Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

“Севастопольский государственный университет”

Методические указания к лабораторным работам по дисциплине “Алгоритмизация и программирование” для студентов дневной и заочной форм обучения направления 09.03.02 – “Информационные системы и технологии”

Часть 1

Севастополь

2016

УДК 004.42 (075.8)

Методические указания к лабораторным работам по дисциплине “Алгоритмизация и программирование” для студентов дневной и заочной форм обучения направления 09.03.02 – “Информационные системы и технологии”, часть 1/ Сост. В.Н. Бондарев, Т.И. Сметанина, А.К. Забаштанский.— Севастополь: Изд-во СевГУ, 2016. — 76с.

Методические указания предназначены для проведения лабораторных работ и обеспечения самостоятельной подготовки студентов по дисциплине “Алгоритмизация и программирование”. Целью методических указаний является обучение студентов основным принципам структурного программирования, навыкам разработки программ на языке Паскаль с использованием текстовых и типизированных файлов, регулярных и комбинированных типов данных.

Методические указания составлены в соответствии с требованиями программы дисциплины “Алгоритмизация и программирование” для студентов направления 09.03.02 – “Информационные системы и технологии” и утверждены на заседании кафедры информационных систем протокол № 1 от 29 августа 2016 года.

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

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

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цели и задачи лабораторных работ

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

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

Выбор вариантов и график выполнения лабораторных работ

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 зачетки 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:

- проработать по конспекту и рекомендованной литературе, приведенной в конце настоящих методических указаний, основные теоретические положения лабораторной работы и подготовить ответы на контрольные вопросы;
- разработать алгоритм решения задачи и составить схему алгоритма;
- описать схему алгоритма;
- написать программу на языке Паскаль и описать ее;
- оформить результаты первого этапа в виде заготовки отчета по лабораторной работе (**студенты-заочники** первый этап выполнения лабораторных работ 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.

Выполнение лабораторной работы завершается защитой отчета. Студенты должны выполнять и защищать работы **строго по графику**. График защиты лабораторных работ студентами дневного отделения:

- лабораторная работа №1 — 1-ая неделя осеннего семестра;
- лабораторная работа №2 — 3-ая неделя осеннего семестра.
- лабораторная работа №3 — 5-ая неделя осеннего семестра;

- лабораторная работа №4 — 7-ая неделя осеннего семестра.
- лабораторная работа №5 — 10-ая неделя осеннего семестра;
- лабораторная работа №6 — 14-ая неделя осеннего 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 постановку 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 собой изложение содержания задачи и 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
«ПРОГРАММИРОВАНИЕ ЛИНЕЙНЫХ И РАЗВЕТВЛЯЮЩИХСЯ АЛГОРИТМОВ»

1.1 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.

1.2 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:

$$T_F = 1{,}8\ T_C + 32,\ T_K = T_C + 273{,}15, \qquad (1.1)$$

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

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

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

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

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

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

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

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

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

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

выведет на экран C = 2.0E+01.

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

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

Условный оператор языка Паскаль записывается в форме:

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

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

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

```
Begin
        <оператор 1>;
        <оператор 2>;
                …
        <оператор n>
End
```

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

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

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

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

Например:

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

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

Рассмотрим пример разветвляющейся программы. Пусть требуется вычислить функцию:

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

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 форме является рекомендуемой, поскольку позволяет облегчить процесс составления программы с использованием языка программирования.

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

После этого нетрудно составить текст программы:

```
Program Calculate_function;
Var   x, y: Real;
Begin
      Writeln('Введите значение аргумента');
      Readln(x);
      If x < -2 then y := 0;
      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.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

1.4 Порядок выполнения работы

1.4.1 Разработать структурную схему алгоритма решения задачи по заданному варианту.

1.4.2 Написать программу, вычисляющую значение функции.

1.4.3 Разработать тестовые примеры.

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

1.4.5 Проверить значение функции в граничных точках.

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

1.5 Содержание отчета

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

1.6 Контрольные вопросы

1.6.1 Охарактеризуйте целый тип данных языка Паскаль.

1.6.2 Охарактеризуйте вещественный тип данных.

1.6.3 Охарактеризуйте литерный тип данных.

1.6.4 Охарактеризуйте логический тип данных.

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

1.6.6 Назовите и охарактеризуйте виды операторов присваивания в языке Паскаль.

1.6.7 Что понимают под условным выражением?

1.6.8 Укажите приоритет операций языка Паскаль и порядок вычисления выражений.

1.6.9 Перечислите действия, реализуемые при выполнении условного оператора.

1.6.10 Какие действия выполняются оператором перехода?

1.6.11 Как организовать разветвление вычислений на три ветви?

1.6.12 Приведите пример использования оператора выбора.

1.6.13 Зачем при отладке программы тестировать все ветви?

2 ЛАБОРАТОРНАЯ РАБОТА №2

«ПРОГРАММИРОВАНИЕ АЛГОРИТМОВ ЦИКЛИЧЕСКОЙ СТРУКТУРЫ»

2.1 Цель работы

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

2.2 Основные сведения и методические указания

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

Оператор цикла с параметром записывается в одной из следующих форм:

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

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

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

Цикл с предусловием в языке Pascal записывается в следующем виде:

While <*условное выражение*> Do <*оператор*>;

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

Оператор цикла с постусловием записывается в виде:

Repeat
<*оператор1*>;
<*оператор2*>;
……..
<*оператор N*>
Until <*условное выражение*>;

В этом случае выход из цикла происходит, когда условное выражение получает значение True.

Следует обратить внимание на то, что оператор в цикле While ни разу не выполнится, если условное выражение с самого начала будет иметь значение False. В цикле Repeat-Until операторы 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. 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 не котором 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 вычисление ряда по приведённой выше общей формуле, когда x возводится в степень, вычисляется факториал, а затем числитель делится на знаменатель, имеет два недостатка, которые делают этот способ непригодным. Первый недостаток — большая погрешность вычислений. При возведении в степень и вычислении факториала можно получить большие числа, при делении которых друг на друга происходит потеря точности, поскольку количество значащих цифр, хранимых в ячейке памяти, ограничено. 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.

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

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

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. $\arcsin(x)=x+\sum_{n=1}^{\infty}\frac{1\cdot3\cdot\ldots\cdot(2n-1)\cdot x^{2n+1}}{2\cdot4\cdot\ldots\cdot2n\cdot(2n+1)}=x+\frac{x^3}{2\cdot3}+\frac{1\cdot3\cdot x^5}{2\cdot4\cdot5}+\frac{1\cdot3\cdot5\cdot x^7}{2\cdot4\cdot6\cdot7}+\frac{1\cdot3\cdot5\cdot7\cdot x^9}{2\cdot4\cdot6\cdot8\cdot9}+\ldots,$
$|x|<1$

20. $\arccos(x)=\frac{\pi}{2}-(x+\sum_{n=1}^{\infty}\frac{1\cdot3\cdot\ldots\cdot(2n-1)\cdot x^{2n+1}}{2\cdot4\cdot\ldots\cdot2n\cdot(2n+1)})=\frac{\pi}{2}-(x+\frac{x^3}{2\cdot3}+\frac{1\cdot3\cdot x^5}{2\cdot4\cdot5}+\frac{1\cdot3\cdot5\cdot x^7}{2\cdot4\cdot6\cdot7}+\frac{1\cdot3\cdot5\cdot7\cdot x^9}{2\cdot4\cdot6\cdot8\cdot9}+\ldots),$
$|x|<1$

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

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

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

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

2.4 Порядок выполнения работы

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

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

2.4.3 Написать программу и выполнить её отладку.

2.4.4 Выполнить вычисления для трёх различных значений точности.

2.4.5 Выявить зависимость количества просуммированных членов ряда от точности.

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

2.5 Содержание отчета

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

2.6 Контрольные вопросы

2.6.1 Укажите необходимые элементы любого цикла.

2.6.2 Запишите общий формат оператора цикла с параметром.

2.6.3 Приведите пример использования оператора цикла с параметром.

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

2.6.5 Запишите и объясните общий формат оператора цикла с предусловием.

2.6.6 Запишите и объясните общий формат цикла оператора с постусловием.

2.6.7 Что называют инвариантом цикла?

2.6.8 Приведите примеры использования операторов цикла с постусловием и предусловием.

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

3 ЛАБОРАТОРНАЯ РАБОТА №3

«ПРОГРАММИРОВАНИЕ АЛГОРИТМОВ ОБРАБОТКИ ОДНОМЕРНЫХ МАССИВОВ»

3.1 Цель работы

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

3.2 Краткие теоретические сведения

3.2.1 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'].

3.2.2 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 в общем виде.

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

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

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

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

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

```
Program Demo_Array;
Const n=10;
Var   a: array [1..n] of integer;
i, imax, imin, count: integer;
ibeg, iend: integer;
Begin
 {Ввод значений элементов массива}
        writeln ('Введите', n, 'элементов массива');
        for i:=1 to n do readln(a[i]);
imax:=1; {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.

3.2.3 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 3.1.

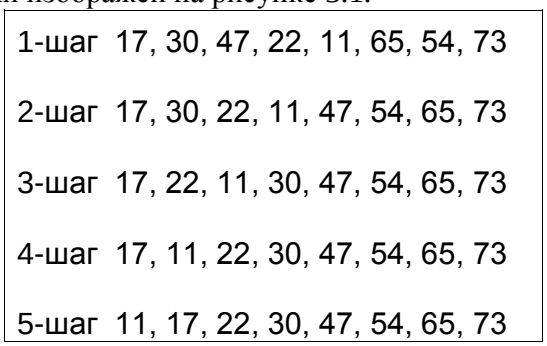

Рисунок 3.1 — Сортировка методом пузырька

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 3.2, где для каждого этапа в скобках указан очередной элемент, который включается в уже упорядоченную часть исходной последовательности.

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;
```

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 3.2 — Сортировка массива методом вставки

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 3.3.

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	б	8 |	44	33	65	23
Шаг 5	1	5	6	8	23 |	44	65	33
Шаг 6	1	5	6	8	23	33 |	65	44
Шаг 7	1	5	6	8	23	33	44 |	65
	1	5	6	8	23	33	44	65

Рисунок 3.3 — 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;
```

3.3 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.

3.4 Порядок выполнения работы

3.4.1 Разработать структурную схему алгоритма решения задачи.

3.4.2 Написать соответствующую программу.

3.4.3 Составить тестовые примеры, следуя указаниям, изложенным в разделе 3.2.

3.4.4 Выполнить отладку программы для всех тестовых примеров.

3.4.5 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.

3.5 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.

3.6 Вопросы для самопроверки

3.6.1 Приведите общую форму задания регулярного типа в языке Pascal.

3.6.2 Какие типы индексов допустимы при задании массивов?

3.6.3 Как осуществляется обращение к отдельным элементам массива?

3.6.4 Приведите примеры задания массивов для различных типов индексов.

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

3.6.6 Что такое «поиск с барьером» в массиве?

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

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

4 ЛАБОРАТОРНАЯ РАБОТА №4

«ОБРАБОТКА ДВУМЕРНЫХ МАССИВОВ С ПОМОЩЬЮ ПРОЦЕДУР И ФУНКЦИЙ»

4.1 Цель работы

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

4.2 Краткие теоретические сведения

4.2.1 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 4.1 — 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.

4.2.2 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-констан-

той. Параметр-константа не копируется во временную область памяти, что сокращает затраты времени на вызов подпрограммы, однако любые изменения в теле подпрограммы невозможны. Если необходимо допустить изменения массива, то следует передавать его как параметр-переменную.

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

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

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

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

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

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

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

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

```
<имя функции>:=<результат>;
```

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

```
…
Const n=100;
Type matrix= array[1..n] of Integer;
…
Function Symmetric (Var A: matrix): Boolean;
```

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

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.

4.2.3 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 №3). 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).

4.3 Варианты заданий

Каждый пункт нижеприведенного задания оформить в виде функции (процедуры). Все необходимые данные для функций (процедур) должны пере-

даваться 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:

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

4.4 Порядок выполнения работы

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

4.4.2 Разработать алгоритм решения задачи в соответствии с вариантом.

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

4.4.4 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.

4.4.5 Выполнить отладку программы.

4.4.6 Получить результаты для всех вариантов тестовых примеров.

4.5 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.

4.6 Контрольные вопросы

4.6.1 Приведите пример задания двумерного массива в языке Паскаль.

4.6.2 Как выполняется обращение к элементам двумерного массива?

4.6.3 Как хранится в памяти ЭВМ двумерный массив?

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

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

4.6.6 Каким образом функция возвращает значение?

4.6.7 Как записываются списки формальных и фактических параметров?

4.6.8 Как передаются параметры-значения?

4.6.9 Как передаются параметры-переменные?

4.6.10 Как передаются параметры-константы?

4.6.11 Что понимают под глобальными и локальными переменными?

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

4.6.13 Как передать в процедуру или функцию массив?

5 ЛАБОРАТОРНАЯ РАБОТА №5
“ПРОГРАММИРОВАНИЕ ОПЕРАЦИЙ НАД СТРОКАМИ И ТЕКСТОВЫМИ ФАЙЛАМИ”

5.1 Цель работы

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

5.2 Краткие теоретические сведения

5.2.1 Множества

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

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

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

```
Type Bukvi = set of char;
```

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

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

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

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

```
Var ch: char;
If ch in alphabet then   ...
```

Над множествами языка Pascal можно выполнять те же операции, что и над математическими множествами, то есть находить их объединение, пересечение, разность. Если S1 и S2 — константы или переменные множественного типа, то S1 + S2будет их объединением, S1 * S2 — пересечением, S1 - S2 — разностью. К множествам могут применяться операции отношений: = (равенство), <> (неравенство), <= (является подмножеством), >= (является надмножеством).

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

```
S :=S + [ch];
```

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

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

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

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

5.2.2 Строковый тип

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

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

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

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

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

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

```
stroka1:=’ABC’;
```

К строкам можно применять операцию *конкатенации,* которая обозначается знаком “+”. Конкатенация — это объединение строк:

```
stroka2:= stroka1 + ’DEF’;
```

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

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

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

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

```
Insert(instr, str, start);
```

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

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

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

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

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

```
Str(num, strnum);
```

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

```
Val(strnum, num, errcode);
```

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

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

5.2.3 Файловый тип

Файловый тип языка Паскаль представляет собой последовательность однотипных элементов. Число элементов в определении файлового типа в отличие от массива не фиксируется. В зависимости от типа элементов файлы

подразделяются на *текстовые и бинарные*. Текстовые файлы предназначены для хранения текстов (например, текстов программ на языке Pascal), а бинарные используются для хранения данных определенных типов.

Текстовый файл содержит последовательность символов, организованных в строки. Каждая строка файла заканчивается *меткой конец строки*, состоящей из пары управляющих символов: возврат каретки CR = #13 и перевод строки LF = #10. Заканчивается файл меткой конец файла (управляющий символ #26).

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

```
Varin_file: text;
```

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

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

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

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

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

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

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

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

Read(<файловая переменная>,<список ввода>).

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

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

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

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

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

5.2.4 Пример программы обработки строк файлов

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

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

Исходные данные и результаты.

Исходные данные:

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

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

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

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

Алгоритм решения задачи.

1 Построчно считывать текст из файла.

2 Просматривать каждую строку и искать в ней первое вхождение слова. При каждом обнаружении слова печатать номер строки и саму строку.

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

Программа и тестовые примеры.

Текст программы с комментариями приведен ниже.

```
Program  search_word_in_file;
Const
      delimiters = [' ','.',',',':',';']; {разделители слов}
      len=80;  {максимальная длина строки файла}
Var
      in_file : text;
      pattern_word,                 {введенное слово-образец}
      word_intext,                  {слово, найденное в тексте}
      stroka :string[len];          {считываемая строка файла}
      k, n, strokalength : word;
      file_name : string[20];
Begin
WriteLn( 'Введите имя текстового файла');
WriteLn;
ReadLn(file_name);
WriteLn('Введите образец для поиска');
WriteLn;
ReadLn(pattern_word);
      {Отключение автоматической проверки ошибок ввода-вывода}
{$I-}
Assign(in_file, file_name);
Reset(in_file);
If IOResult > 0 then Halt;  {Если ошибка открытия файла, то останов}

 {чтение строк до конца файла}
n := 0;
while   not Eof(in_file)   do
begin
```

```
    Inc(n); {увеличить номер строки}
    ReadLn(in_file, stroka);
       {запомнить длину текущей строки}
    strokalength := Length(stroka);
       {выделение слова из текущей строки}
    k := 1;
    word_intext := ' ';
    while   k <= strokalength do
    begin
       if (stroka[k] in delimiters) or (k=strokalength) then
       begin  {обнаружен конец слова}
          if k=strokalength then  {включение последней буквы в слово}
          word_intext := word_intext+stroka[k];
          {проверка совпадения с образцом}
          ifword_intext = pattern_word  then
          begin
             WriteLn('Слово ', pattern_word,' найдено в строке ', n:4, ':');
             WriteLn(stroka);
             Break; {слово-образец обнаружено, прервать поиск}
          end;
          word_intext := '';
       end
       else   {если не конец слова, то добавить букву в слово}
             word_intext := word_intext+stroka[k];
       Inc(k);
    end;
end;
Close(in_file);
WriteLn('Для завершения работы нажмите <Enter>');
ReadLn;
end.
```

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

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

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

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

5.3 Варианты заданий

Вариант 1

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

Вариант 2

Написать программу, которая считывает текст из файла и выводит на экран только предложения, содержащие введенное с клавиатуры слово.

Вариант 3

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

Вариант 4

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

Вариант 5

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

Вариант 6

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

Вариант 7

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

Вариант 8

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

Вариант 9

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

Вариант 10

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

Вариант 11

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

Вариант 12

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

Вариант 13

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

Вариант 14

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

Вариант 15

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

Вариант 16

Написать программу, которая считывает текст из файла и выводит на экран сначала вопросительные, а затем восклицательные предложения.

Вариант 17

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

Вариант 18

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

Вариант 19

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

Вариант 20

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

Вариант 21

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

Вариант 22

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

Вариант 23

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

Вариант 24

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

Вариант 25

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

Вариант 26

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

Вариант 27

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

Вариант 28

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

Вариант 29

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

Вариант 30

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

5.4 Порядок выполнения работы

5.4.1 Выбрать способ представления исходных данных задачи и результатов.

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

5.4.3 Разработать программу на языке Pascal.

5.4.4 Разработать тестовые примеры, следуя указаниям, изложенным в разделе 5.2.

5.4.5 Выполнить отладку программы

5.4.6 Исследовать работоспособность программы для различных режимов ее использования.

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

5.5 Содержание отчета

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

5.6 Контрольные вопросы

5.6.1 Какие типы данных используются в качестве базовых при построении множественных типов?

5.6.2 Приведите пример описания множественного типа

5.6.3 Какие операции определены над множественными типами и их приоритет?

5.6.4 Приведите примеры выполнения операций над множественными типами.

5.6.5 Напишите программу создания множества из букв латинского алфавита.

5.6.6 Чем отличается текущая длина строки от ее общей длины?

5.6.7 Как хранятся в памяти строки типа string?

5.6.8 Как инициализируется строка при ее объявлении?

5.6.9 Напишите примеры ввода и вывода строк.

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

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

5.6.12 Напишите примеры открытия и закрытия файлов.

5.6.13 Как выполняется запись данных в файл последовательного доступа, чтение из файла?

5.6.14 Как обнаружить конец файла? Как обнаружить конец строки текстового файла?

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

6 ЛАБОРАТОРНАЯ РАБОТА №6

“ПРОГРАММИРОВАНИЕ ОПЕРАЦИЙ НАД ЗАПИСЯМИ И ТИПИЗИРОВАННЫМИ ФАЙЛАМИ”

6.1 Цель работы

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

6.2 Краткие теоретические сведения

6.2.1 Записи

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

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

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

```
S.First_name:= 'ИВАНОВ';
```

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

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

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

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

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

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

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

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

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

Отформатировано: интервал Перед: 0 пт, Узор: Нет

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

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

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

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

Для переменной P типа person можно записать следующие операторы присваивания:

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

```
P.number:= 52;
```

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

6.2.2 Типизированные и нетипизированные файлы

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

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

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

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

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

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

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

```
Var Untyped_file : file;
```

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

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

Дополнительный параметр nописывает размер индивидуальной записи в файле в байтах. Если параметр nотсутствует, его значение по умолчанию принимается равным 128.

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

6.2.3 Пример программы обработки записей типизированного файла

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

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

Исходные данные и результаты.

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

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

Алгоритм решения задачи.

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

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

Программа и тестовые примеры

```
Program Demo_TypedFiles;
Uses Crt; {подключить модуль управления клавиатуры и монитора}
```

```
Type person=record    {запись о сотруднике}
            FIO:string[15];    {фамилия и инициалы}
            Year:integer;      {год рождения}
            Salary:real;       {оклад}
      End;
      Var   f: file of person;          {файл с записями о сотрудниках}
            Elem1,Elem2: person; {два элемента файла типа запись}
      {===============создание файла==================}
      Procedure Create_file;
Begin
{Установить указатель в позицию, определяемую размером файла.
Если создается новый файл, то в начало. Если записи добавляют-
ся в существующий файл, то в конец}
 Seek(f,FileSize(f));
      Writeln('Вводите сведения о сотрудниках.');
      Writeln('Для выхода вместо фамилии напечатайте символ *');
      While True Do
      Begin
            Write('Введите фамилию и инициалы (обязательно)');
            Readln(Elem1.FIO);
            IfElem1.FIO='*' Then Break; {Выход, если введена *}
            Write('Введите год рождения');
            Readln(Elem1.Year);
            Writeln('Введите оклад');
            Readln(Elem1.Salary);
            Write(f,Elem1)
      End
 End;
{==============сортировка файла==================}
Procedure Sort_file;                {Метод пузырька}
Var i, j: integer;
Begin
Seek(f,0);  {Указатель файла установить в начало}
For i:=filesize(f)-1 downto 1 do {цикл, задающий число проходов}
For j:=0 to i-1 do {сравнивать элементы от 0-го до (i-1)-го}
      Begin Seek(f,j);  {Указатель установить на j-ый элемент}
            Read(f,Elem1,Elem2);          {Прочитать две записи}
            If Elem1.Salary>Elem2.Salary then    {Сравнить оклады}
                  Begin Seek(f,j);  {Вернуть указатель в j-ую позицию}
                     Write(f,Elem2,Elem1); {Поменять местами записи}
                  End
      End
End;
{===========вывод содержимого файла==========}
```

```
Procedure Print_file;
Begin Seek(f,0);        {Указатель файла установить в начало}
      Writeln('Фамилия':15, 'Год':5, 'Зарплата':10);
      While not eof(f) do  {Просмотр всех записей до конца файла}
            Begin
            Read(f,Elem1);
            Writeln(Elem1.FIO:15, Elem1.Year:5, Elem1.Salary:10:2);
            End
      Readkey; {Ожидание нажатия клавиши Enter}
End;
{====поиск сотрудников и вычисление ср. оклада==========}
Procedure Search_persons;
Var   Found: boolean; {признак успеха поиска}
      S: string[15];          {строка с фамилией}
      N_persons: integer;     {количество найденных сотрудников}
      SummaSalary: Real;   {суммарный оклад сотрудников}
Begin N_persons:=0; SummaSalary:=0;
While True Do
Begin  Writeln('Введите фамилию или *');
       Readln(s);
       If s='*' Then Break;    {Прерывание поиска, если введена *}
       Found:=False;
       Seek(f,0);        {Указатель файла установить в начало}
{Пока не конец файла и пока не найден сотрудник с введенной
фамилией продолжать поиск}
While not eof(f) and not Found Do
Begin
Read(f,Elem1);  {Прочитать очередную запись из файла}
If (Pos(s,Elem1.FIO)<>0) and   {Есть ли в FIO фамилияs?}
  (Elem1.FIO[Length(s)+1]=' ') {Есть ли после нее пробел?}
Then Begin
{Вывод сведений о найденном сотруднике, увеличение счетчика
найденных сотрудников и вычисление суммарного оклада}
       Writeln(Elem1.FIO:15, Elem1.Year:5, Elem1.Salary:10:2);
       N_persons:=N_persons+1;
       SummaSalary:=SummaSalary+Elem1.Salary;
       Found:=True; {Сотрудник найден, прервать цикл}
       End;
End;              {Если не найден}
       If not Found Then  Writeln('Такого сотрудника нет');
End;
       If N_persons<>0 Then
       Writeln('Средний оклад=', SummaSalary/N_persons);
```

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

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

Все необходимые действия с файлом выполняются с помощью процедур, вызываемых из пунктов меню, отображаемых на экране. При заполнении файла

требуется обязательно вводить инициалы, которые должны отделяться от фамилии пробелом. Фамилия должна начинаться с первой позиции каждой строки. В дальнейшем это свойство используется в процедуре поиска сотрудников.

Рассмотрим подробнее процедуру Search_persons. Здесь цикл поиска сотрудников по фамилии организован как бесконечный цикл, выход из которого выполняется при вводе вместо фамилии символа “*”. В процедуре введена переменная Found для того, чтобы после окончания цикла поиска сотрудника в файле было известно, завершился ли он успешно. При этом проверка совпадения фамилии сотрудника производится в два этапа. Сначала с помощью функции Pos проверяется, входит ли подстрока, представляющая фамилию, в строку, содержащую фамилию и инициалы. Если входит, то функция Pos вернет ненулевое значение. После этого проверяется, есть ли непосредственно после фамилии пробел (если пробела нет, то искомая фамилия является частью другой, и эта запись нам не подходит).

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

6.3 Варианты заданий

Вариант 1

Описать структуру с именем STUDENT, содержащую следующие поля:

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

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

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

Вариант 2

Описать структуру с именем STUDENT, содержащую следующие поля:

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

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

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

Вариант 3

Описать структуру с именем STUDENT, содержащую следующие поля:
- фамилия и инициалы;
- номер группы;
- успеваемость (массив из пяти элементов).

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

Вариант 4

Описать структуру с именем AEROFLOT, содержащую следующие поля:
- название пункта назначения рейса;
- номер рейса;
- тип самолета.

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

Вариант 5

Описать структуру с именем AEROFLOT, содержащую следующие поля:
- название пункта назначения рейса;
- номер рейса;
- тип самолета.

Написать программу, выполняющую следующие действия с помощью процедур или функций:
- ввод с клавиатуры данных в файл, состоящий из элементов типа AEROFLOT; записи должны быть размещены в алфавитном порядке по названиям пунктов назначения;
- чтение данных из этого файла;
- вывод на экран пунктов назначения и номеров рейсов, обслуживаемых самолетом, тип которого введен с клавиатуры;

- если таких рейсов нет, выдать на дисплей соответствующее сообщение.

Вариант 6

Описать структуру с именем WORKER, содержащую следующие поля:

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

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

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

Вариант 7

Описать структуру с именем TRAIN, содержащую следующие поля:

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

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

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

Вариант 8

Описать структуру с именем TRAIN, содержащую следующие поля:

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

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

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

Вариант 9

Описать структуру с именем TRAIN, содержащую следующие поля:

- название пункта назначения;

- номер поезда;
- время отправления.

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

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

Вариант 10

Описать структуру с именем MARSH, содержащую следующие поля:

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

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

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

Вариант 11

Описать структуру с именем MARSH, содержащую следующие поля:

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

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

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

Вариант 12

Описать структуру с именем NOTE, содержащую следующие поля:

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

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

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

Вариант 13

Описать структуру с именем NOTE, содержащую следующие поля:

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

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

- ввод с клавиатуры данных в файл, состоящий из элементов типа NOTE; записи должны быть размещены по алфавиту;
- чтение данных из этого файла;
- вывод на экран информации о людях, чьи дни рождения приходятся на месяц, значение которого введено с клавиатуры;
- если таких нет, выдать на дисплей соответствующее сообщение.

Вариант 14

Описать структуру с именем NOTE, содержащую следующие поля:

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

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

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

Вариант 15

Описать структуру с именем ZNAK, содержащую следующие поля:

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

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

- ввод с клавиатуры данных в файл, состоящий из элементов типа ZNAK; записи должны быть упорядочены по датам рождения;
- чтение данных из этого файла;

- вывод на экран информации о человеке, чья фамилия введена с клавиатуры;
- если такого нет, выдать на дисплей соответствующее сообщение.

Вариант 16

Описать структуру с именем ZNAK, содержащую следующие поля:
- фамилия, имя;
- знак Зодиака;
- дата рождения (массив из трех чисел).

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

Вариант 17

Описать структуру с именем ZNAK, содержащую следующие поля:
- фамилия, имя;
- знак Зодиака;
- дата рождения (массив из трех чисел).

Написать программу, выполняющую следующие действия с помощью процедур или функций:
- ввод с клавиатуры данных в файл, состоящий из элементов типа ZNAK;записи должны быть упорядочены по знакам Зодиака;
- чтение данных из этого файла;
- вывод на экран информации о людях, родившихся в месяц, значение которого введено с клавиатуры;
- если таких нет, выдать на дисплей соответствующее сообщение.

Вариант 18

Описать структуру с именем PRICE, содержащую следующие поля:
- название товара;
- название магазина, в котором продается товар;
- стоимость товара в грн.

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

Вариант 19

Описать структуру с именем PRICE, содержащую следующие поля:
- название товара;

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

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

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

Вариант 20

Описать структуру с именем ORDER, содержащую следующие поля:

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

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

- ввод с клавиатуры данных в файл, состоящий из элементов типа ORDER; записи должны быть размещены в алфавитном порядке по расчетным счетам плательщиков;
- чтение данных из этого файла;
- вывод на экран информации о сумме, снятой с расчетного счета плательщика, введенного с клавиатуры;
- если такого расчетного счета нет, выдать на дисплей соответствующее сообщение.

Вариант 21

Описать структуру с именем STUDENT, содержащую следующие поля:

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

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

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

Вариант 22

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

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

Вариант 23

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

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

Вариант 24

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

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

Вариант 25

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

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

Вариант 26

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

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

Вариант 27

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

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

Вариант 28

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

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

Вариант 29

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

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

Вариант 30

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

- ввод с клавиатуры данных в файл, состоящий из структур типа STUDENT; записи должны быть упорядочены по номеру;
- чтение данных из этого файла;
- вывод на дисплей фамилий и года рождения студентов, не получивших ни одной тройки;
- если таких студентов нет, вывести соответствующее сообщение.

6.4 Порядок выполнения работы

6.4.1 Выбрать способ представления исходных данных задачи и результатов.

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

6.4.3 Разработать программу на языке Pascal.

6.4.4 Разработать тестовые примеры, следуя указаниям раздела 6.3.

6.4.5 Выполнить отладку программы.

6.4.6 Исследовать работоспособность программы в различных режимах ее использования.

6.5 Содержание отчета

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

6.6 Контрольные вопросы

6.6.1 Что понимают под комбинированным типом данных?

6.6.2 Приведите пример описания комбинированного типа.

6.6.3 Как осуществляется обращение к полям комбинированного типа?

6.6.4 Для чего применяют оператор присоединения?

6.6.5 Приведите пример использования оператора присоединения.

6.6.6 Приведите пример описания вариантных записей.

6.6.7 Как обратиться к полям вариантов?

6.6.8 Как выполняется описание пустых полей вариантов?

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

6.6.10 В чем состоит различие между типизированными и нетипизированными файлами?

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

6.6.12 Как записать и считать значения из нетипизированного файла?

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Информатика: Базовый курс / С. В. Симонович и др. — СПб.: Питер, 2003. — 640 с.

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

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

Заказ № _____ от «_____» _____2016г. Тираж _____экз.
Изд-во СевГУ