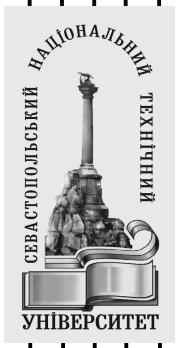
Министерство образования и науки Украины
Севастопольский национальный технический университет



ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ ПАСКАЛЬ

Методические указания к лабораторным работам
по дисциплине «Алгоритмизация и программирование»
для студентов дневной и заочной форм обучения
направления 050101 — «Компьютерные науки»

Часть 1

Севастополь
2013

УДК 004.42 (075.8)

Программирование на языке Паскаль: методические указания к лабораторным работам по дисциплине “Алгоритмизация и программирование” для студентов дневной и заочной форм обучения направления 050101 — “Компьютерные науки”, часть 1 / Сост. В.Н. Бондарев, Т.И. Сметанина.— Севастополь: Изд-во СевНТУ, 2013. — 60 с.

Методические указания предназначены для проведения лабораторных работ и обеспечения самостоятельной подготовки студентов по дисциплине “Алгоритмизация и программирование”. Целью методических указаний является обучение студентов базовым понятиям языка программирования Паскаль и основным принципам структурного программирования.

Методические указания составлены в соответствии с требованиями программы дисциплины “Алгоритмизация и программирование” для студентов направления 050101 и утверждены на заседании кафедры Информационных систем протокол №4 от 12 ноября 2013 года.

Допущено учебно-методическим центром СевНТУ в качестве методических указаний.

Рецензент: Кожаев Е.А., к.т.н., доцент кафедры кибернетики и вычислительной техники

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цели и задачи лабораторных работ

Основная цель выполнения лабораторных работ — приобретение навыков создания и отладки программ на языке Паскаль. В результате выполнения лабораторных работ студенты должны углубить знания основных теоретических положений дисциплины «Алгоритмизация и программирование», решая практические задачи на ЭВМ.

Студенты должны получить практические работы с интегрированной системой программирования, навыки разработки программ линейной, разветвляющейся и циклической структур, исследовать способы представления и обработки данных простых и регулярных типов (массивов), освоить простейшие приемы нисходящего проектирования программ.

Выбор вариантов и график выполнения лабораторных работ

При изучении раздела «Программирование на языке Паскаль» выполняется шесть лабораторных работ. Все работы выполняются в течение 4 академических часов.

В лабораторных работах студент решает заданную индивидуальным вариантом задачу. Варианты заданий приведены к каждой лабораторной работе и уточняются преподавателем. Номер варианта выбирается в соответствии с номером индивидуального плана студента. Номер варианта вычисляется как остаток от деления числа, соответствующего двум последним цифрам в номере зачетной книжки, на 30. Например, две последние цифры — 46. Тогда остаток деления 46 на 30 будет равен 16. Следовательно, студент выбирает вариант 16.

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

- изучить основные теоретические положения лабораторной работы и подготовить ответы на контрольные вопросы, используя конспект лекций и рекомендованную литературу;
- разработать алгоритм решения задачи и составить его структурную схему;
- описать схему алгоритма;
- написать программу на языке Паскаль и описать ее;
- оформить результаты первого этапа в виде заготовки отчета по лабораторной работе (**студенты-заочники** первый этап оформляют в виде контрольной работы, которую сдают на кафедру до начала зачетно-экзаменационной сессии).

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

- отладить программу;
- разработать и выполнить тестовый пример;
- подготовить и защитить отчёт по лабораторной работе.

Студенты должны выполнять и защищать работы **строго по графику**. График защиты лабораторных работ студентами дневного отделения:

- лабораторная работа №1 — 2-ая неделя семестра;
- лабораторная работа №2 — 4-ая неделя семестра;
- лабораторная работа №3 — 5-ая неделя семестра;
- лабораторная работа №4 — 6-ая неделя семестра;
- лабораторная работа №5 — 8-ая неделя семестра;
- лабораторная работа №6 — 10-ая неделя семестра.

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 постановка задачи; разработку и описание алгоритма решения задачи; текст и описание программы; результаты выполнения программы; выводы по работе; приложения. Содержание отчета указано в методических указаниях к каждой лабораторной работе.

Постановка задачи представляет изложение содержания задачи, цели её решения. На этом этапе должны быть полностью определены исходные данные, перечислены задачи по преобразованию и обработке входных данных, дана характеристика результатов, описаны входные и выходные данные.

Разработка алгоритмов решения задачи предполагает математическую формулировку задачи и описание решения задачи на уровне схем алгоритмов. Схемы алгоритмов должны быть выполнены в соответствии с требованиями ГОСТ и сопровождаться описанием.

Описание программы включает описание вычислительного процесса на языке Паскаль. Кроме текста программы, в отчёте представляется её пооператорное описание.

В приложении приводится текст программы, распечатки, полученные во всех отладочных прогонах программы с анализом ошибок, результаты решений тестового примера.

1. ЛАБОРАТОРНАЯ РАБОТА №1
«ОСНОВЫ РАБОТЫ С ОПЕРАЦИОННОЙ СИСТЕМОЙ»

1.1. Цель работы

Изучение основных приемов управления компьютером средствами операционной системы, 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*), взаимодействие между разными видами программного обеспечения (*программный интерфейс*).

По реализации интерфейса пользователя различают *неграфические* и *графические* ОС. Неграфические ОС используют *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. 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: собственно *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:

1.2.2. Основные приемы управления ОС

1.2.2.1. Управление ОС с помощью оболочки FAR

FAR (File and archive manager) — это работающая в неграфическом (текстовом) режиме программа управления файлами Windows 95/98/ME/NT/XP/7, которая обеспечивает обработку файлов с данными и имеет обширный набор дополнительных функций.

Запуск программы выполняется путем двойного щелчка левой клавишей мыши на значке FAR, расположенном на рабочем столе ОС Windows, или из Главного меню Windows (Пуск → Программы → FAR manager → FAR manager).

После запуска FAR на экране открывается окно с двумя одинаковыми панелями, разделяющими экран на две равные части. Каждая панель содержит список

имен папок и файлов. Одна из панелей является активной (та, в которой находится курсор), другая — пассивной. Переход между панелями выполняется с помощью клавиши <Tab>. Панели наглядно представляют файловую структуру дисков. Чтобы просмотреть содержимое конкретного каталога (войти в каталог), достаточно подвести к нему курсор и нажать клавишу <Enter>. Если файл является исполняемым (напомним, при этом он имеет расширение .exe, .com или .bat (командный файл)), то для его запуска достаточно подвести к требуемому файлу курсор и нажать клавишу <Enter>.

В верхней строке содержится меню настроек FAR (для входа в него используйте <F9>), а в нижней строке экрана находится строка, в которой указано соответствие между функциональными (или «горячими») клавишами и выполняемыми действиями. Над этой строкой находится командная строка (предназначена для ввода команд ОС MS DOS).

В FAR существует встроенный редактор, с его помощью можно просмотреть содержимое файла, нажав клавишу <F3>, и изменить содержимое файла, нажав клавишу <F4>. Для того, чтобы создать новый файл на диске, необходимо нажать комбинацию клавиш <Shift>+<F4>, затем набрать имя файла и нажать <Enter>. При этом осуществляется переход в режим редактирования файла.

Ниже приводятся основные клавиши, используемые при работе с FAR (таблица 1.1) и со встроенным редактором (таблица 1.2).

Таблица 1.1. — Основные команды программы FAR

Клавиши	Назначение
<F1> (Help)	Вывод на экран справочной информации
<F2>	Меню пользователя
<F3>	Просмотр файла
<F4>	Редактирование файла
<F5>	Копирование файлов
<F6>	Переименование/перемещение файлов
<F7>	Создание каталога
<F8>	Удаление файла/каталога
<F9>	Меню настроек FAR
<F10>	Выход из FAR
<Tab>	Переход между панелями
<Enter>	Вход в каталог
<Alt>+<F1> (<Alt>+<F2>)	Переход на другой диск на левой (правой) панели
<Ctrl>+<F1> (<Ctrl>+<F2>)	Выключение/включение левой (правой) панели
<Ctrl>+O	Выключение/включение обоих панелей
<Ctrl>+L	Информация о компьютере
<Alt>+<F7>	Поиск файла/каталога
<Ctrl>+E	Повтор предыдущей команды, введенной в командной строке.

Таблица 1.2. — Команды встроенного редактора FAR

Клавиши	Назначение
<F1>	Вывод на экран справочной информации
<F2>	Сохранение файла
<Shift>+↑↓	Выделение блока
<F7>	Поиск и замена строк
<Ctrl>+C	Копирование выделенного блока
<Ctrl>+V	Вставка блока
<Ctrl>+X	Удаление выделенного блока
<F10>	Выход из редактора

1.2.2.2. Управление ОС с помощью команд MS DOS

Вышеперечисленные операции можно выполнить с помощью команд ОС MS DOS, записав их в командной строке FAR. Основные команды ОС MS DOS приведены в таблице 1.3.

Таблица 1.3. — Основные команды операционной системы MS DOS

Действие	Команда	Пример	Примечание
Создать каталог	md каталог	md IVANOV	Создание в текущем каталоге каталога IVANOV
Перейти в каталог следующего уровня	cd каталог	cd LAB1	Относительно текущего каталога каталог LAB1 становится текущим
Перейти в каталог предыдущего уровня	cd . .	cd . .	Каталог IVANOV становится текущим
Перейти в корневую папку	cd \	cd \	Текущим каталогом становится корневой каталог
Удалить каталог	rd каталог	rd LAB1	Удаление пустого каталога
Создать файл	copy con: имя файла	copy con: my.txt	Создание файла my.txt. Завершение <Ctrl>+Z
Удалить файл	del имя файла	del my.txt	Удаление файла my.txt
Копировать файл	copy источник приемник	copy a.txt b.txt	Копирование a.txt. в b.txt
Переименовать файл	ren стар нов	ren a.txt c.txt	Замена a.txt на c.txt
Редактировать файл	edit файл	edit c.txt	Редактирование файла c.txt с помощью редактора MS DOS
Просмотреть содержимое каталога	dir каталог	dir IVANOV	Просмотр содержимого каталога IVANOV

1.2.2.3. Управление в 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.

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*).

1.2.2.4. 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 1.1. Окно проводника имеет две рабочие области: левую панель, называемую *панелью папок*, и правую панель, называемую *панелью содержимого*.

Навигацию по файловой структуре выполняют на левой панели Проводника, на которой изображена структура папок. Папки могут быть *развернуты* или *свернуты*, а так же *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.

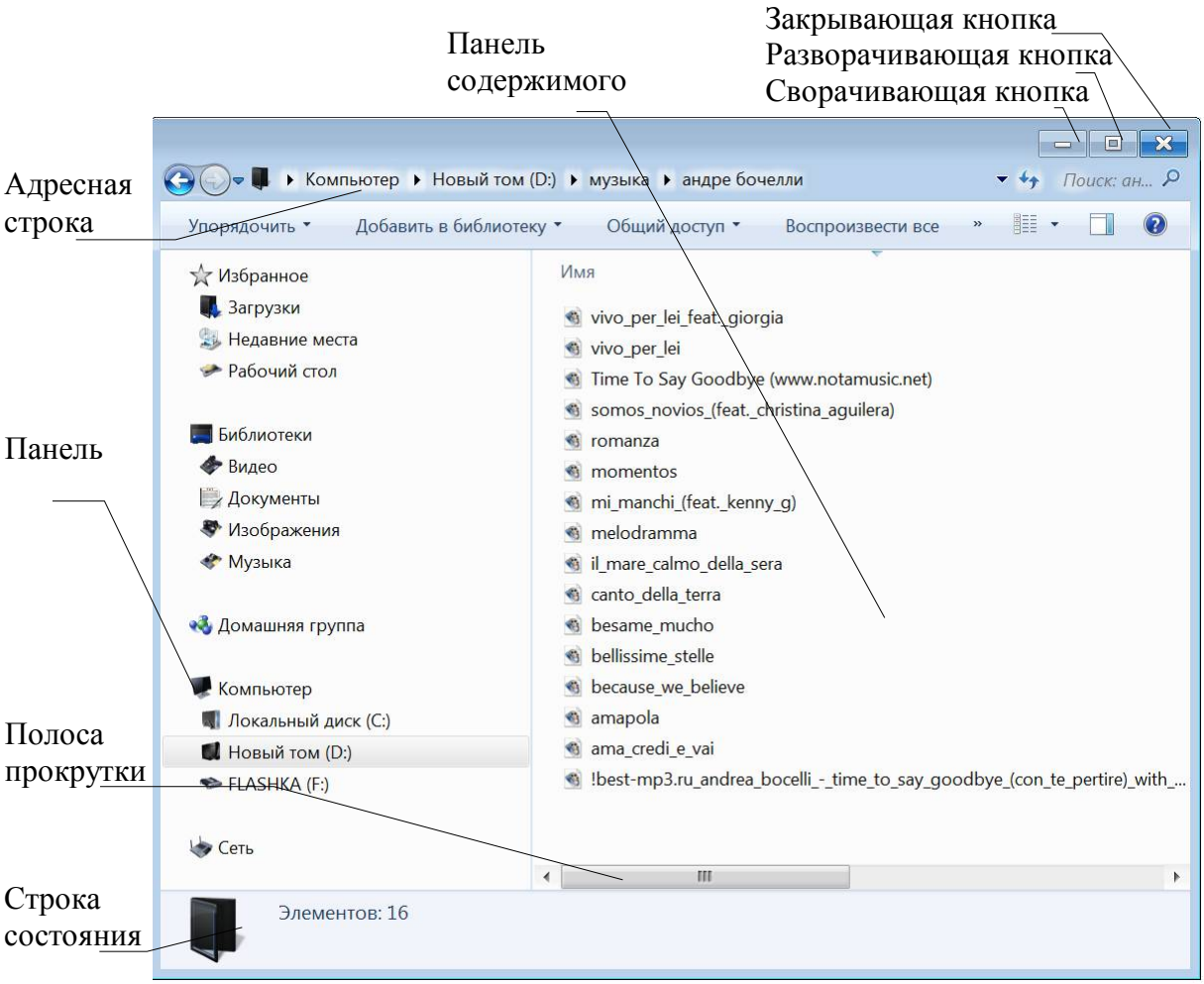


Рисунок 1.1. — Окно программы Проводник

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 — воспользо-

ваться соответствующими кнопками панели инструментов. Самый эффективный способ — использовать комбинации клавиш клавиатуры:

CTRL + C — копировать в буфер;

CTRL + X — вырезать в буфер;

CTRL + V — 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.

1.3. Варианты заданий

Большинство операций по управлению компьютером можно выполнить разными способами. В лабораторной работе необходимо отработать основные приемы работы с операционной системой (ОС) Windows 98/2000/XP/7, а также исследовать различные способы работы с файлами средствами ОС Windows, средствами ОС MS DOS и средствами оболочки FAR[1)].

Для этого студенты должны выполнить следующее:

1) отработать приемы управления с помощью мыши;

2) изучить приемы работы с объектами ОС Windows;

3) освоить работу с файловой структурой в программах Проводник и FAR.

Порядок выполнения лабораторной работы описан в разделе 4.

Студенты заочной формы обучения представляют в качестве соответствующей контрольной работы ответы на вопросы для самопроверки, приведенные ниже.

1.4. Порядок выполнения работы

1.4.1. С помощью программы Проводник ОС Windows выполнить следующие действия:

- создать папку с именем LAB1;
- создать в ней две папки с именами FOLDER1 и FOLDER2;
- создать в папке FOLDER1 текстовый файл (документ);
- дважды щелкнув на значке созданного файла, вызвать программу Блокнот, напечатать текст с ответом на один из вопросов для самопроверки, сохранить файл;
- скопировать файл в папку FOLDER2, используя специальное перетаскивание;
- внести изменения в файл с помощью программы Блокнот, сохранить его, а затем переименовать, щелкнув правой кнопкой мыши и выбрав подпункт меню «Переименовать»;
- продемонстрировать выполненное задание преподавателю и удалить файлы из папок FOLDER1 и FOLDER2, а затем удалить сами папки;
- удалить папку LAB1;

1.4.2. Выполнить указанные в п.1 действия с помощью оболочки FAR, используя функциональные клавиши.

[1)] Вместо оболочки FAR могут использоваться оболочки DOS Navigator, Norton Commander, Volkov Commander.

1.4.3. Выполнить указанные в п.1 действия, используя команды MS DOS.

1.5. Содержание отчета

Цель работы, постановка задачи, способы работы с файлами в программе Проводник, программе FAR и при помощи команд MS DOS, выводы по работе (сравнительный анализ эффективности различных способов работы с файлами).

1.6. Контрольные вопросы

1.6.1. Назначение и функции операционной системы.
1.6.2. Файлы и имена файлов.
1.6.3. Каталоги и путь доступа к файлу.
1.6.4. Основные команды MS DOS и примеры их записи.
1.6.5. Как выполнить копирование файла в оболочке FAR?
1.6.6. Назначение функциональных клавиш оболочки FAR.
1.6.7. Как выполнить перемещение блока текста в редакторе FAR?
1.6.8. Назовите и объясните основные приемы работы с мышью.
1.6.9. Объясните структуру окна программы Проводник.
1.6.10. Как выполнить создание файла и папки в программе Проводник?
1.6.11. Как выполнить копирование и перемещение файла в программе Проводник?
1.6.12. Как выполнить удаление файла в программе Проводник?
1.6.13. Как использовать буфер обмена для копирования группы файлов?

2. ЛАБОРАТОРНАЯ РАБОТА №2

«РАБОТА В ИНТЕГРИРОВАННОЙ СРЕДЕ BORLAND PASCAL (TURBO PASCAL)»

2.1. Цель работы

Приобретение практических навыков работы в интегрированной среде разработки Borland Pascal (Turbo Pascal), исследование средств, применяемых на основных этапах разработки программы.

2.2. Краткие теоретические сведения

2.2.1. ИСР, ее запуск и настройка

Интегрированная среда разработки (Integrated development Environment - IDE) объединяет текстовый редактор, компилятор, отладчик и справочную систему. Все эти составные части необходимы для успешной работы по созданию исходного текста программы, его компиляции, запуску и поиску возможных ошибок на этапе выполнения.

Обычно ИСР располагается в папке C:\BP. В каталоге BP\BIN, содержащем исполняемые файлы, имеются файлы TURBO.EXE и BP.EXE. С каждым из них связана своя ИСР, хотя внешне они очень похожи. Различие между ними заключается в том, что TURBO.EXE работает в реальном режиме MS DOS и генерирует прикладные программы MS DOS реального режима, а BP.EXE работает в защищенном

режиме MS DOS и генерирует прикладные программы MS DOS реального и защищенного режимов. В каталоге BP\BIN имеются также исполняемые файлы компиляторов, работающих в режиме командной строки (т.е. без использования возможностей интегрированной среды). Это TPC.EXE — компилятор реального режима MS DOS и BPC.EXE — компилятор защищенного режима.

В папке TP\BIN имеются и другие файлы. Файл TURBO.TPL является библиотекой стандартных модулей для приложений реального режима (это модули System, Graph, Dos, Crt). Файл TURBO.TPH содержит справочную информацию. Файл TURBO.TP представляет собой конфигурационный файл среды.

В папке BP\BGI находятся драйверы, необходимые для выполнения графических программ. Папка BP\UNITS содержит модули общего назначения. В папках BP\DOC, BP\EXAMPLES, BP\SOURCE находятся файлы документации, примеры программ.

Для запуска ИСР необходимо с помощью программы FAR или Проводник выбрать в папке TP\BIN файл BP.EXE или TURBO.EXE и нажать клавишу ENTER или дважды щелкнуть клавишей мыши. В дальнейшем подразумевается, что запущен исполняемый файл TURBO.EXE. Для выхода из среды используется комбинация клавиш ALT+X.

Окно ИСР состоит из строки меню в верхней части экрана, рабочей части в центре и строки состояния внизу. Активизировать строку меню можно, нажав клавишу F10.

Заданные по умолчанию параметры настроек ИСР могут быть изменены. К этим параметрам относятся параметры встроенного редактора и, в частности, комбинации клавиш, реализующие те или иные действия. Однако начинающему пользователю рекомендуется оставить параметры, заданные по умолчанию. Настройки ИСР хранятся в файле TURBO.TP.

Некоторые параметры ИСР могут быть изменены из меню Options. Так, окно настроек компилятора (подпункт Compiler) содержит несколько групп флажков. Первая — Code generation — относится к генерируемому коду. Группа Runtime Errors позволяет включать или выключать проверку компилятором некоторых ошибок хода выполнения программы. Группа Syntax Options позволяет задать некоторые параметры проверки синтаксиса языка. Установив в группе Numeric processing один из флажков, пользователь может дать указание компилятору использовать аппаратный математический сопроцессор или его программный эмулятор. Группа Debugging позволяет определить, следует ли генерировать необходимую для отладки программы информацию. На этапе разработки и отладки флажки данной группы должны быть установлены. В этом случае в процессе компиляции создается специальная таблица, которая устанавливает соответствие между номерами строк исходного текста программы и их адресами в объектном коде. Это позволяет быстро определить, в каком месте программы при ее выполнении произошел сбой. Последнее поле в этом окне Conditional defines. Здесь могут быть заданы символы, используемые при условной компиляции.

В окне, появляющемся при выборе команды OPTIONS→Memory Sizes, можно определить размер стека, а также минимальный и максимальный объемы динамически распределяемой памяти. Параметры компоновщика задаются в окне, от-

крываемом командой OPTIONS→Linker. Команда OPTIONS→Debugger открывает окно отладчика. С помощью этого окна можно определить вид отладочной информации, включаемой в исполняемый файл.

Каталоги хранения необходимых файлов указываются в окне, вызываемой командой OPTIONS→Directories. В поле EXE&TPU directory перечисляются каталоги, в которые помещаются результаты компиляции — исполняемые файлы и скомпилированные модули. В поле Include directories указываются каталоги, в которых помещаются файлы, включаемые в исходный текст программы. В полях Unit directories и Object directories указываются каталоги с модулями и объектными файлами, порожденными ассемблерными программами.

Для сохранения выбранных настроек необходимо выбрать команду OPTIONS→ Save TURBO.TP.

2.2.2. Набор и редактирование исходного текста

Набор и редактирование исходного текста программы выполняется средствами встроенного редактора ИСР. Если раскрыть пункт меню Edit, то можно увидеть перечень команд редактора. Этот перечень включает команды:

- отмены предыдущего действия Undo (Alt + BkSp);
- повторного выполнения ранее отмененного действия Redo;
- вырезания выделенного блока в буфер обмена Cut (Shift+Del);
- копирование выделенного блока в буфер обмена Copy (Ctrl+Ins);
- вставки блока из буфера обмена Paste (Shift+Ins);
- безвозвратного удаления выделенного блока Clear (Ctrl+Del);
- просмотра содержимого буфера обмена (Show clipboard).

В процессе подготовки файла удобно пользоваться следующими специальными клавишами и комбинациями клавиш:

F2 — сохранение файла, находящего в активном окне редактора;

F3 — открытие файла;

Shift + клавиша управления курсором — выделение блока курсором.

Для подготовки новой программы необходимо с помощью команды FILE→NEW открыть новое окно для ввода текста. В верхней части окна редактирования появляется имя, которое ИСР автоматически присваивает новому файлу — NONAME00.PAS. После набора текста программы следует нажать клавишу F2 и в открывшемся окне сохранения в поле Save file as напечатать новое имя файла (например, Lab2) и нажать кнопку «OK». Расширение .PAS добавляется автоматически.

С редактированием связано еще одно меню — Search, команды которого используются для поиска и замены фрагментов текста.

В меню File содержатся следующие основные команды:

- открытие файла Open (F3);
- сохранение файла Save (F2);
- закрытие и выход из ИСР Exit (Alt+X).

Команда закрытия файла Close находится в меню Windows, а не File. Файл можно закрыть также нажатием клавиш Alt+F3 или щелчком мыши на маленьком прямоугольнике в левом верхнем углу окна, в котором открыт файл.

По умолчанию создаваемые файлы сохраняются в папке BP\BIN. Так как эта папка предназначена для хранения исполняемых фалов ИСР, то рекомендуется хранить свои файлы в отдельной папке, например BP\WORK, и перед окончанием работы в компьютерном классе копировать их на флеш-накопитель. Для этого необходимо создать на диске в каталоге BP папку.

В случае если в ИСР загружено несколько файлов, то переключение между окнами можно выполнять с помощью команды Alt + цифра 1-9. Данная команда обеспечивает переход к окну с заданным номером. Для просмотра экрана пользователя следует нажать комбинацию клавиш Alt+F5. Изменять размеры окон, перемещать их по рабочему пространству ИСР можно из меню Windows командой Size/Move (Ctrl+F5). После нажатия клавиш Ctrl+F5 перемещение окна производится с помощью клавиш управления курсором, а для того, чтобы изменить размер активного окна, при нажатии этих клавиш следует удерживать клавишу Shift. Завершение указанных операций выполняется при нажатии Enter.

В заключение уместно привести несколько рекомендаций. Использование клавиши Esc часто помогает выйти из трудных ситуаций. Щелчок правой кнопкой мыши выводит в окно редактирования контекстное меню, содержащее небольшой, но актуальный набор команд. Это же меню вызывается нажатием клавиш Alt+F10.

2.2.3. Компиляция программы, поиск и устранение ошибок

После завершения набора программы и записи ее на диск можно приступить к компиляции, нажав клавишу F9. При этом компилятор обнаружит ошибки, которые следует исправить. Сообщение об ошибке выводится в верхней части экрана и выделяется красным цветом. Курсор при этом устанавливается в той строке программы, где обнаружена ошибка.

Исправление ошибок сводится к внесению изменений в исходный текст программы.

Компиляция программы выполняется в разных режимах. В меню Compile присутствует команда Build, которая позволяет перекомпилировать все модули, используемые программой, прежде чем начнется компиляция самой программы. Команда Make (связанная с клавишей F9) приводит к перекомпиляции только модифицированных модулей.

Во время компиляции на экране отображается окно, иллюстрирующее ход компиляции. При успешной компиляции в этом окне отображается сообщение “Compile successful”. Результат компиляции — файл, имя которого совпадает с именем исходного файла, а расширение есть .EXE. Этот файл может размещаться в оперативной памяти (в этом случае он будет потерян при выходе из ИСР) или на диске. Задать местонахождение файла можно с помощью команды Destination, которая является переключателем 2-х возможных значений Destination Memory (оперативная память) и Destination Disk (жесткий диск).

Далее в меню Compile следуют команды, позволяющие задать основной файл проекта, а также команда Information, активизация которой позволяет получить последнюю информацию о скомпилированной программе (размер программы, распределение оперативной памяти и др.)

2.2.4. Запуск программы и просмотр результатов

Запустить программу можно, нажав клавиши Ctrl+F9. Просмотреть результаты работы программы можно на экране пользователя, нажав клавиши Alt+F5. После просмотра результатов нажатие любой клавиши вернет на экран рабочее поле редактора. Меню Run содержит несколько команд. Их назначение следующее:

Program reset (Ctrl+F2) — возвращение выполняемой программы в исходное состояние;

Go to cursor (F4) — выполнение фрагмента программы от подсвеченной строки до строки, на которой находится курсор;

Trace into (F7) — запуск программы в режиме отладки с заходом внутрь процедур;

Step over (F8) — запуск программы в режиме отладки, минуя вызовы процедур.

Если в момент выполнения программе требуется параметры, то их можно задать в окне Parameters меню Run.

2.2.5. Простые приемы отладки программ

В процессе создания новой программы программисту приходится сталкиваться с несколькими видами ошибок. Во-первых, это *синтаксические ошибки*, которые обнаруживаются на этапе компиляции программы. Они связаны с неправильным использованием различных конструкций языка Паскаль. Такие ошибки обычно легко исправить. Второй вид ошибок проявляется во время выполнения программы (*ошибки времени выполнения*) и доставляют программисту гораздо больше неприятностей. Сообщение о такой ошибке имеет вид:

Run-time error <err num> at <segment: offset>.

Здесь <err num> — код ошибки, а <segment: offset> — адрес памяти, где произошла ошибка. Выяснить причину такой ошибки можно, применив соответствующие методы отладки программы.

И, наконец, наиболее тяжелые ошибки связаны с неправильным выбором алгоритма решения задачи.

Здесь рассмотрим только те методы отладки, которые доступны из ИСР. Речь пойдет о возможностях встроенного отладчика, доступ к командам которого открывает меню Debug.

Отладчик работает с исходным текстом программы. Для того чтобы использовать отладчик, следует задать для компилятора параметры генерации отладочной информации (Options→Compiler). Отладчик дает возможность пошагового выполнения программы. При этом можно просматривать значения различных переменных. Для запуска сеанса отладки выберете команду Run→Trace into или нажмите F7. При этом программа сначала компилируется, а затем начинается ее пошаговое выполнение. На каждом шаге выполняется очередная строка операторов, и происходит этот шаг при очередном нажатии F7.

Во время отладки полезно использовать окно просмотра (Watch). Если выбрать команду Debug→Add watch (Ctrl+F7), то в появившемся окне можно набрать имя переменной, текущее значение которой необходимо узнать. Для добавления в окно просмотра других переменных можно повторно выбрать команду Add watch.

Окно Watch открывается путем выбора команды Debug→Watch. Используя средства пошагового выполнения программы в сочетании с просмотром в окне Watch текущих значений ее переменных, можно провести достаточно подробный анализ работы программы.

Просмотреть значения любой переменной программы можно, установив курсор на ее идентификатор и выбрав команду Debug→Evaluate/modify (Ctrl+F4). Здесь также имеется возможность по ходу выполнения программы изменять текущее значение выбранной переменной.

Метод пошагового выполнения программы неудобен, если исходный текст большой, а программиста интересуют значения переменных только в избранных местах программы, да и то не всегда, а при выполнении определенных условий. В этом случае можно воспользоваться командой Debug→Add breakpoint, которая позволяет установить в программе «*точки останова*». Для каждой такой точки указывается номер сроки, где она устанавливается, условие, при выполнении которого программа приостановится в указанном месте, а также количество прохождений точки до ее «срабатывания». Ставшие ненужными точки останова можно удалить, воспользовавшись той же командой.

Кроме точек условного останова, есть точки безусловного останова. Такую точку можно установить/удалить, нажав в строке с курсором клавиши Ctrl+F8.

Завершить работу «зациклившейся» программы можно нажатием клавиш Ctrl+Break.

2.3. Варианты заданий

В лабораторной работе необходимо выполнить следующее:

1) запустить и настроить интегрированную среду разработки (ИСР) Borland Pascal;

2) выполнить редактирование исходного файла программы;

3) выполнить поиск и устранение ошибок компиляции программы;

4) запустить программу и просмотреть результаты;

5) выполнить отладку программы.

Порядок выполнения лабораторной работы описан в разделе 2.4.

Студенты заочной формы обучения представляют в качестве соответствующей контрольной работы ответы на контрольные вопросы для самопроверки, приведенные ниже.

2.4. Порядок выполнения работы

2.4.1. Средствами программы FAR или Проводник создать папку BP\WORK.

2.4.2. Изучить алгоритм решения задачи определения траектории полета снаряда, запущенного под углом к горизонту.

Уравнения, описывающие движение снаряда, т.е. зависимость его координат x и y от времени t, прошедшего с момента выстрела:

$$x(t) = x_0 + V_{x0}t,$$
$$y(t) = y_0 + V_{y0}t - gt^2/2.$$

Здесь введены следующие обозначения:

- x_0, y_0 — координаты начальной точки движения снаряда;
- V_{x0}, V_{y0} — начальная скорость вдоль осей x и y соответственно;
- g — ускорение свободного падения, $g = 9.81$ м/с^2.

Будем считать, что необходимо найти положение снаряда в заданные моменты времени $t_0, t_1, t_2, \ldots t_N$. При этом

$$t_{n+1} - t_n = \frac{T}{N-1} = \Delta t,$$

где T — время полета снаряда,

$$T = \frac{2V_{y0}}{g}.$$

Алгоритм решения задачи можно представить в следующем виде:

1. Ввести начальные значения скорости снаряда V_0, угла наклона начального участка траектории α и число точек N.
2. Вычислить значение $V_{x0} = V_0 \cos(\alpha)$.
3. Вычислить значение $V_{y0} = V_0 \sin(\alpha)$.
4. Присвоить $g = 9.81$ м/с^2.
5. Вычислить T.
6. Вычислить Δt.
7. Присвоить $i = 1$.
8. Присвоить $t = 0$.
9. Вычислить $x(t)$.
10. Вычислить $y(t)$.
11. Вывести координаты снаряда.
12. Присвоить $t = t + \Delta t$.
13. Присвоить $i = i + 1$.
14. Если $i < N$, то перейти к шагу 9, иначе остановить выполнение программы.

2.4.3. Создать файл с программой, вычисляющей координаты снаряда в соответствии с описанным выше алгоритмом:

```
Program coordinates;
  Uses Crt;
  Var V0, alpha, T,
     Vx0, Vy0, tc,
     dt, x, y, x0, y0: real;
```

```
    N,i:integer;
Const g=9.81;
begin
 ClrScr;  {очистка экрана}
 Writeln('Введите начальную скорость');
 Readln(V0);
 Writeln;
  Writeln('Введите угол альфа в градусах');
  Readln(alpha);
   alpha:=alpha*Pi/180;{перевод в радианы}
  Writeln;
   Writeln('Введите число точек');
   Readln(N);
    Vx0:=V0*cos(alpha);
    Vy0:=V0*sin(alpha);
     x0:=0;
     y0:=0;
     T:=2.0*Vy0/g;
     dt:=T/(N-1);
     i:=1;
     tc:=0; {текущее время}
     while i<=N do
     begin
       x:=x0+Vx0*tc;
       y:=y0+Vy0*tc-g*sqr(tc)/2;
       writeln(x,' ',y);
       i:=i+1;
       tc:=tc+dt;
       if i mod 20=0 then
          begin
            writeln('Нажмите <Enter>');
            readln;
          end
     end
end.
```

2.4.4. Сохранить файл в папке BP\WORK под именем lab2.pas. Для этого выбрать команду File→Save as или нажать клавишу F2.

2.4.5. Выполнить компиляцию программы, нажав клавишу F9.

2.4.6. Исправить синтаксические ошибки, если они будут и повторить компиляцию программы.

2.4.7. Запустить программу и просмотреть результаты ее работы.

2.4.8. Добавить в окно просмотра Watch переменные, определяющие состояние программы — i, tc, N и T. Для этого нажать Ctrl + F7, i, Enter, Ctrl + F7, tc, Enter, Ctrl + F7, N, Enter, Ctrl + F7, T, Enter.

2.4.9. Выполнить программу в пошаговом режиме. Нажать F8 (или F7). Курсор выполнения установится на служебное слово begin программы.

2.4.10. Открыть окно Watch, выбрав команду Debug→ Watch. В появившемся окне просмотреть начальные значения переменных (в режиме отладки ИСР инициализирует переменные). Зафиксировать эти значения в отчете. Следует обратить внимание на то обстоятельство, что при автоматическом выполнении программы такая инициализация не выполняется, и начальные значения переменных будут неопределенными.

2.4.11. Последовательно выполнить нажатие клавиши F8 несколько раз. При переключении программы на экран пользователя ввести значения необходимых параметров, сопровождая данное действие нажатием клавиши Enter.

2.4.12. После достижения оператора цикла (строка программы While $i <= N$ do) в окне Watch проанализировать значения переменных i, tc, N и T. Зафиксировать те значения переменных i и tc, при которых произойдет выход из цикла. Сравнить эти значения со значениями переменных N и T соответственно. Завершить работу программы.

2.4.13. Установить условную точку останова (прерывания) в строке $i := i + 1$ с условием $i = 10$. Для этого переместить курсор в строку $i := i + 1$ и выбрать команду Debug→Add breakpoint. В открывшемся окне Add breakpoint в поле Condition записать условие $i = 10$, в поле Pass count – 1. Подтвердить заполнение окна, выбрав <OK>.

2.4.14. Запустить программу в автоматическом режиме (Ctrl + F9). Теперь программа остановится в строке $i := i + 1$ при $i = 10$. Зафиксировать при этом соответствующее значение переменной tc в окне Watch. С помощью команды Debug→Evaluate/modify определить значение переменной y.

2.4.15. Удалить точку останова в строке $i := i + 1$, нажав Ctrl + F8.

2.4.16. Установить точку прерывания в строке While $i <= N$ do с условием $i = N$ и запустить программу в автоматическом режиме. После остановки программы зафиксировать значение координаты y с помощью команды Debug→Evaluate/modify.

2.5. Содержание отчета

Цель работы, описание задачи определения траектории движения снаряда, описание алгоритма решения задачи, текст программы, описание всех служебных слов программы, описание видов ошибок в программах и методов отладки программ, выводы по работе.

2.6. Контрольные вопросы

2.6.1. Для чего предназначена ИСР?

2.6.2 Объясните назначение папок ИСР Borland Pascal.

2.6.3. Объясните, как выполнить настройку ИСР с помощью меню Options.

2.6.4. Объясните команды редактора ИСР.

2.6.5. Объясните команды меню Compile.

2.6.6. Объясните команды меню Run.

2.6.7. Какие виды ошибок встречаются в программах?

2.6.8. Как добавить переменные в окно просмотра Watch?

2.6.9. Каким образом можно определить/модифицировать значение переменной в ИСР.

2.6.10. Как установить условные и безусловные точки прерываний и как их отменить?

3. ЛАБОРАТОРНАЯ РАБОТА №3

«ПРОГРАММИРОВАНИЕ ЛИНЕЙНЫХ И РАЗВЕТВЛЯЮЩИХСЯ АЛГОРИТМОВ»

3.1. 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.

3.2. 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:

$$T_F = 1{,}8\ T_C + 32,\ T_K = T_C + 273{,}15, \qquad (3.1.)$$

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 и 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 од-

ной 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:

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

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 формы 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;
      If (x >= -2) and (x < -1) then y := -x – 2;
      If (x >= -1) and (x < 1) then y := x;
      If (x >= 1) and (x < 2) then y := -x + 2;
      If x >= 2 then 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 обратите на проверку граничных условий. Данную программу можно переписать компактнее, если воспользоваться полной формой условного оператора.

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.

3.3. 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.

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

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

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

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

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

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

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

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

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

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

11) $z=\begin{cases}\ln(x)+|x|, \text{если } x <= a\\ x^{3.3}\cdot\cos(x), \text{если } a < x < b\\ x^4, \text{если } x >= b\end{cases}$

12) $z=\begin{cases}e^x-\sin(x), \text{если } x <= a\\ \cos(x-2.3), \text{если } a < x < b\\ x^{5.2}, \text{если } x >= b\end{cases}$

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

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

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

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

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

18) $z=\begin{cases}e^x/(3+\sin(x)), \text{если } x <= a\\ \tan(x)+x^{2.1}, \text{если } a < x < b\\ x^7, \text{если } x >= b\end{cases}$

19) $z=\begin{cases}e^{x^3}\cdot\sin(x), \text{если } x <= a\\ \tan(|x|)+x^2, \text{если } a < x < b\\ x^7, \text{если } x >= b\end{cases}$

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

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

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

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

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

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

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

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

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

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

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

3.4. Порядок выполнения работы

3.4.1. Разработать структурную схему алгоритма решения задачи по заданному варианту.

3.4.2.Написать программу, вычисляющую значение функции.

3.4.3.Разработать тестовые примеры.

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

3.4.5.Проверить значение функции в граничных точках.

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

3.5. Содержание отчета

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

3.6. Контрольные вопросы

3.6.1. Охарактеризуйте целый тип данных языка Паскаль.

3.6.2. Охарактеризуйте вещественный тип данных.

3.6.3. Охарактеризуйте литерный тип данных.

3.6.4. Охарактеризуйте логический тип данных.

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

3.6.6. Назовите и охарактеризуйте виды операторов присваивания в языке Паскаль.

3.6.7. Что понимают под условным выражением?

3.6.8. Укажите приоритет операций языка Паскаль и порядок вычисления выражений.

3.6.9. Перечислите действия, реализуемые при выполнении условного оператора.

3.6.10. Какие действия выполняются оператором перехода?

3.6.11. Как организовать разветвление вычислений на три ветви?

3.6.12. Приведите пример использования оператора выбора.

3.6.13. Зачем при отладке программы тестировать все ветви?

4. ЛАБОРАТОРНАЯ РАБОТА №4

«ПРОГРАММИРОВАНИЕ АЛГОРИТМОВ ЦИКЛИЧЕСКОЙ СТРУКТУРЫ»

4.1. Цель работы

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

4.2. Краткие теоретические сведения

4.2.1.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 выполнятся 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 +…+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. Такое возможно не всегда, и иногда производительность программы целиком зависит от того, как запрограммирован цикл. Инвариантные фрагменты цикла следует вычислять вне цикла.

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

$$ch(x) = 1 + \frac{x^2}{2!} + \frac{x^4}{4!} + ... + \frac{x^{2n}}{2n!} + ...$$

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

Прямое вычисления 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. Второй недостаток 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:

$$C_{n+1} = C_n T,$$

где T — некоторый множитель,

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

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

4.3. Варианты заданий

Вычислить и вывести на экран в виде таблицы значения функции, заданной с помощью ряда, на интервале от $X_{НАЧ}$ до $X_{КОН}$ 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. $\ln\frac{x+1}{x-1} = 2\sum_{n=0}^{\infty}\frac{1}{(2n+1)x^{2n+1}} = 2(\frac{1}{x}+\frac{1}{3x^3}+\frac{1}{5x^5}+...), \quad |x|>1$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

4.4. Порядок выполнения работы

4.4.1. Исследовать функции и определить область допустимых значений аргумента X, выбрать $X_{НАЧ}$, $X_{КОН}$ и шаг dX.

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

4.4.3. Написать программу и выполнить её отладку.

4.4.4. Выполнить вычисления для трёх различных значений точности.

4.4.5. Выявить зависимость количества просуммированных членов ряда от точности.

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

4.5. Содержание отчета

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

4.6. Контрольные вопросы

4.6.1. Укажите необходимые элементы любого цикла.

4.6.2. Запишите общий формат оператора цикла с параметром.

4.6.3. Приведите пример использования оператора цикла с параметром.

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

4.6.5. Запишите и объясните общий формат оператора цикла с предусловием.

4.6.6. Запишите и объясните общий формат цикла оператора с постусловием.

4.6.7. Что называют инвариантом цикла?

4.6.8. Приведите примеры использования операторов цикла с постусловием и предусловием.

4.6.9. Какие условия должны выполнятся, чтобы циклы While и Repeat-Until не превратились в бесконечные циклы?

5. ЛАБОРАТОРНАЯ РАБОТА №5

«ПРОГРАММИРОВАНИЕ АЛГОРИТМОВ ОБРАБОТКИ ОДНОМЕРНЫХ МАССИВОВ»

5.1. Цель работы

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

5.2. Краткие теоретические сведения

5.2.1. 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'].

5.2.2. Поиск в массивах

Напишем программу, которая для целочисленного массива из 100 элементов определяет, сколько положительных элементов располагается между его максимальным и минимальным элементами.

Запишем алгоритм 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; {Принимаем за максимум и}
  imin:=1; {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.

5.2.3. Сортировка массивов

Сортировка 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 5.1.

```
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 5.1. — 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;
```

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 5.2., 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 5.2. — Сортировка массива методом вставки

Ниже представлен фрагмент программы сортировки массива методом вставки.

```
var i,j:integer;
    a:array[0..n] of integer;
    x: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;
```

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 5.3.

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
Щаг 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 5.3. — 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;
```

5.3. 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 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 возрастанию, 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* целых 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.

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 возрастанию, 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.

5.4. Порядок выполнения работы

5.4.1. Разработать структурную схему алгоритма решения задачи.

5.4.2. Написать соответствующую программу.

5.4.3. Составить тестовые примеры, следуя указаниям, изложенным в разделе 5.2.

5.4.4. Выполнить отладку программы для всех тестовых примеров.

5.4.5. 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 введённых 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.

5.5. 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.

5.6. Вопросы для самопроверки

5.6.1. Приведите общую форму задания регулярного типа в языке Pascal.

5.6.2. Какие типы индексов допустимы при задании массивов?

5.6.3. Как осуществляется обращение к отдельным элементам массива?

5.6.4. Приведите примеры задания массивов для различных типов индексов.

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

5.6.6.Что такое «поиск с барьером» в массиве?

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

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

6. ЛАБОРАТОРНАЯ РАБОТА №6

«ОБРАБОТКА ДВУМЕРНЫХ МАССИВОВ С ПОМОЩЬЮ ПРОЦЕДУР И ФУНКЦИЙ»

6.1. Цель работы

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

6.2. Краткие теоретические сведения

6.2.1. 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 6.1 — 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 10x20 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;
Const nrow=10;
     ncol=20;
Var
   a:array[1..nrow,1..ncol] of integer;
   n,i,j:integer;
   s:real;
begin
  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;
   s:=s/nrow*ncol;
   Writeln('Среднее арифметическое =', s);
   readln;
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.

6.2.2. 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 не-

возможны. 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;
```

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.

6.2.3. 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 исходным 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 разработке алгоритма, поскольку алгоритм зависит от того, каким образом представлены данные.

Алгоритм работы программы. Для сортировки строк воспользуемся алгоритмом прямого выбора (см. лабораторную работу №5). 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 блок-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;
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;
{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;
Begin    {основная программа}
  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}
  Readln;
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).

6.2. 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 положительных 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 побочной 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

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

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

Вариант 8

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

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

Вариант 9

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

Найти сумму модулей элементов, расположенных ниже 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

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

Найти сумму модулей элементов, расположенных ниже главной диагонали.

Вариант 12

Уплотнить заданную матрицу, удаляя из нее строки и столбцы, заполненные нулями. Найти номер первой из строк, содержащих хотя бы один положительный элемент.

Вариант 13

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

Вариант 14

Осуществить циклический сдвиг элементов квадратной матрицы вправо на k элементов таким образом: элементы 1-й строки сдвигаются в последний столбец сверху вниз, из него — в последнюю строку справа налево, из нее — в первый столбец снизу вверх, из него — в первую строку; для остальных элементов сдвиг выполняется аналогичным образом.

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

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

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

Вариант 17

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.

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

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;

3) поиск позиций всех седловых точек матрицы, полученной в результате выполнения пункта 2;

4) зеркальную перестановку строк матрицы, полученной в результате выполнения пункта 2, относительно горизонтальной оси;

5) поиск позиций всех седловых точек матрицы, полученной в результате выполнения пункта 4;

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;

3) поиск позиций всех локальных максимумов матрицы, полученной в результате выполнения пункта 2;

4) зеркальную перестановку столбцов матрицы относительно вертикальной оси;

5) поиск позиций всех локальных максимумов матрицы, полученной в результате выполнения пункта 4.

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 $i-1 \le k \le i+1,\ j-1 \le l \le j+1$. 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;

3) поиск позиций всех локальных минимумов матрицы, полученной в результате выполнения пункта 2;

4) зеркальную перестановку строк матрицы относительно горизонтальной оси;

5) поиск позиций всех локальных минимумов матрицы, полученной в результате выполнения пункта 4.

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 $i-1 \le k \le i+1,\ j-1 \le l \le j+1$. Элемент матрицы называется локальным минимумом, если он строго меньше всех имеющихся 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:

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

6.4. Порядок выполнения работы

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

6.4.2. Разработать алгоритм решения задачи в соответствии с вариантом.

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

6.4.4. 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.

6.4.5. Выполнить отладку программы.

6.4.6. Получить результаты для всех вариантов тестовых примеров.

6.5. 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.

6.6. Контрольные вопросы

6.6.1. Приведите пример задания двумерного массива в языке Паскаль.

6.6.2. Как выполняется обращение к элементам двумерного массива?

6.6.3. Как хранится в памяти ЭВМ двумерный массив?

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

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

6.6.6. Каким образом функция возвращает значение?

6.6.7. Как записываются списки формальных и фактических параметров?

6.6.8. Как передаются параметры-значения?

6.6.9. Как передаются параметры-переменные?

6.6.10. Как передаются параметры-константы?

6.6.11. Что понимают под глобальными и локальными переменными?

6.6.12. Как передать в процедуру значения, используя механизм глобальных параметров?

6.6.13. Как передать в процедуру или функцию массив?

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Павловская Т. А. Паскаль. Программирование на языке высокого уровня : практикум / Т. А. Павловская. — СПб. : Питер, 2006. — 317 с.

2. Павловская Т. А. Паскаль. Программирование на языке высокого уровня : учеб. для вузов / Т. А. Павловская. — СПб. : Питер, 2008. — 393 с.