространение получили *бинарные деревья*. Любая вершина бинарного дерева связана только с вершиной ближайшего высшего уровня и с двумя вершинами низшего уровня.

Вершина (узел) дерева, которая не имеет вышестоящей вершины (предка), называется *корнем*. Считается, что корень дерева расположен на первом уровне. Каждая вершина уровня  $k$  имеет одного предка на уровне  $k-1$ . Максимальный уровень дерева называется его *глубиной*. Среди любых пар непосредственно связанных вершин можно выделить предка и дочернюю вершину. Вершина дерева, которая не имеет дочерних вершин, называется *листом*.

В связном списке каждый элемент содержит указатель на другой элемент. Бинарное дерево похоже на связный список с тем отличием, что каждый узел содержит два указателя: на левое бинарное поддерево и правое бинарное поддерево. Ниже приведен пример описания узла бинарного дерева:

TYPE

TreePtr=^Tree;

Tree=Record

Data: String; {тип значения, хранимый в вершине}

Left,Right: TreePtr {указатели на левое и правое поддерево}

End;

Наиболее распространенным условием организации бинарных деревьев является упорядоченность. Элементы дерева в этом случае снабжаются ключевыми признаками. Каждый элемент в упорядоченном дереве имеет на своей левой ветви элементы с меньшими, чем у него, значениями ключей, а на правой ветви элементы с большими значениями ключей.

Упорядоченные бинарные деревья обеспечивают быстрый поиск записи по ее ключу. Такие деревья называют *деревьями бинарного поиска*. Ниже приведен пример описания функции поиска записи по ключу в бинарном дереве.

Function Search(Root:TreePtr; Node:string):TreePtr;

{ Node – ключевое (поисковое) значение, например типа string;

Root – указатель на корень дерева (поддерева)}

Var Found:Boolean; {признак успешности поиска}

```

{ Если дерево содержит вершину Node то функция
  возвращает указатель на эту вершину}
Begin
  Found:=False;
  While (Root<>Nil) and (not Found) Do
    {повторять цикл пока не дошли до листа или не нашли значение}
    If Root^.Data=Node Then {Проверить значение вершины}
      Found:=True           {значение найдено}
    Else                     {значение не найдено}
      If Root^.Data>Node Then
        Root:=Root^.Left;   {поиск в левом поддереве}
      Else
        Root:=Root^.Right;  {поиск в правом поддереве}
    Search:=Root;
  End;

```

Рассмотрим алгоритм построения упорядоченных бинарных древовидных структур. Пусть имеется неупорядоченный массив  $A[1], A[2], \dots, A[n]$ . Первый элемент  $A[1]$  назовем корнем. Сравним второй элемент  $A[2]$  с первым элементом  $A[1]$ . Если  $A[2]>A[1]$ , запишем в правый указатель элемента  $A[1]$  адрес элемента  $A[2]$ , иначе адрес  $A[2]$  запишем в левый указатель. На следующем шаге сравниваем  $A[3]$  с  $A[1]$ . При  $A[3]>A[1]$  сравниваем  $A[3]$  с элементом, адрес которого хранится в правом указателе, а если этого адреса нет, то записываем в правый указатель элемента  $A[1]$  адрес элемента  $A[3]$ . Если же в правом указателе элемента  $A[1]$  указан адрес элемента  $A[2]$ , то сравниваем  $A[3]$  с  $A[2]$  и заносим адрес элемента  $A[3]$  в правый или левый указатель элемента  $A[2]$  в зависимости от результатов сравнения. Процесс повторяется для каждого нового элемента. В этом случае новые элементы всегда присоединяются к листьям дерева. Алгоритм добавления элемента в дерево может быть сформулирован рекурсивно:

```

Function AddTree(Top:TreePtr; NewNode: String):TreePtr;
  {Top – указатель на корень дерева (поддерева);
   NewNode – добавляемое значение элемента}
Begin
  If Top=Nil Then {указатель нулевой, если дошли до листа}
    Begin
      New(Top);      {выделить память для нового узла}
      Top^.Data:=Node;{записать в узел значение}
      Top^.Left:=Nil; {указатели на поддеревья пустые}
      Top^.Right:=Nil;
    End
  Else
    IF Top^.Data>NewNode Then {поиск места вставки}
      Top^.Left:=AddTree(Top^.Left,NewNode)
    Else
      Top^.Right:=AddTree(Top^.Right,NewNode);

```

```

    AddTree:=Top
End;
```

Эффективность поиска в бинарном дереве в значительной степени зависит от его симметричности. Под *симметричным* понимается дерево, которое состоит из  $n$  уровней, причем  $n-1$  уровень занят полностью. Если листья дерева располагаются только на двух соседних уровнях  $n-1$  и  $n$ , а  $n-1 = \text{trunc}(\log_2 M)$ , то такое дерево называется *выровненным* ( $M$  – количество вершин дерева).

Формирование упорядоченного выровненного дерева можно выполнить с помощью следующего алгоритма. Исходные записи должны иметь вид упорядоченного массива. За корень дерева принимается средняя запись массива, которая разделяет массив примерно на две равные части. Средние записи в обеих частях массива образуют вершины второго уровня, а массив оказывается разделенным на 4 части. Средние записи каждой из четырех частей помещаются на третьем уровне бинарного дерева. Процесс продолжается до тех пор, пока все записи массива не будут внесены в дерево.

Чтобы распечатать дерево бинарного поиска в заданном порядке используют различные *стратегии обхода дерева*:

- обход сверху;
- обход слева направо;
- обход снизу.

Для обхода дерева применяют процедуры:

- 1) обработки корня дерева или поддеревя;
- 2) обход левого поддеревя;
- 3) обход правого поддеревя.

Если перечисленные процедуры выполняются в порядке 1-2-3, то выполняется обход сверху; если в порядке 2-1-3, то выполняется обход слева направо; если в порядке 2-3-1, то выполняется обход снизу. Для печати дерева с сохранением относительного порядка используют обход слева направо. Пример процедуры обхода двоиного дерева слева направо:

```

Procedure LR(Top:TreePtr);
Begin
  If Top<>Nil Then
    Begin
      LR(Top^.Left); {обойти левое поддерево}
      Writeln(Top^.Data);
      LR(Top^.Right) {обойти правое поддерево}
    End
  End;
```

### 9.2.2 Пример программы обработки бинарного дерева

Пусть требуется написать процедуры создания, добавления листа в бинарное дерево, просмотра бинарного дерева, отображения структуры дерева, а также рекурсивную процедуру, которая подсчитывает число вершин на  $n$ -ом уровне непустого дерева  $T$  (корень считать вершиной 1-го уровня).

Ниже приведен текст соответствующей программы с комментариями. С каждым узлом дерева связана запись, состоящая из фамилии и адреса грузополучателя. Дерево упорядочено по фамилиям.

```

Program TreeProcess;
Uses Crt;
Type
  Zap=Record {Описание записи о грузополучателе}
    FIO:string[15];
    Adr:string[30]
  End;
  TreePtr=^Tree; {Описание узла дерева}
  Tree=Record
    Data:Zap;
    Left,Right:TreePtr
  End;
Var  Top:Treeptr;
      Z:Zap;
      Level,N,i:Integer;
      Number:Integer;
{Функция добавляющая лист к дереву}
Function AddTree (Top:TreePtr;Newnode:Zap):TreePtr;
Begin
  If Top=nil THEN
    Begin
      New(Top);
      Top^.Data:=Newnode;
      Top^.Left:=Nil;
      Top^.Right:=Nil;
    End
  Else
    If Top^.Data.Fio>Newnode.Fio Then
      Top^.Left:=AddTree(Top^.Left,Newnode)
    Else
      Top^.Right:=AddTree(Top^.Right,Newnode);
    Addtree:=Top
  End;
Procedure OrgTree;
Begin
  Writeln('Выполняется процедура организации дерева');
  Writeln('Для выхода из процедуры вводите символ * ');
  Writeln('===== ');
  Top:=nil;
  While True Do
    Begin

```



```

    Writeln('Введите фамилию и инициалы грузополучателя');
    Readln(Z.Fio);
    If Z.Fio='*' Then Exit;
    Writeln('Введите адрес грузополучателя');
    Readln(Z.Adr);
    Top:=Addtree(Top,Z);
End
End;
Procedure DobL;
Begin
    Writeln('Выполняется процедура добавления листа');
    Writeln('Для выхода из процедуры вводите символ * ');
    Writeln('===== ');
    Writeln('Введите фамилию и инициалы грузополучателя');
    Readln(Z.Fio);
    If Z.Fio='*' Then Exit;
    Writeln('Введите адрес грузополучателя');
    Readln(Z.Adr);
    Top:=Addtree(Top,Z);
End;
Procedure Prosmotr(Top:TreePtr);
{Процедура просмотра значений узлов дерева слева направо}
Begin
    If Top<>Nil Then
        Begin
            Prosmotr(Top^.Left);
            Writeln(i,' ',Top^.Data.Fio,' ',Top^.Data.Adr);
            i:=i+1;
            Prosmotr(Top^.Right)
        End;
End;
Procedure Otobr(Top:TreePtr;Otstup:Integer);
{Процедура отображения структуры дерева. Дерево отображается по-
вернутым на 90 градусов против часовой стрелки. Узлы дерева, находящиеся
на одном уровне, отображаются с одинаковым отступом от края экрана.}
Begin
    If Top<>Nil Then
        Begin
            Otstup:=Otstup+3;
            Otobr(Top^.Right,Otstup);
            Writeln(' ':Otstup,Top^.Data.Fio);
            Otobr(Top^.Left,Otstup);
        End
    End;
End;

```

```

Procedure NodeCount(Top:TreePtr; Level:Integer; Var N:Integer);
{Процедура подсчета количества вершин уровня Level}
Begin
  If (Level>=1) and (Top<>Nil) Then
    Begin
      If Level=1 Then N:=N+1;
      NodeCount(Top^.Left,Level-1,N);
      NodeCount(Top^.Right,Level-1,N);
    End
  End;
End;

{=====основная программа=====}
Begin
  { цикл, обеспечивающий вывод на экран пунктов меню}
  Repeat
    ClrScr; { очистка экрана }
    Writeln('1-Организация двоичного дерева');
    Writeln('2-Добавление листа к дереву');
    Writeln('3-Просмотр дерева');
    Writeln('4-Подсчет количества вершин на n-ом уровне');
    Writeln('5-Выход');
    Writeln('-----');
    Writeln('Введите номер пункта меню');
    Readln(Number);
    Case Number Of { вызов необходимой процедуры по номеру}
      1:OrgTree;
      2:Dobl;
      3:Begin
        Writeln('Выполняется процедура просмотра дерева');
        Writeln('===== ');
        i:=0;
        Prosmotr(Top);
        Otobr(Top,1);
        Writeln('Нажмите клавишу Enter');
        Readkey
      End;
      4:Begin
        Writeln('Выполняется процедура подсчета количества');
        Writeln('вершин на n-ом уровне');
        Writeln('===== ');
        Write('Введите значение уровня-->');
        Read(Level);
        N:=0;
        NodeCount(Top,Level,N);
        Writeln;
        Writeln('На уровне ',Level,' находится ',N,' вершин');
      End;
    End
  Until Number=5;
End;

```

```

        Writeln('Нажмите клавишу Enter');
        ReadKey
    End;
End;
Until Number=5; {выход из цикла, если введено 5}
End.

```

### 9.3 Варианты заданий

Представить приведенную в предыдущей работе таблицу в виде бинарного дерева. Написать процедуры создания и обхода дерева, а также одну из процедур или функций, приведенных ниже. Значения полей и количество записей в таблице студент выбирает самостоятельно. Программа должна сохранять дерево в файле и создавать его заново при её повторном запуске.

**Вариант 1** Таблица 8.1. Процедуру, которая присваивает параметру Е элемент из самого левого узла непустого дерева Т.

**Вариант 2** Таблица 8.2. Процедуру, которая присваивает параметру Е элемент из самого левого узла непустого дерева Т.

**Вариант 3** Таблица 8.3. Процедуру, которая присваивает параметру Е элемент из самого левого узла непустого дерева Т.

**Вариант 4** Таблица 8.1. Процедуру, которая определяет уровень, на котором находится элемент Е в дереве Т.

**Вариант 5** Таблица 8.2. Процедуру, которая определяет уровень, на котором находится элемент Е в дереве Т.

**Вариант 6** Таблица 8.3. Процедуру, которая определяет уровень, на котором находится элемент Е в дереве Т.

**Вариант 7** Таблица 8.1. Процедуру, которая вычисляет среднее арифметическое всех элементов непустого дерева Т (по одному из полей таблицы, которое имеет числовое значение)

**Вариант 8** Таблица 8.2. Процедуру, которая вычисляет среднее арифметическое всех элементов непустого дерева Т (по одному из полей таблицы, которое имеет числовое значение)

**Вариант 9** Таблица 8.3. Процедуру, которая вычисляет среднее арифметическое всех элементов непустого дерева Т (по одному из полей таблицы, которое имеет числовое значение)

**Вариант 10** Таблица 8.1. Процедуру, которая заменяет в дереве Т все элементы меньшие, чем некоторое положительное число А, на это число (по одному из полей таблицы, которое имеет числовое значение).

**Вариант 11** Таблица 8.2. Процедуру, которая заменяет в дереве Т все элементы меньшие, чем некоторое положительное число А, на это число (по одному из полей таблицы, которое имеет числовое значение).

**Вариант 12** Таблица 8.3. Процедуру, которая заменяет в дереве Т все элементы меньшие, чем некоторое положительное число А, на это число (по одному из полей таблицы, которое имеет числовое значение).

**Вариант 13** Таблица 8.1. Процедуру, которая печатает элементы всех листьев дерева  $T$ .

**Вариант 14** Таблица 8.2. Процедуру, которая печатает элементы всех листьев дерева  $T$ .

**Вариант 15** Таблица 8.3. Процедуру, которая печатает элементы всех листьев дерева  $T$ .

**Вариант 16** Таблица 8.1. Процедуру, которая находит в непустом дереве  $T$  длину (число ветвей) пути от корня до вершины с элементом  $E$ .

**Вариант 17** Таблица 8.2. Процедуру, которая находит в непустом дереве  $T$  длину (число ветвей) пути от корня до вершины с элементом  $E$ .

**Вариант 18** Таблица 8.3. Процедуру, которая находит в непустом дереве  $T$  длину (число ветвей) пути от корня до вершины с элементом  $E$ .

**Вариант 19** Таблица 8.1. Написать рекурсивную процедуру или функцию, которая определяет, входит ли элемент в дерево  $T$ .

**Вариант 20** Таблица 8.2. Написать рекурсивную процедуру или функцию, которая определяет, входит ли элемент в дерево  $T$ .

**Вариант 21** Таблица 8.3. Написать рекурсивную процедуру или функцию, которая определяет, входит ли элемент в дерево  $T$ .

**Вариант 22** Таблица 8.1. Написать рекурсивную процедуру или функцию, которая определяет число вхождений элемента  $E$  в дерево  $T$ .

**Вариант 23** Таблица 8.2. Написать рекурсивную процедуру или функцию, которая определяет число вхождений элемента  $E$  в дерево  $T$ .

**Вариант 24** Таблица 8.3. Написать рекурсивную процедуру или функцию, которая определяет число вхождений элемента  $E$  в дерево  $T$ .

**Вариант 25** Таблица 8.1. Написать рекурсивную процедуру или функцию, которая вычисляет сумму элементов (по одному из полей таблицы) непустого дерева  $T$ .

**Вариант 26** Таблица 8.2. Написать рекурсивную процедуру или функцию, которая вычисляет сумму элементов (по одному из полей таблицы) непустого дерева  $T$ .

**Вариант 27** Таблица 8.3. Написать рекурсивную процедуру или функцию, которая вычисляет сумму элементов (по одному из полей таблицы) непустого дерева  $T$ .

**Вариант 28** Таблица 8.1. Процедуру, которая подсчитывает число вершин на  $n$ -ом уровне непустого дерева  $T$ .

**Вариант 29** Таблица 8.2. Процедуру, которая подсчитывает число вершин на  $n$ -ом уровне непустого дерева  $T$ .

**Вариант 30** Таблица 8.3. Процедуру, которая подсчитывает число вершин на  $n$ -ом уровне непустого дерева  $T$ .

#### **9.4 Порядок выполнения работы**

9.4.1 Описать элемент бинарного дерева в соответствии с вариантом задачи.

9.4.2 Разработать алгоритм решения задачи, разбив его на отдельные процедуры и функции, как указано в варианте задания.

9.4.3 Разработать программу на языке Pascal.

9.4.4 Разработать тестовые примеры, которые предусматривают проверку корректности работы программы в разных режимах.

9.4.5 Выполнить отладку программы.

9.4.6 Исследовать результаты работы программы для различных режимов ее использования.

### **9.5 Содержание отчета**

Цель работы, описание математического метода решения задачи, структурная схема алгоритма, текст программы, анализ результатов вычислений, выводы.

Цель работы, вариант задания, структурные схемы процедур с описанием, текст программы с комментариями, тестовые примеры, результаты вычислений, выводы.

### **9.6 Контрольные вопросы**

9.6.1 Что понимают под нелинейной структурой данных?

9.6.2 Что называется бинарным деревом?

9.6.3 Как построить упорядоченное бинарное дерево?

9.6.4 От чего зависит эффективность поиска в бинарном дереве?

9.6.5 Что понимают под симметричным и выровненным деревом?

9.6.6 Как формируется выровненное дерево?

9.6.7 Напишите на языке Паскаль процедуры для работы с бинарными деревьями:

- создания упорядоченного бинарного дерева;
- добавления элементов в дерево;
- удаления листа дерева;
- процедуры обхода дерева.

## 10 ЛАБОРАТОРНАЯ РАБОТА №10

### «ПРОГРАММИРОВАНИЕ С ИСПОЛЬЗОВАНИЕМ ОБЪЕКТОВ И МОДУЛЕЙ»

#### 10.1 Цель работы

Изучение базовых понятий объектно-ориентированного программирования, приобретение навыков разработки и отладки программ, использующих модули и объекты. Исследование особенностей поисковых бинарных деревьев, представленных в виде объекта.

#### 10.2 Краткие теоретические сведения

##### 10.2.1 Модули

*Модуль* в языке Turbo Pascal – это отдельно хранимая и независимо компилируемая программа. Модуль содержит коллекцию программных ресурсов, которые можно использовать в других программах. Модуль содержит описания типов данных, переменных и других объектов, а также подпрограммы. Модуль сам по себе не является выполнимой программой – его объекты используются другими программными единицами. Подпрограммы, входящие в модуль, можно написать, отладить и откомпилировать один раз, а использовать многократно. Доступ к ресурсам модуля из других программ обеспечивает оператор **uses**, в котором указывается имя модуля. Этот оператор размещается в разделе описаний программы сразу после заголовка. Если в программе используется не один модуль, а несколько, необходимо указать имена всех модулей, перечислив их через запятую. Исключением является модуль **System**, ссылка на который необязательна. Этот модуль содержит, в частности, процедуры файлового ввода-вывода, процедуры и функции для работы со строками и некоторые другие.

Модуль начинается заголовком:

**unit <имя модуля>;**

Имя модуля выбирается в соответствии с правилами языка Turbo Pascal. Файл, содержащий модуль, обязан иметь то же имя, что и модуль. Структура модуля изображена на рисунке 10.1.

<b>Unit &lt;имя модуля&gt;</b>
Interface
Интерфейсная секция
Implementation
Секция реализации
Секция инициализации

Рисунок 10.1 – Структура модуля

Модуль состоит из трех секций. Первая секция – *интерфейсная* – содержит описания типов, переменных и других объектов данных, которые можно использовать в других программах или модулях. Она начинается с зарезервированного слова **interface**. Для процедур и функций в интерфейсной секции указывают только заголовки, а их полные описания содержатся в секции реализации. *Секция реализации* начинается с зарезервирован-

ного слова **implementation**. Все описания, содержащиеся в секции реализации, являются локальными, их область действия – данный модуль. Здесь же содержатся полные описания функций и процедур модуля. Последняя часть модуля – *секция инициализации* (начинается словом **begin**, а заканчивается словом **end**). Она может быть пустой или включать в себя исполняемые операторы, реализующие необходимые действия по инициализации (например, присваиванию начальных значений переменным) модуля. Пример модуля будет приведен ниже.

В результате компиляции модуля на диске создается файл с расширением **tpu**.

### 10.2.2 Объекты. Основные понятия

Программа, написанная с использованием методологии объектно-ориентированного программирования (ООП), состоит из объектов, которые взаимодействуют между собой. *Объект* – это реальная либо абстрактная сущность, которая моделирует часть предметной области программы. С точки зрения программиста объект состоит из атрибутов и методов. *Атрибуты* описывают свойства объекта, а *методы* – характерное для объекта поведение. Поэтому программная реализация объекта представляет собой объединение данных (атрибутов) и процедур их обработки (методов). В языке Turbo Pascal для описания объектов применяется тип **object**, который можно считать обобщением структурного типа **record**. Переменные объектового типа называются *экземплярами* объекта.

В отличие от типа “запись” объектовый тип содержит не только поля, описывающие данные, но также процедуры и функции, описания которых входят в описание объекта. Эти процедуры и функции называются *методами*. Методам объекта доступны его атрибуты (поля данных). Непосредственно в объектовом типе задаются только заголовки методов, а их полные описания должны быть сделаны вне объектового типа. Приведем пример описания объекта:

```
Type
Location = object
    X,Y : Integer;
    procedure Init(InitX, InitY: Integer);
    function GetX : Integer;
    function GetY : Integer;
end;
```

Объект описывается с помощью зарезервированных слов **object** и **end**, между которыми находятся описания полей данных (атрибутов) и методов. В приведенном примере объект содержит два поля **X** и **Y** для хранения значений координат, а также заголовки двух функций (**GetX**, **GetY**) и процедуры (**Init**) – методов данного объекта. В отличие от других описаний описание объектного типа может находиться только на самом верхнем уровне программной единицы (программы, модуля), в которой используется этот тип. В разделе описаний процедур и функций такое описание содержаться не может.

Зарезервированное слово **private** позволяет ограничить доступ к полям объекта. В следующем примере доступ к полям **X** и **Y** возможен только посредством методов объектного типа **Location**:

```
Type Location = object
    procedure Init(InitX, InitY : Integer);
    function GetX : Integer;
```

```

function GetY : Integer;
private
    X, Y : Integer;
end;

```

В секции **private** могут находиться и методы объекта.

Полное описание методов, то есть описание их реализации, должно находиться после описания объекта (обычно в секции реализации модуля). В этом случае имя метода является составным и складывается из имени объекта и метода, разделяемых точкой:

```

procedure Location.Init(InitX, InitY : Integer);
begin
    X := InitX;
    Y := InitY;
end;
function Location.GetX : Integer;
begin
    GetX := X;
end;
function Location.GetY : Integer;
begin
    GetY := Y;
end;

```

После того как объект описан, в программе можно использовать его экземпляры, то есть переменные указанного объектного типа:

```
Var GrFigurePosition: Location;
```

### 10.2.3 Инкапсуляция

*Инкапсуляция* – это механизм, который позволяет защитить данные (атрибуты) и методы объекта от некорректного использования. В соответствии с принципами инкапсуляции атрибуты объекта не могут быть доступными непосредственно. Доступ к атрибутам (полям данных) осуществляется *только через соответствующие методы объекта*. Например, чтение значений атрибутов **X** и **Y** экземпляра объекта **GrFigurePosition** возможно только с помощью вызова его методов **GetX** и **GetY**:

```

Xpos:= GrFigurePosition.GetX;
Ypos:= GrFigurePosition.GetY;

```

При этом процедура инициализации **Init** и функции **GetX** и **GetY** уже не существуют как отдельные самостоятельные единицы. Это неотъемлемые части объектного типа **Location**. Если в программе имеется несколько экземпляров объектов (переменных) указанного типа, то для хранения данных каждого экземпляра объекта резервируется своя собственная область памяти, а указатели на точки входа в процедуру и функции являются общими. Вызов каждого метода возможен только с помощью составного имени, явно указывающего, для обработки каких полей данных предназначен данный метод.

### 10.2.4 Наследование



Объектно-ориентированный подход позволяет определить новый объект, как потомок другого объекта:

Type

```
Point = object(Location)
    Visible : Boolean;
    procedure Show;
    procedure Hide;
    function IsVisible : Boolean;
    procedure MoveTo(NewX, NewY : Integer);
end;
```

В объектовом типе **Point** присутствуют все поля и методы типа **Location**. Описанное свойство называется *наследованием*, то есть потомок наследует свойства родителя. Наследником здесь является объект **Point**, представляющий точку на экране компьютера, а родителем – объект **Location**.

Такое определение объектового типа содержит следующие элементы:

- поля **X** и **Y**, унаследованные от **Location**;
- собственное поле **Visible**;
- унаследованные методы **Init**, **GetX**, **GetY**;
- новые методы **Show**, **Hide**, **IsVisible**, **MoveTo**.

Что касается совместимости объектных типов, то экземпляру объекта родительского типа можно присваивать значения объекта-наследника, но не наоборот.

### 10.2.5 Виртуальные методы и полиморфизм

Связь между *виртуальными* методами и вызывающими их процедурами устанавливается во время выполнения программы (это называется *поздним связыванием*, в отличие от *раннего связывания*, когда связь устанавливается во время компиляции). В описании виртуального метода после заголовка метода указывается зарезервированное слово **virtual**. Заголовки виртуальных методов родителя и наследника должны совпадать. Инициализация экземпляра объекта, имеющего виртуальные методы, выполняется с помощью специального метода – *конструктора*. Конструктор присваивает полям объекта начальные значения и выполняет другие необходимые действия по инициализации объекта. В заголовке метода-конструктора слово **procedure** заменяется словом **constructor**. Действия, обратные действиям конструктора, выполняет еще один специальный метод – *деструктор*. Его описание начинается со слова **destructor**.

Конструктор создает указатель на *таблицу виртуальных методов*, которая содержит адреса всех виртуальных методов объекта и в дальнейшем используется для их поиска. При вызове виртуального метода по его имени определяется адрес, а затем по этому адресу передается управление. Если имеется несколько экземпляров объектов одного типа, то конструктор должен быть вызван отдельно для каждого экземпляра объекта при его инициализации. В противном случае возможен сбой в работе программы.

У каждого объектного типа имеется собственная таблица виртуальных методов, что позволяет одному и тому же оператору вызывать разные процедуры. Таким образом, вызов одного и того же метода будет работать по-разному, в зависимости от того какой объект его вызывает. Это свойство называют *полиморфизмом*.

Конструктор и деструктор могут быть и “пустыми”, то есть не содержать операторов. Весь необходимый код в этом случае создается автоматически.

#### 10.2.6. Динамическое создание объектов

Для работы с динамическими объектами используется расширенный синтаксис процедур **New** и **Dispose**. Обе процедуры в этом случае содержат в качестве второго параметра имя конструктора или деструктора объекта:

**New**(P, <имя конструктора>)

**Dispose**(P, <имя деструктора>).

Здесь P – указатель на переменную объектного типа.

Действие процедуры **New** в случае расширенного синтаксиса равносильно действию следующих операторов:

**New**(P); P<sup>^</sup>.<имя конструктора>;

Вызов процедуры **Dispose** эквивалентен такой последовательности:

P<sup>^</sup>.<имя деструктора>; **Dispose**(P);

#### 10.2.7 Пример модульной программы для работы с объектом “очередь”

Требуется создать модуль, содержащий описание объекта, который представляет очередь. Объект должен обладать возможностью организации очереди, добавления элемента в очередь, удаления элемента из очереди, просмотра очереди. Написать программу для представления данных о грузополучателях в виде очереди. Программа должна содержать меню, позволяющее проверить работу всех методов объекта.

В качестве основы для данной программы возьмем пример программы работы с очередью из лабораторной работы №9. При этом все необходимые типы данных, в том числе и объект, представляющий очередь, опишем в модуле **DynQuery**. Данный модуль будет подключаться к программе в операторе **Uses**. Программа будет выводить на экран соответствующие пункты меню и вызывать необходимые методы объекта.

В программе создается экземпляр объекта **Query**, описание которого приведено в модуле **DynQuery**. Данный объект содержит описание двух полей-указателей на начало (**Left**) и конец очереди (**Right**), а также описание всех методов, необходимых для работы с очередью. При этом доступ к значениям указателей имеют только методы объекта, так как указатели описаны в секции **Private**.

Unit **DynQuery**; {Модуль для работы с очередью}

{=====интерфейсная секция модуля=====}

Interface

Type

Data= Record {Описание записи о грузополучателе}

FIO:string[15];

Adr:string[30]

End;

Ukaz=^ElQuery; {Описание указателя на элемент очереди}

ElQuery=Record

```

        Inf:Data;
        Next:Ukaz
    End;
Query=object    {Описание объекта}
    Procedure Init;    {Метод инициализирующий указатели}
    Procedure Org;      {Метод организации очереди}
    Procedure Dob;      {Метод добавляющий элемент}
    Procedure Udal;     {Метод исключаящий элемент}
    Procedure Prosmotr; {Метод для просмотра очереди}
    Private
        Left:Ukaz;      {Указатель на голову очереди}
        Right:Ukaz;     {Указатель на хвост очереди}
    End;
{=====секция реализации модуля=====}
Implementation
Procedure Query.Init; {Метод инициализирующий указатели}
Begin
    Right:=Nil;        {Начальное значение указателей Nil}
    Left:=Nil;
End;
Procedure Query.Org; {Метод (процедура)организации очереди}
Var  Z:Data;          {Запись, добавляемая в очередь}
      NewE:Ukaz;
Begin
    Writeln('Выполняется процедура организации очереди');
    Writeln('Для выхода из процедуры вводите символ * ');
    Writeln('===== ');
    Writeln('Введите фамилию и инициалы грузополучателя');
    Readln(Z.Fio);
    Writeln('Введите адрес грузополучателя ');
    Readln(Z.Adr);
    If Z.Fio='*' Then Exit; {Выход из процедуры при вводе '*'}
    New(NewE);             {Создание нового элемента }
    NewE^.Inf.Fio:=Z.Fio;  {Заполнение его полей }
    NewE^.Inf.Adr:=Z.Adr; ;
    NewE^.Next:=nil;
    Right:=NewE; { Right - указатель хвоста очереди }
    Left:=NewE;  { Left - указатель головы очереди }
    While True Do {Повторение этих же действий }
    Begin
        Writeln('Введите фамилию и инициалы грузополучателя');
        Readln(Z.Fio);    If Z.Fio='*' Then Exit;
        Writeln('Введите адрес грузополучателя');
        Readln(Z.Adr);
    End;
End;

```

```

    New(NewE);
    NewE^.Inf.Fio:=Z.Fio;
    NewE^.Inf.Adr:=Z.Adr;
    NewE^.Next:=Nil;
    Right^.Next:=NewE; {Связь с предыдущим элементом}
    Right:=NewE        {Перемещение указателя хвоста очереди}
End
End;
Procedure Query.Dob; {Добавление элемента в конец очереди}
Var   Z:Data;      {Запись, добавляемая в очередь}
      NewE:Ukaz;
Begin
    Writeln('Введите фамилию и инициалы грузополучателя');
    Readln(Z.Fio);   If Z.Fio='*' Then Exit;
    Writeln('Введите адрес грузополучателя');
    Readln(Z.Adr);
    New(NewE);
    NewE^.Inf.Fio:=Z.Fio;
    NewE^.Inf.Adr:=Z.Adr;
    NewE^.Next:=nil;
    If Right=Nil Then {Если добавляется первый элемент, то}
        Left:=NewE   {инициализировать указатель головы }
    Else              {иначе }
        Right^.Next:=NewE; {связать конец очереди с элементом}
    Right:=NewE;
End;
Procedure Query.Udal; {Метод (процедура) исключения элемента}
Var Temp:Ukaz;
Begin
    Writeln('Исключается головной элемент очереди');
    Writeln('Нажмите клавишу Enter');
    Readln;
    If Left<>Nil Then {Если очередь не пустая, то}
    Begin
        Temp:=Left;   {запоминаем указатель на голову очереди}
        Left:=Left^.Next; {сдвигаем его на второй элемент}
        Dispose(Temp); {освобождаем память от головного элемента}
        If Left=Nil Then {Если удалили последний элемент, то}
            Right:=Nil;  {указатель на конец очереди равен Nil}
    End
End;
Procedure Query.Prosmotr; {Просмотр очереди}
Var i:integer; {Просмотр выполняется от головы к хвосту}
    Temp:Ukaz;

```

```

Begin
  Writeln('Очередь содержит следующие элементы:');
  Temp:=Left;
  i:=1;
  While Temp<>nil Do
  Begin
    Writeln(i, ' ',Temp^.Inf.Fio,' ',Temp^.Inf.Adr);
    Temp:=Temp^.Next;
    i:=i+1;
  End;
  Writeln('Нажмите клавишу Enter');
  Readkey;
End;

{=====секция инициализации модуля=====}
Begin
End.

{=====основная программа=====}
Program DemoQueryObject;
Uses Crt, DynQuery; {Подключение модулей}
Var   Key:Char;      {номер пункта меню программы}
      Q:Query;       {Создание экземпляра объекта}
begin
  Q.Init;             {Инициализация указателей очереди}
  Repeat
    ClrScr;           { очистка экрана }
    {-----вывод на экран пунктов меню-----}
    Writeln('1-Организация очереди');
    Writeln('2-Добавление элемента в очередь');
    Writeln('3-Удаление элемента из очереди');
    Writeln('4-Просмотр очереди');
    Writeln('5-Выход');
    Writeln('-----');
    Writeln('Нажмите клавишу от 1 до 5');
    Key:=ReadKey; {считывание кода нажатой клавиши}
    Case Key Of { вызов методов по номеру пункта меню}
      '1':Q.Org;
      '2':Q.Dob;
      '3':Q.Udal;
      '4':Q.Prosmotr;

    End
  Until Key='5' {выход из программы}
End.

```

### 10.3 Варианты заданий

Создать модуль, содержащий описание объекта, который представляет бинарное дерево в соответствии с вариантом из предыдущей лабораторной работы. Объект должен обладать возможностью добавления новых элементов, удаления существующих, поиска элемента по ключу, обхода дерева, а также выполнять дополнительные операции в соответствии с вариантом задания.

Написать программу для представления таблицы, заданной в предыдущей лабораторной работе, в виде бинарного дерева. Программа должна содержать меню, позволяющее проверить все методы объекта. Предусмотреть возможность ввода данных из файла и с клавиатуры.

#### **10.4 Порядок выполнения работы**

10.4.1. Разработать модуль, содержащий описание вспомогательных типов и объект, представляющий бинарное дерево, взяв в качестве основы программу из предыдущей лабораторной работы.

10.4.2. Разработать основную программу, в которой создается экземпляр объекта и обеспечивается вызов необходимых методов объекта.

10.4.3. Выполнить компиляцию программы. При этом в интегрированной среде в подпункте меню **Options→Directories** необходимо указать пути, где будут размещаться .tpu и .exe файлы, а также исходный текст модуля (**unit directories**), например: c:\bp\work.

10.4.4. Разработать тестовые примеры, которые предусматривают проверку корректности работы программы в разных режимах.

10.4.5. Выполнить отладку программы.

10.4.6. Исследовать работу программы при различных режимах ее использования.

#### **10.5 Содержание отчета**

Цель работы, вариант задания, текст модуля и программы с комментариями, тестовые примеры, результаты вычислений, выводы.

#### **10.6 Контрольные вопросы**

10.6.1. Что понимают под модулем в языке Turbo Pascal?

10.6.2. Назовите составные части модуля и объясните их назначение.

10.6.3. Как подключить модуль к программе?

10.6.4. Какое требование предъявляется к имени файла, содержащего модуль?

10.6.5. Что такое объект? Как его описать на языке Turbo Pascal?

10.6.6. Что понимают под методом объекта? Как описываются методы?

10.6.7. В какой части модуля обычно описывают объекты?

10.6.8. Что такое наследование? Как определить объект-потомок?

10.6.9. Как определить экземпляр объекта и присвоить значения его полям?

10.6.10. Как определяется виртуальный метод? Какие дополнительные методы необходимо при этом описать?

10.6.11. Что такое полиморфизм?

10.6.12. Как создать динамический экземпляр объекта?

## 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. Медведик, В.И. Практика программирования на языке Паскаль (задачи и решения) [Электронный ресурс] : учебное пособие / В.И. Медведик. – Электрон. дан. – Москва : ДМК Пресс, 2013. – 590 с. – Режим доступа: <https://e.lanbook.com/book/58700>.
4. 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>.
5. 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>.
6. 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