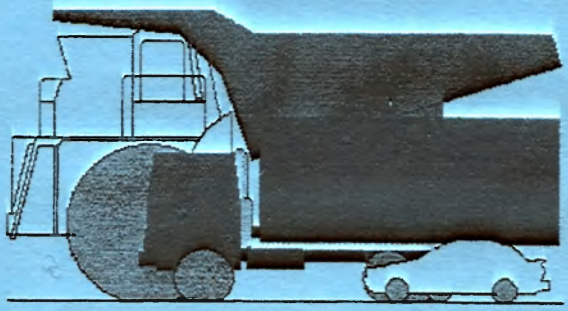
Министерство образования
Республики Беларусь

БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ
ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Кафедра «Автомобили»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ И КОНТРОЛЬНЫЕ ЗАДАНИЯ

по дисциплине «Информатика» для студентов-заочников специальности 1-37 01 02 «Автомобилестроение»



Минск 2003

Министерство образования Республики Беларусь
БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Кафедра «Автомобили»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ И КОНТРОЛЬНЫЕ ЗАДАНИЯ

по дисциплине «Информатика» для студентов-заочников специальности 1-37 01 02 «Автомобилестроение»

Минск 2003

УДК 681.3+ 681.3.06(076.5)
М54

В работе приведены содержание дисциплины «Информатика», методические указания, контрольные вопросы, рекомендуемая литература и задания для контрольных работ.

В теоретической части пособия излагаются основополагающие вопросы изучаемой дисциплины «Информатика». Основное внимание уделено рассмотрению методов и приемов решения задач с помощью алгоритмических языков PASCAL и FORTRAN.

Составители:
О.С.Руктешель, В.А.Кусяк

Рецензент В.Г.Иванов

Введение

Научно-технический прогресс, развитие практически всех областей деятельности человека тесно связаны с использованием электронно-вычислительных машин (ЭВМ) и вычислительных систем. Достижения в области вычислительной техники стимулируют развитие многих областей знаний, появление новых технологий, сложных управляющих комплексов.

ЭВМ широко используются при расчете, конструировании, изготовлении и испытаниях автомобильной техники на заводах и в научно-исследовательских организациях, а также при диагностике, техническом обслуживании и ремонте автомобилей. Умение составлять задачи для ЭВМ – необходимое требование современного технического образования.

Задача дисциплины "Информатика" – научить студентов пользоваться ЭВМ в будущей инженерной деятельности. Поэтому при изучении дисциплины основное внимание уделяется постановке задач в виде, удобном для машинного решения, составлению схем алгоритмов решения поставленных задач, выбору рациональных алгоритмов и изучению практических приемов программирования на алгоритмических языках. Для этого в данном пособии излагаются основополагающие вопросы изучаемого курса. Главное внимание сосредоточено на рассмотрении методов решения задач средствами языков программирования PASCAL и FORTRAN.

Приведенные в работе сведения помогут студентам-заочникам самостоятельно выполнить контрольную работу и освоить минимально необходимый для этого материал.

1. МЕТОДИЧЕСКИЕ УКАЗАНИЯ

Дисциплина "Информатика" изучается путем самостоятельной проработки рекомендуемой литературы.

Материал следует изучать последовательно: от темы к теме, от раздела к разделу. После проработки соответствующей темы с целью проверки степени усвоения материала необходимо ответить на предлагаемые вопросы. При этом рекомендуется составить краткий конспект. По наиболее трудным разделам дисциплины читаются лекции, проводятся лабораторные работы и консультации.

Лабораторные работы должны привить студентам практические навыки в подготовке задач и их решении на ЭВМ, а также в составлении программ на алгоритмических языках PASCAL и FORTRAN.

В процессе самостоятельной проработки дисциплины выполняются контрольные работы.

Во время сессии читаются лекции, выполняются лабораторные работы и проводится зачет. После изучения дисциплины студенты выполняют курсовую работу по индивидуальному заданию и сдают экзамен.

2. СОДЕРЖАНИЕ ДИСЦИПЛИНЫ

Приведены основные разделы, темы и содержание учебных вопросов.

Введение

Вычислительная техника и научно-технический прогресс. Предмет и задачи курса. Краткая история развития вычислительной техники.

Структурная схема и принцип работы ЭВМ

Принцип действия и структурная схема ЭВМ. Взаимодействие устройств при решении задач. Понятие алгоритма. Этапы подготовки задач к решению на ЭВМ. Системы счисления.

Программирование на алгоритмическом языке FORTRAN

Тема 1. ОБЩИЕ СВЕДЕНИЯ О СТРУКТУРЕ ПРОГРАММЫ. ЗАПУСК ПРОГРАММЫ НА ВЫПОЛНЕНИЕ. ОСНОВНЫЕ ЭЛЕМЕНТЫ ЯЗЫКА

Алфавит и объекты данных. Имена. Типы данных. Правила умолчания о типах данных. Изменения правил умолчания. Операции и выражения FPS. Встроенные элементные функции. Задание начальных значений переменных.

Тема 2. ОРГАНИЗАЦИЯ ВВОДА-ВЫВОДА ДАННЫХ

Форматный ввод-вывод. Дескрипторы данных. Дескрипторы управления. Оператор Format. Спецификация формата. Задание фор-

мата в операторах ввода-вывода (В/В). Вывод без продвижения. Согласование списка В/В и спецификации формата. Коэффициент повторения. Реверсия формата.

Тема 3. ПРОГРАММИРОВАНИЕ РАЗВЕТВЛЯЮЩИХСЯ АЛГОРИТМОВ

Условный логический оператор If. Условный арифметический оператор If. Конструкции If. Конструкция If Then Endif. Конструкция If Then Else Endif. Конструкция If Then Else If Endif.

Тема 4. ПРОГРАММИРОВАНИЕ ЦИКЛИЧЕСКИХ АЛГОРИТМОВ

Цикл "с параметром". Цикл "пока". Цикл "до". Управляющие операторы прерывания цикла. Циклические списки ввода-вывода.

Тема 5. МАССИВЫ

Объявление массивов. Инициализация массива. Размещение элементов массива в памяти ЭВМ. Доступ к элементам массива.

Тема 6. ВНЕШНИЕ ПРОЦЕДУРЫ

Структура подпрограмм. Согласование формальных и фактических параметров. Вызов подпрограммы. Структура функции. Обращение к функции.

Вопросы для самопроверки

1. Какие основные устройства входят в ЭВМ?
2. Какие виды запоминающих устройств имеет ЭВМ?
3. Какие функции выполняет процессор?
4. Чем объяснить, что числа в ЭВМ представляются в двоичной системе?
5. Какие этапы подготовки задач к решению на ЭВМ Вы знаете?
6. Что такое программный модуль и из каких элементов он состоит?
7. Что такое рабочая программа?

8. Какие основные литеры (символы) используются в языке FORTRAN?

9. Какие типы данных используются в языке FORTRAN?

10. Какие типы констант Вы знаете?

11. Сколько различают видов вещественных констант и какие они?

12. Какие логические константы Вам известны?

13. Какие переменные используются в FORTRAN-программе?

14. Сколько переменных содержит трехмерный массив, если максимальное значение первого индекса равно 5, второго – 3, третьего – 4?

15. Чем определяется тип переменной?

16. Какие стандартные функции языка FORTRAN Вы знаете?

17. Какие знаки арифметических операций используются в языке FORTRAN?

18. Какие знаки операций отношения Вы знаете?

19. Какие значения может принимать логическое выражение?

20. Перечислите, какие операторы относятся к выполняемым и какие – к невыполняемым?

21. Каков порядок операторов в основной программе?

22. Перечислите основные операторы ввода-вывода и укажите их назначение.

23. Для чего предназначен формат типа I?

24. С какой точностью будут представляться значения величин вещественного типа в формате F 6.3?

25. Каков диапазон значений величин вещественного типа стандартной длины, представимых в формате E12.4?

26. Для каких целей можно использовать форматы типа G?

27. Что собой представляет метка и для каких целей она используется?

28. Перечислите операторы присвоения.

29. Какие операторы относятся к операторам управления?

30. Что называется циклом?

31. Может ли параметр цикла принимать нулевое или отрицательное значение?

32. Что называется областью цикла?

33. Какие операторы не могут быть конечными операторами цикла?

34. Какой вычислительный процесс называется разветвляющимся?

35. Перечислите действия, которые осуществляются при выполнении условного арифметического оператора.

36. Какие действия происходят при выполнении условного логического оператора?

37. Какие операторы относятся к операторам спецификации?

38. Какие требования предъявляются к формальным параметрам операторов-функций?

39. В чем отличие модуля-функции от модуля-подпрограммы?

40. Каким образом производится обращение к модулю-подпрограмме и возвращение в основную программу?

41. Какими средствами могут быть переданы исходные данные в модуль-подпрограмму?

42. Каковы основные области использования ЭВМ?

Программирование на алгоритмическом языке PASCAL

Тема 1. КЛАССИФИКАЦИЯ ДАННЫХ ЯЗЫКА PASCAL

Алфавит и словарь языка PASCAL. Правила написания идентификаторов. Константы и переменные. Типы данных. Стандартные скалярные типы. Скалярные типы пользователя. Структурированные типы данных. Строки. Строковые выражения. Общие сведения о структуре программ. Модули. Раздел описания меток. Раздел описания констант. Раздел описания типов данных. Раздел описания переменных. Раздел описания процедур и функций. Раздел операторов. Комментарии.

Тема 2. ВЫРАЖЕНИЯ

Выражения. Операнды. Операции. Арифметические выражения и операции. Выражения и операции отношения. Логические выражения и операции. Стандартные функции.

Тема 3. ОПЕРАТОРЫ ЯЗЫКА PASCAL

Операторы языка PASCAL. Простые операторы. Структурные операторы. Условные операторы IF и CASE. Операторы ввода-вывода. Оператор чтения READ. Оператор чтения READLN. Оператор записи WRITE. Форматы оператора вывода WRITE. Оператор записи WRITELN. Файлы. Процедуры и функции обработки файлов. Процедуры. Функции. Программирование линейных и разветвляющихся алгоритмов.

Тема 4. ОПЕРАТОРЫ ОРГАНИЗАЦИИ ЦИКЛОВ

Операторы повтора. Оператор повтора FOR. Оператор повтора REPEAT. Оператор повтора WHILE.

Тема 5. МАССИВЫ

Определение массива. Действия над массивами. Действия над элементами массива. Программирование алгоритмов с использованием массивов.

Тема 6. ПОДПРОГРАММЫ

Процедуры и функции. Процедуры, определенные пользователем. Функции, определенные пользователем. Параметры.

Вопросы для самопроверки

1. Какие основные литеры (символы) используются в языке PASCAL?
2. Какие типы данных используются в языке PASCAL?
3. Приведите примеры различных способов задания дополнительных типов данных.
4. Какие способы задания массивов Вы знаете?
5. Какова структура PASCAL-программы?
6. Какие стандартные функции языка PASCAL Вам известны?
7. Какие арифметические операции и знаки операций отношения используются в языке PASCAL?
8. Каким образом производится ввод-вывод информации?
9. В чем отличие операторов условного и безусловного перехода?
10. Можно ли задать шаг 0,5 в операторе цикла с параметром?
11. Какое отличие между операторами цикла с предварительным и последующим условиями?
12. Что такое операторные скобки и для каких целей они используются?
13. Какие типы выражений и констант могут использоваться в операторе выбора CASE?
14. Нужно ли описывать метки оператора CASE в разделе описаний?

Перечень тем лабораторных работ, выполняемых при изучении программирования на алгоритмических языках FORTRAN и PASCAL

1. Константы, переменные и стандартные функции. Арифметические операции и выражения.
2. Ввод исходных данных и вывод результатов расчета.
3. Составление циклических программ.
4. Составление разветвляющихся программ.
5. Составление программ с использованием внешних процедур и функций.
6. Решение инженерной задачи на ПЭВМ.

3. ЛИТЕРАТУРА

Основная

1. Офицеров Д. В., Старых В. А. Программирование в интегрированной среде ТУРБО-ПАСКАЛЬ: Справ. пособие. – Мн.: Беларусь, 1992. – 240 с.
2. Бартеньев О. В. Современный ФОРТРАН. – 2 изд., испр. – М.: Диалог-МИФИ, 1998. – 397 с.

Дополнительная

1. Рыжиков Ю. И. Программирование на ФОРТРАНе POWER STATION для инженеров: Практическое руководство. – СПб.: Коронапринт, 1999. – 160 с.
2. Бородич Ю. С. Разработка программных систем на языке ПАСКАЛЬ: Справ. пособие. – Мн.: Выш. школа, 1992. – 143 с.

4. ЗАДАНИЕ ДЛЯ КОНТРОЛЬНОЙ РАБОТЫ

Контрольная работа ставит своей целью проверить, как студент усвоил наиболее важные разделы курса, и заключается в решении задач 1 и 2. Каждая задача выполняется на двух алгоритмических языках – FORTRAN и PASCAL.

При выполнении контрольных работ необходимо соблюдать следующие правила:

1. Для обеих задач номер варианта задания следует выбирать по последней цифре шифра зачетной книжки студента. Работа с неправильно выбранным номером варианта не засчитывается.

2. Перед решением каждой задачи нужно выписать ее условие с указанием конкретных числовых значений в соответствии с заданным вариантом.

3. Работа должна быть аккуратно оформлена и снабжена заголовком. В последнем указывается предмет, по которому выполнена работа, специальность, шифр, курс, группа, фамилия, имя и отчество студента, его домашний адрес.

Содержание задания

1. Постановка задачи.
2. Форма представления исходных данных и результатов расчета.
3. Схема алгоритма (одна для каждой задачи).
4. Тексты программ (каждая задача – на двух языках).

Задача 1

Составить программу вычисления функций X и Y, приведенных в табл. 4.1. Числовые значения переменных a, b, c приведены в табл. 4.2. Исходные данные ввести с клавиатуры, считая, что каждое из числовых значений переменных вводится после подсказки (в диалоговом режиме), причем переменные a и b – вещественного типа, переменная c – целого типа.

После ввода исходных данных напечатать заголовок: "Исходные данные" и, пропустив две строки, вывести исходные данные, снабдив их пояснениями.

Результаты вычислений x и y вывести на печать по спецификации E, отведя на каждое число по 15 позиций с тремя знаками после десятичной точки. Результат снабдить заголовком "Результат счета", отступив две строки от распечатки исходных данных.

В задаче необходимо предусмотреть проверку корректности вычислений:

1) деление на 0;

2) логарифм отрицательного числа и др.

В случае обнаружения ошибки программа должна выдать об этом сообщение и обеспечить повторный ввод измененных данных.

Таблица 4.1

Вариант	Функции	Условие
1	2	3
0	$x = \dfrac{\ln c}{e^{a}} - b$ $y = \begin{cases} \sqrt{a + \cos cx} \\ \dfrac{arctga^{c}}{b \cdot \sin a + \lg(tgb) \cdot c} \end{cases}$	если $x \geq \dfrac{a+b}{2}$ если $x < \dfrac{a+b}{2}$
1	$x = ac^{15} - \lg(b \cdot c)$ $y = \begin{cases} \ln(ax^2 + b) - \sqrt[3]{\dfrac{a \cdot c}{\sin x}} \\ \dfrac{\sqrt{(a^2 - b)} \cdot \cos c - b^{16}}{\lvert \cos(a^3 - 0{,}5) \rvert + e^{a/c}} \\ e^{\sin x} + b\sqrt{2\cos(3x - 0{,}44)^2} \end{cases}$	$x > 0$ $x = 0$ $x < 0$
2	$x = a \cdot \sin c$ $y = \begin{cases} \ln x + \sqrt[3]{\sin\sqrt{ac}} \\ tg(ax - b^2) - c \cdot e^{ax^2} \\ \dfrac{(x^3 - b)\cos(3x - 0{,}5)}{tg\, x^3 - a \cdot sign(a - b)} \end{cases}$	$\sqrt{bx^2 - 75} > a$ $\sqrt{bx^2 - 75} = a$ $\sqrt{bx^2 - 75} < a$
3	$x = a \cdot \sin c$ $y = \begin{cases} b \cdot sign(x^3 - a) - 12e^{-1{,}5x^2} \\ tg\, 4{,}75x + \dfrac{\lvert x^{-17} \rvert}{\sin(0{,}5ax)} \end{cases}$	$\sqrt{a^2 + b^2} > c$ $\sqrt{a^2 + b^2} \leq c$

1	2	3
4	$x = 1 - \lg\sqrt{a^2 - c}$ $y = \begin{cases} a\sqrt{\sin x + \cos^2(ax)} + e^{ax+b} \\ e^{\sin x} + b\dfrac{arctgx + 0{,}273}{10\,sign(a - b^2)} \\ \dfrac{(x^{-3} - b)\cdot\cos(3\sqrt[5]{x^2} - 0{,}7)}{tgx^3 - b\sin x} \end{cases}$	если $\lvert a^2 - b^2\rvert < 1{,}5x$ если $\lvert a^2 - b^2\rvert = 1{,}5x$ если $\lvert a^2 - b^2\rvert > 1{,}5x$
5	$x = \sqrt{\ln e^{\sin c} + ab^2}$ $y = \begin{cases} \dfrac{ax - e^x + b^2}{\lg(2x + 3)} + \cos(4x - 0{,}2) \\ e^{-x^2} + \dfrac{ac}{tg\sqrt{\ln(1{,}5x)}} \\ a + \sin(2x - 0{,}16) + \sqrt{a + bx} \end{cases}$	если $x < \sqrt{a^2 + b}$ если $x = \sqrt{a^2 + b}$ если $x > \sqrt{a^2 + b}$
6	$x = a + \sqrt{c\cdot tge^b}$ $y = \begin{cases} \lg\left\lvert\dfrac{x^{-7}}{ax - 0{,}35}\right\rvert - e^{2x^3} \\ \cos x + \dfrac{15{,}5\cdot sign(ax^2 - 0{,}56)}{\ln(tg\,1{,}5c) - 0{,}1x} \end{cases}$	$c^2 x < b$ $c^2 x \geq b$
7	$x = tg\sqrt{\ln(ac^3) - b}$ $y = \begin{cases} b\cdot ax^{\sin c} + \cos\left(\dfrac{a}{c}\right)\cdot sign(x - 10) \\ \lg\sqrt{\lvert\sin 0{,}5x\rvert} + ae^{0{,}7\cos 12x} \\ tg4{,}5x + \dfrac{x^{-2{,}5}}{\sin 0{,}5x} + c \end{cases}$	$ax^2 > c$ $ax^2 < c$ $ax^2 = c$

1	2	3
8	$x = (a + \sin c)^b$ $y = \begin{cases} a + b \cdot sign\, x - \cos\sqrt{tg\, x} \\ \sin\left(e^a \sqrt[4]{tg^3 c}\right) \\ \lg\left(\dfrac{a^2 - b^2}{a + \sin x}\right) - e^{12x} \end{cases}$	$\cos x > \sqrt{a^2 - b^2}$ $\cos x = \sqrt{a^2 - b^2}$ $\cos x < \sqrt{a^2 - b^2}$
9	$x = a^3 - \dfrac{b}{\sin c} + c^{ab}$ $y = \begin{cases} 4\, arctg\,(2x - 0{,}5) + \dfrac{ax + 8}{\sqrt{b^2 - x}} \\ \ln x^2 - e^{ax+b} + \lg\lvert a - b \rvert \end{cases}$	$x \le 15$ $x > 15$

Таблица 4.2

Вариант	Числовые значения переменных		
	a	*b*	*c*
0	-0,37497	$-0{,}753 \cdot 10^{12}$	97845
1	29,5760	$936{,}7 \cdot 10^{-3}$	170
2	0,00495	37897	4300
3	-1017,10	$8{,}797 \cdot 10^{5}$	212
4	719,460	$-0{,}0047 \cdot 10^{7}$	-7752
5	9878,20	$9997{,}2 \cdot 10^{-6}$	-1222
6	-0,00025	49738	1000
7	475,000	$-757{,}24 \cdot 10^{-5}$	15
8	9378,00	$17{,}293 \cdot 10^{4}$	-1212
9	27457,0	$12{,}377 \cdot 10^{-8}$	-999

Задача 2

Используя оператор цикла, составить программу, выполняя следующие условия:

1. Исходные данные вводятся с клавиатуры в диалоговом режиме.
2. Результаты расчета выводятся на монитор и в файл.
3. Результаты счета выводятся по вещественному формату. Перед выводом результата напечатать: "Результат", поместив его на 30-й позиции строки. Отступив строку, напечатать числовые значения результатов счета.

Вариант 0

Найти произведение элементов двумерного массива A(10,10) (квадратной матрицы), лежащих на главной диагонали (т.е. таких элементов, у которых номера строки и столбца одинаковы).

Вариант 1

Найти сумму элементов двумерного массива BIM(15,15) (квадратной матрицы), лежащих на главной диагонали (т.е. таких элементов, у которых номера строки и столбца одинаковы).

Вариант 2

Найти произведение элементов четных строк двумерного массива DOG(9,9) (квадратной матрицы), лежащих на главной диагонали (т.е. таких элементов, у которых номера строки и столбца одинаковы). На печать вывести все произведения поочередно.

Вариант 3

Найти сумму элементов нечетных строк двумерного массива STY(12,12), лежащих на главной диагонали. На печать вывести все суммы поочередно.

Вариант 4

Вычислить среднее арифметическое элементов массива A(10,12).

Вариант 5

Найти наибольший элемент массива M(5,10) и номер строки и столбца, в которых он находится.

Вариант 6

Найти наименьший элемент массива N(15,10) и номер строки и столбца, в которых он находится.

Вариант 7

Заменить отрицательные элементы массива F(7,10) на нулевые.

Вариант 8

Подсчитать количество положительных элементов массива P(6,6).

Вариант 9

Подсчитать количество отрицательных элементов массива T(6,6).

5. ОБЩИЕ СВЕДЕНИЯ ОБ ЭВМ

Обычно персональные компьютеры IBM PC состоят из 3-х основных частей (блоков):

1) системного блока;

2) клавиатуры, позволяющей вводить символы в компьютер;

3) монитора (дисплея) – для изображения текстовой и графической информации.

В системном блоке располагаются все основные узлы компьютера:

1) электронные схемы, управляющие работой компьютера (микропроцессор, оперативная память, контроллеры устройств и т.д.);

2) блок питания, преобразующий электропитание сети в постоянный ток низкого напряжения, подаваемый на электронные схемы компьютера;

3) накопители (или дисководы) для гибких магнитных дисков, используемые для чтения и записи на гибкие магнитные диски (дискеты);

4) накопитель на жестком магнитном диске, предназначенный для чтения и записи на жесткий несъемный магнитный диск (винчестер).

К системному блоку компьютера IBM PC можно подключать различные устройства ввода-вывода информации, расширяя тем самым его функциональные возможности. Кроме монитора и клавиатуры, такими устройствами являются:

1) принтер – для вывода на печать текстовой и графической информации;

2) мышь – устройство, облегчающее ввод информации в компьютер;

3) джойстик – манипулятор в виде укрепленной на шарнире ручки с кнопкой (применяется, в основном, для компьютерных игр);

4) сканер – для сканирования и последующего ввода текстовой или графической информации в компьютер.

6. ОСНОВЫ АЛГОРИТМИЧЕСКОГО ЯЗЫКА FORTRAN

6.1. Общие сведения о программе. Основные элементы языка

ФОРТРАН-программа – последовательность операторов языка программирования.

Операторы делятся на:

1) ***невыполняемые***, описывающие элементы программы (данные и программные компоненты);

2) ***выполняемые***, описывающие действия над элементами программы согласно разработанному пользователем алгоритму.

Структура программы на алгоритмическом языке FORTRAN POWER STATION (FPS) в простейшем виде может быть представлена так:

```
Program имя-программы                    Заголовок программы
Раздел объявления типов используемых переменных
Раздел выполняемых операторов
end                                      Завершение программы
```

Примечание. Заголовок программы может быть опущен.

Некоторые правила записи текста программы:

1) длина строки текста равна 132 символам (72 символам в фиксированном формате);

2) позиции 1…5 отведены под метку оператора, состоящую из набора от одной до пяти десятичных цифр; если в первую позицию заносится литера С, то содержание строки не транслируется (рассматривается как комментарий);

3) если в 6-й позиции строки проставлена любая литера из алфавита FORTRAN, кроме нуля и пробела, последующая строка рассматривается как строка продолжения;

4) запись оператора начинается с 7-й позиции строки; запись двух операторов в одной строке не допускается;

5) в операторе FPS может быть до 7200 символов; число строк продолжения при свободном формате не может быть более 54;

6) любые символы, расположенные между восклицательным знаком и концом строки, рассматриваются как комментарий.

6.1.1. Запуск программы

Любая программа рассматривается в FPS как проект. Для запуска новой программы необходимо, прежде всего, создать такой проект. Для этого в главном меню находим команду File, нажимаем клавишу "Enter", выбираем в появившемся падающем меню опцию New, нажимаем клавишу "Enter". Далее набираем текст программы. Затем записываем его на диск: File – Save; выбираем на диске директорию для записи файла и задаем имя файла – OK.

Сохраняемые на диске файлы с исходным текстом могут иметь расширения: F90, F и FOR. По умолчанию FPS считает, что файлы с расширением F90 написаны в свободном формате, а с расширениями F и For – в фиксированном.

Пример записи файла с именем *mur* на диск D в каталог Users, в подкаталог 301110 (соответствует номеру группы студента): File – Save – D:\users\301110\mur.for – OK.

Для запуска программы необходимо войти в меню команды Compile и последовательно выбрать следующие опции:

1) Build-Compile – компиляция проекта и исправление обнаруженных ошибок, появляющихся в специальном окне;

2) Build-Build – создание выполняемого exe-файла;

3) Build-Execute – запуск созданного exe-файла.

Для выхода из рабочего окна нажимаем любую клавишу, – например, ESC или Enter.

6.1.2. Алфавит и объекты данных

Алфавит языка FORTRAN делится на три группы символов: цифры, буквы и специальные литеры.

Цифра – это одна из десяти литер: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

Буква – это одна из 26 литер (a-z) латинского алфавита.

Специальная литера – это одна из 15 литер: = (равно), + (плюс), - (минус), * (звездочка), / (наклонная черта или слэш), \ (обратный слэш), ( ) (соответственно левая и правая скобки), , (запятая), . (точка), $ (денежный знак или знак доллара), ' (апостроф), & (коммерческое "и"), ∪ (пробел), _ (символ подчеркивания).

Литера пробела – это отсутствие какого-либо графического изображения в данной позиции.

Программа выполняет обработку данных. Данные представлены в программе в виде ***переменных*** и ***констант***. Объекты данных (переменные и константы) различаются именами, типами и другими свойствами.

Имена присваиваются переменным, константам, программным компонентам.

Имя – это последовательность латинских букв, цифр, символа $ или символа подчеркивания "_", начинающаяся с буквы или $ и продолжающаяся далее в любой комбинации. Имя не должно содержать более 31 символа. Регистр букв – незначащий.

Имена делятся на:

1) глобальные (имя головной программы, встроенной процедуры);

2) локальные (имя переменной, константы).

Типы данных разделяются на ***встроенные*** и ***производные***, создаваемые пользователем.

Встроенные типы данных:

1) целый – Integer, Byte, Integer*1, Integer*2, Integer*4;

2) вещественный – Real, Real*4, Real*8, Double Precision;

3) комплексный – Complex, Complex*4, Complex*8., Double Complex;

4) логический – Logical, Logical*1, Logical*2, Logical*4;

5) символьный – Character*n (где n – длина символьной строки, $1 < n < 32767$).

Объекты данных логического типа могут принимать значение .True. (истина) или .False. (ложь).

В FPS каждый тип данных характеризуется параметром ***разновидности типа*** (значение после звездочки). Для числовых типов данных этот параметр описывает точность и диапазон изменения значений, а также указывает число отводимых под тип байт.

Каждый тип данных имеет стандартную, задаваемую по умолчанию разновидность (звездочка и цифра отсутствуют). Встроенный тип с задаваемой по умолчанию разновидностью называется ***стандартным типом данных.***

Для примера приведем диапазон изменения значений некоторых типов данных в зависимости от их разновидности:

Integer*1	от	-128	до	+127
Integer*2	от	-32768	до	+32767
Integer*4	от	-2147483648	до	+2147483647
Integer	от	-2147483648	до	+2147483647
Byte	от	-128	до	+127
Real*4	от	-3.4028235E+38	до	-1.1754944E+38; число 0;
	от	+1.1754944E-38	до	+3.4028235E+38

(дробная часть может иметь до шести десятичных знаков)

Real – то же, что и Real*4.

Правила умолчания о типах данных. В FPS допускается не объявлять объекты данных целого и вещественного типов. При этом тип данных объекта будет установлен в соответствии с существующими ***правилами умолчания***: объекты данных, имена которых начинаются с букв i, j, k, l, m, n, имеют по умолчанию ***стандартный целый тип*** (Integer). Все остальные объекты (начинающиеся с других букв) имеют по умолчанию ***стандартный вещественный тип*** (Real).

Изменения правил умолчания вносятся с помощью ***оператора Implicit.***

Синтаксис оператора:

Implicit *тип* (*буквы*) [, *тип* (*буквы*),....]

где *тип* – один из встроенных или производных типов данных;

буквы – список одинарных букв или диапазона букв (указывается с помощью тире). Одинарные буквы и диапазоны в списке разделяются запятыми.

Примечание. Квадратные скобки при записи оператора не пишутся. В данном случае они применены, чтобы указать на то, что информация, приведенная в них, – необязательна, т.е. может быть опущена.

Пример:

```
Implicit real (k, m), integer*2(a, t-z), character*6 (b-d)
```

После такого задания все объекты, имена которых начинаются с букв *k* и *m*, будут по умолчанию иметь стандартный вещественный тип Real; объекты, имена которых начинаются с букв *a* или с букв из диапазона *t-z*, будут по умолчанию иметь тип Integer*2; объекты, имена которых начинаются с букв из диапазона *b-d*, будут по умолчанию иметь тип Character*6.

6.1.3. Операции и выражения FPS

Операции FORTRAN разделяются на ***встроенные*** и ***перегружаемые*** (задаваемые программистом).

Встроенные операции бывают:

1) арифметические;
2) символьная операция конкатенации;
3) операции отношения;
4) логические.

Ниже приведен синтаксис ***арифметических операций*** в порядке убывания их приоритета:

** – возведение в степень;

*, / – умножение, деление;

унарные + и – (например: 20/(-5) = -4);

(+, -) – сложение, вычитание.

Операции применяются для создания выражений, которые затем используются в операторах FORTRAN.

Результатом *арифметического выражения* может быть величина целого, вещественного или комплексного типов либо массив одного из этих типов.

Операндами арифметического выражения могут быть:

1) арифметические константы;
2) скалярные числовые переменные;
3) числовые массивы и их сечения;
4) вызовы функций целого, вещественного или комплексного типов.

Символьная операция FPS конкатенации – единственная символьная операция. Ее результатом является объединение строк – операндов символьного выражения. Длина результирующей строки равна сумме длин строк-операндов.

Операндами символьного выражения могут быть:

1) символьные константы и переменные;
2) символьные массивы, их сечения и элементы;

3) вызовы символьных функций;
4) символьные компоненты производных типов.

Пример:

```
.....
Character*20 a, b, c, d, e
data a /'masha'/, b /'piter'/, c /'+'/, d /'='/, e /'love'/
print*, a // c // b // d // e          ! На экране появится сообщение:
                                       ! masha + piter = love
```

6.1.4. Встроенные элементные функции

Abs(*a*) – абсолютная величина целого, вещественного или комплексного аргумента.

Если *a* – целого типа, то и результат – целого типа; в остальных случаях результат будет вещественным. Для комплексного аргумента a = x + y*i* модуль вычисляется: $|a| = \sqrt{x^2 + y^2}$.

Пример:

```
complex*4 z /3.0, 4.0/
write (*,*) abs(z)              ! Результат  5
```

Mod(*a*, *b*) – возвращает остаток от деления *a* на *b*. Параметры *a* и *b* должны быть либо оба – вещественные, либо оба – целые.

Пример:

```
write (*,*) mod(5, 4), mod(5.1, 4.0)     ! Результат  1   1.1
```

Exp(*x*) – возвращает e^x для вещественного или комплексного *x*.

Log(*x*) – возвращает значение натурального логарифма для вещественного или комплексного аргумента *x*, причем $x > 0$.

Log10(*x*) – возвращает значение десятичного логарифма вещественного аргумента ($x > 0$).

Sqrt(*x*) – возвращает квадратный корень для вещественного или комплексного аргумента *x*.

Sin(*x*) – возвращает синус вещественного или комплексного аргумента *x*, который интерпретируется как значение в радианах.

Cos(*x*) - возвращает косинус вещественного или комплексного аргумента *x*, который интерпретируется как значение в радианах.

6.1.5. Задание начальных значений переменных

Начальное значение переменной может быть задано:
1. При помощи оператора присваивания.
Синтаксис:
имя переменной = выражение
Пример:

```
a = 5.10
b = -7
c = b+5*a
```

2. В разделе описаний с использованием слэша (наклонной черты):

```
real b /14.2/
integer*4 c /25/
```

3. С помощью оператора Data.
Синтаксис:
Data *список имен /список значений/*
Пример:

```
Program kuk
real a, cmax
data a /7.1/, cmax /12.9/
или другой вариант записи оператора Data
.....
data a, cmax /7.1, 12.9/
```

4. С клавиатуры в диалоговом режиме:

```
print *, 'input a, b, c'  ! На экране дисплея на черном фоне поя-
                          ! вится сообщение-запрос о вводе исход-
                          ! ных данных a, b, c.
read (*,*) a, b, c        ! Курсор мигает на черном фоне до тех пор,
                          ! пока не будет произведен ввод числовых
                          !  данных a, b, c.
```

5. С внешнего файла посредством оператора Open.
Оператор Open создает устройство ввода-вывода с номером *u* и подсоединяет к нему внешний файл. При успешном подсоединении

файл считается открытым, и к нему может быть обеспечен доступ других работающих с файлами операторов FPS.

Синтаксис:

Open ([Unit =] u [, File = file][, Status = status]),

где u – выражение стандартного целого типа, задающее номер устройства, к которому подсоединяется файл file;

file – символьное выражение, задающее имя файла, подсоединяемого к устройству с номером *u*;

status – символьное выражение, которое может принимать значения 'Old', 'New' и др. Если выбран статус 'Old', файл должен уже существовать. Если он не существует, он будет создан, в противном случае возникнет ошибка ввода-вывода.

Пример:

Внешний вид файла исходных данных с именем kot.dat:

```
4.5  -6.89  57.9
56  -48
```

Фрагмент текста головной программы:

```
.....
Real a, b, c
integer kk, bb
open (unit = 5, file = 'd:\users\301110\ kot.dat', status = 'old')
read (5, *) a, b, c
read (5, *) kk, bb
```

6.2. Организация ввода-вывода данных

Стандартные средства FORTRAN поддерживают 4 вида ввода-вывода (В/В) данных:

1) под управлением списка В/В;
2) форматный;
3) неформатный;
4) двоичный.

Первые два вида В/В предназначены для преобразования текстовой информации во внутреннее представление и наоборот соответственно при вводе и выводе.

Если используется В/В, управляемый списком, преобразование выполняется в соответствии с установленными по умолчанию правилами. Управляемые списком операторы ввода с клавиатуры и вывода на экран имеют вид:

Read (*,*) список ввода
Read *, список ввода
Write (*,*) список вывода
Print *, список вывода

Список ввода – список, в котором перечисляются разделенные запятыми имена переменных, значения которых необходимо ввести.

Список вывода – список, устанавливающий величины, которые надо вывести.

Список вывода может содержать выражения любого типа и вида (арифметические, логические, константные), список ввода – только переменные.

Последняя или единственная звездочка операторов означает, что В/В управляется списком. В операторах, содержащих две заключенные в скобки и разделенные запятой звездочки, первая задает устройство В/В (клавиатуру и экран).

Пример:

Требуется ввести 5 значений:
4.5 -6.89 57.9
56 -48
и вывести на экран их сумму.
Фрагмент текста головной программы (первый вариант):

```
.....
Real aw, bd, c
integer k, bb
print *, ' vvedite znacheniya a, b, c'  ! На экране появится сообщение:
                                        ! vvedite znacheniya a, b, c
read (*, *) aw, bd, c                   ! Потребуется ввести с клавиа-
                                        ! туры 3 значения
print *, ' vvedite znacheniya k, bb'    ! На экране появится сообщение:
                                        ! vvedite znacheniya k, bb
read *, k, bb                           ! Потребуется ввести с клавиа-
                                        ! туры 2 значения
```

```
print *, 'rezult of calculation REZ=', aw + bd + c + k + bb  ! Контрольный
                                                             ! вывод на экран
……..
```

Фрагмент текста головной программы (второй вариант):

```
……..
Real aw, bd, c, rez                          ! Переменная rez – резуль-
                                             ! тат суммы
integer k, bb
print *, ' vvedite znacheniya a, b, c'       ! На экране появится сооб-
                                             ! щение: vvedite znacheniya
                                             ! a, b, c
read (*, *) aw, bd, c                        ! Потребуется ввести с кла-
                                             ! виатуры 3 значения
print *, ' vvedite znacheniya k, bb'         ! На экране появится сооб-
                                             ! щение: vvedite znacheniya
                                             ! k, bb
read *,  k, bb                               ! Потребуется ввести с кла-
                                             ! виатуры 2 значения
rez = aw + bd + c + k + bb
write (*,*) 'rezult of calculation REZ=',    ! Контрольный вывод на
rez                                          ! экран
……..
```

6.2.1. Форматный ввод-вывод

В некоторых задачах для выполнения операции ввода-вывода данных ЭВМ должна иметь сведения о том, какие данные надо вводить или выводить и каков формат этих данных (вещественные числа или целые, цифровые данные или буквенные, сколько разрядов занимают и т.д.). В этом случае перевод данных из внутреннего представления в текстовое задается ***дескрипторами преобразований (ДП)***, которые подразделяются на:

1) дескрипторы данных (ДД);
2) дескрипторы управления (ДУ);
3) строки символов.

Дескрипторы данных (ДД) определяют размер и форму полей В/В, в которых размещается текстовое представление данных. Перечень наиболее часто употребляемых ДД приведен в табл. 6.1.

Таблица 6.1

Дескриптор	Тип аргумента	Внешнее представление
Iw[.m]	Целый	Целое число
Bw[.m]	-	Двоичное представление
Ow[.m]	-	Восьмеричное представление
Zw[.m]	Любой	Шестнадцатеричное представление
Fw.d	Вещественный	Вещественное число в F-форме
Ew.d[Ee]	-	Вещественное число в E-форме
Dw.d	-	Вещественное число двойной точности
Lw	Логический	T и F, .T и .F, .True. и .False.
A[w]	Символьный	Строка символов
Gw.d[Ee]	Любой	Зависит от типа данных

В таблице использованы следующие обозначения:

w – длина поля, отведенного под представление элемента В/В;

m – число ведущих нулей ($m \le w$);

d – число цифр после десятичной точки ($d < w$).

Некоторые правила преобразования числовых данных:

1) при выводе символы выравниваются по правой границе поля; при необходимости добавляются ведущие пробелы;

2) если при выводе число полученных в результате преобразований символов превосходит длину поля w, все поле заполняется звездочками;

3) если вещественное число содержит больше цифр после десятичной точки, чем предусмотрено параметром d, происходит округление значения числа до d знаков после десятичной точки.

Дескрипторы управления необходимы:

1) для управления позиций В/В (преобразования nX, T, TL,TR);

2) для внесения в запись дополнительной информации (преобразование апострофа и холлерита);

3) для масштабирования данных и других функций управления.

Наиболее часто употребляются дескрипторы управления позициями ввода-вывода:

Tn - задает абсолютную табуляцию: передача следующего символа будет выполняться, начиная с позиции n (отсчет позиций выполняется от начала записи);

TRn, TLn – соответственно правая и левая табуляции;

nX - перемещает позицию В/В на n символов вперед.

6.2.2. Оператор Format. Спецификация формата

Разделенный запятыми список ДП заключается в скобки и указывается в ***спецификации формата***.

Спецификация может быть задана:

1) как встроенная в оператор В/В символьная строка;

2) как отдельный оператор Format, на который операторы В/В ссылаются при помощи метки:

метка Format (список ДП)

Пример:

Вывод значений двух переменных на экран, причем переменная a – вещественного типа, b – целого.

Фрагмент текста головной программы:

```
    ......
    write (*, '(1x, f8.4, i7)') a, b  ! Вывод начнется со второй позиции; под пе-
                                      ! ременную a отведено поле длиной 8 симво-
                                      ! лов (4 из них – под дробную часть), под пере-
                                      ! менную b – длиной 7 символов
```

или

```
    ......
    write (*,1) a,b
1   format (1x, f8.4, i7))
```

Задание формата в операторах ввода-вывода. При форматном вводе-выводе операторы В/В содержат ссылку на используемый *формат*. Эту ссылку можно задать 4-мя способами:

1) в виде метки, указывающей на оператор Format:

```
    ......
    write (*, 2) 'a=', a, 'b=', b
2   format (1x, a, f5.3, 1x, a3, i7)
```

или

```
    ......
    write (*, fmt = 2) 'a=', a, 'b=', b
2   format (1x, a, f5.3, 1x, a3, i7)
```

2) в виде встроенного в оператор В/В символьного выражения:

```
    ......
    write (*, '(t2, a, f5.3, 1x, a3, i7)') 'a=', a, 'b=', b
```

или

```
    ......
    write (*, fmt='(t2, a, f5.3, 1x, a3, i7)') 'a=', a, 'b=', b
```

3) в виде имени именного списка В/В:

```
    .....
    real k /8./, massiv_A(5) /1., -6.0, 4., -3.1, .1/  ! massiv_A – имя одномерного масси-
    namelist /gus/ K, massiv_A                          ! ва из 5-ти вещественных элементов
    write (*, gus)                                      ! возможна запись write (*, nml = gus)
```

4) в виде звездочки:

```
    ......
    write (*, *) a, k
    ......
    write (*, fmt=*) aa, kdf
    ......
    read *, a, k
```

Последняя или единственная звездочка операторов означает, что В/В управляется неименованным списком; первая задает устройство В/В (клавиатуру и экран).

Примечание.

1. В первых двух вариантах спецификация формата содержит ДУ 1x и t2. Это необходимо, т.к. в FPS при форматном выводе первая позиция поля отводится под простановку символа управления кареткой печатающего устройства. Для избежания генерации ложных сигналов рекомендуется вставлять хотя бы один пробел в качестве первого символа. Это выполняется дескриптором 1x или t2.

2. При вводе данных рекомендуется использовать бесформатный ввод, при выводе – форматный.

Вывод без продвижения. Для того, чтобы после очередного вывода записи курсор не перемещался на начало новой строки, в спецификации формата используют символ $ или \ .

```
.....
print '(1x, a, $)', 'chuchelo'
```

или

```
.....
print '(1x, a, \)', 'chuchelo'
```

Согласование списка В/В и спецификации формата. Коэффициент повторения. Реверсия формата. При форматном вводе-выводе каждому элементу списка В/В соответствует свой дескриптор данных (ДД). Элементы списка В/В и ДД должны быть согласованы по типам.

Пример:

```
.....
Integer k, n, m(9)          ! m(9) – одномерный массив m, состоящий из
                            ! 9-ти элементов
real mu
write (*, '(2x, I5, i8, i5, I5, 3x, f3.1)') k, n, m(3), m(5), mu ! m(3), m(5) – 3-й и
                                                                 ! 5-й элем. массива m
```

Если для вывода используются повторяющиеся ДД, используется ***коэффициент повторения*** (задаваемая перед ДД целая буквальная константа без знака).

Пример:

```
.....
Integer k, n, m(9)
real mu
write (*, '(2x, I5, i8, 2I5, 3x, f3.1)') k, n, m(3), m(5), mu
```

Коэффициент повторения может быть применен и для группы ДД:

```
.....
Integer k, n, m(9)
real mu
write (*, '(2x, I5, 2(i8, 2i5), 3x,  f3.1)') k, n, m(3), m(5), m(1), m(7),
*m(9), mu
```

Если число ДД в спецификации формата меньше числа элементов в списке В/В, возможны 2 варианта управления форматом:

1) при отсутствии в спецификации формата заключенных в скобки ДД *i*-й элемент списка В/В выбирает *i*-й ДД, после чего оставшиеся элементы списка В/В выберут те же ДД, что и предыдущие элементы;

2) если спецификация формата содержит заключенные в скобки ДД, используется *реверсия формата*: после того как формат будет исчерпан, управление форматом вернется к последней открывающей скобке либо к коэффициенту повторения перед ней (если он имеется).

Пример:

```
.....
Integer hh, bob, fox
real ant, uncle, www, zl, rez
.....
write (*, '(3x, 3I3, 2x, 2(f4.1,1x, f5.2))') hh, bob, fox, ant,
*uncle, www, zl, rez                    ! Используется ре-
                                        ! версия формата
```

6.3. Программирование разветвляющихся алгоритмов

Ветвление – выбор одного из возможных направлений выполнения алгоритма в зависимости от значений некоторых условий.

В приводимых в настоящем разделе операторе и конструкциях If логическое выражение (ЛВ) должно быть скаляром, то есть операндами ЛВ не могут быть массивы или их сечения.

6.3.1. Условный логический оператор If

Синтаксис:

If (ЛВ) оператор

Если истинно ЛВ, то выполняется оператор, в противном случае управление передается на последующий оператор программы.

6.3.2. Условный арифметический оператор If

Оператор имеет вид:

If (*exp*) m1, m2, m3

где *exp* – целочисленное или вещественное выражение;

m1, m2, m3 – метки исполняемых операторов – целочисленные константы в диапазоне от 1 до 999999 (значения меток могут совпадать).

Оператор обеспечивает переход по метке m1, если *exp* < 0; по метке m2, если *exp* = 0; по метке m3, если *exp* > 0.

6.3.3. Конструкции If

По сравнению с более ранними версиями в MS FPS в управляющие конструкции дополнительно введено необязательное имя конструкции. Применение именованных конструкций позволяет создать хорошо читаемые программы даже при большой глубине вложенности управляющих конструкций.

Существует 4 основных типа ветвлений:

1) если – то (конструкция If Then Endif);
2) если – то – иначе (конструкция If Then Else Endif);
3) если – то – иначе – если (конструкция If Then Else If Endif);
4) выбор по ключу.

Конструкция If Then Endif.

Синтаксис:

[имя:] If (ЛВ) Then
БОК
End If [имя]

Примечание.

Здесь и ниже БОК (блок операторов и конструкций) служит для выполнения одного или нескольких простых или сложных действий. БОК может содержать ветвления и циклы, которые являются примером сложных действий. Простым действием является, например, В/В данных, выполнение присваивания, вызов процедуры.

БОК выполняется, если истинно ЛВ. Если присутствует имя конструкции, оно должно быть и в первом, и в последнем операторе конструкции.

Если в БОК входит 1 оператор, синтаксис конструкции имеет вид:

If (ЛВ) оператор

Для записи ЛВ используются логические операции и операции отношения. В ЛВ могут также присутствовать арифметические и

символьные операции. Пробелы в записи логических операций и операций отношения не допускаются.

Конструкция If Then Else Endif.
Синтаксис:

[имя:] If (ЛВ) Then
БОК1
Else [имя]
БОК2
End If [имя]

В случае истинности ЛВ выполняется БОК1, если ЛВ ложно, – БОК2.

Таблица 6.2

Некоторые логические операции и операции отношения

Операции	Запись на языке FORTRAN	Типы операций
$=, \neq, >, <, \geq, \leq$	.EQ., .NE., .GT., .LT., .GE., .LE.	отношения
Не (отрицание)	.Not.	логическая
И	.And.	логическая
Или	.Or.	логическая

Конструкция If Then Else If Endif.
Синтаксис:

[имя:] If (ЛВ1) Then
БОК 1
Else If (ЛВ2) Then [имя]
БОК 2
.....
[Else [имя]
БОКn]
End If [имя]

В случае истинности ЛВ1 выполняется БОК1, и управление передается на следующий Else If, то есть вычисляется значение ЛВ2, и, если оно истинно, выполняется БОК2, если ложно, управление передается на следующий Else If, и т.д. Если ложны все ЛВ, выпол-

няется следующий за завершающим Else БОКn. Если завершающий Else отсутствует, управление передается на расположенный за End If оператор.

Пример. Найти корни квадратного уравнения $ax^2 + bx + c = 0$:

```
Program roots
real a,b, c, d, x1,x2
<ввод   a, b, c>
d=b**2-4*a*c
if (d.gt.0.0) then
x1=(-b+sqrt(d)) / (2*a)
x2=(-b-sqrt(d)) / (2*a)
write (*, '(1x, 2(a, f5.2, 1x))') 'root x1=', x1, 'root x2=', x2
else if (d.eq.0.0) then
x1=-b/(2*a)
x2=x1
write (*, '(1x, a, f5.2)') 'root x1=root x2=', x1
else
write (*, '(1x, a)') 'roots is not determined'
endif
end
```

6.4. Программирование циклических алгоритмов

Цикл – многократный повтор некоторых действий, последовательность которых задается исполняемыми операторами и (или) управляющими конструкциями.

Однократное выполнение БОК-цикла называется ***итерацией***.

Операторы и конструкции, входящие в цикл, называются ***телом цикла***.

Различают 3 вида циклов:

1) цикл "с параметром";

2) цикл "пока";

3) цикл "до".

Do-конструкции могут быть вложенными, причем степень вложения неограниченна.

6.4.1. Цикл "с параметром"

Синтаксис:

[имя:] Do p = p_s, p_e [,s]
БОК
End Do [имя]

где p – целая, вещественная, одинарной или двойной точности переменная, называемая ***переменной цикла***, или ***параметром цикла***;

p_s, p_e – целые, вещественные, одинарной или двойной точности скалярные выражения, задающие диапазон изменения P;

s - отличная от нуля величина, на которую изменяется значение параметра P после выполнения очередной итерации (при отсутствии шага s его значение по умолчанию устанавливается равным 1; шаг может быть отрицательной величиной).

Графически цикл с параметром иллюстрирует рис. 6.1.

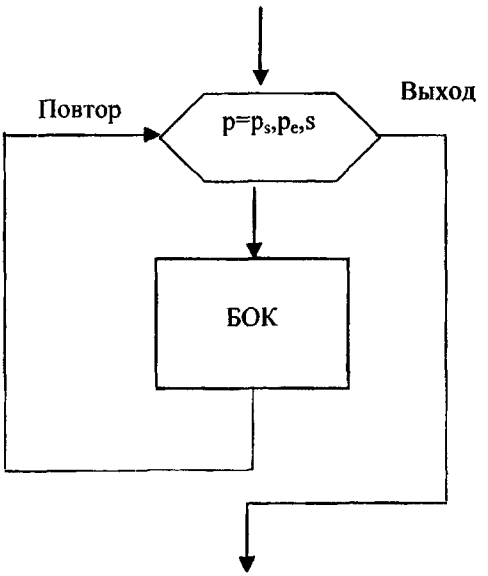


Рис. 6.1. Цикл с параметром

Цикл с параметром работает следующим образом (случай s > 0):

1) присвоить: p = p_s;

2) если p ≤ p_e, перейти к п.3, иначе завершить цикл;

3) выполнить БОК;
4) присвоить : p = p + s и перейти к п.2 (повтор).
Когда s < 0, п.2 выглядит так:
если $p \geq p_c$, перейти к п.3, иначе завершить цикл.

6.4.2. Цикл "пока"

Цикл "пока" (рис. 6.2) выполняется до тех пор, пока истинно некоторое ЛВ. Проверка истинности происходит перед началом очередной итерации (рис. 6.2 а).
Синтаксис:

[имя:] Do While (ЛВ)
БОК
End Do [имя]

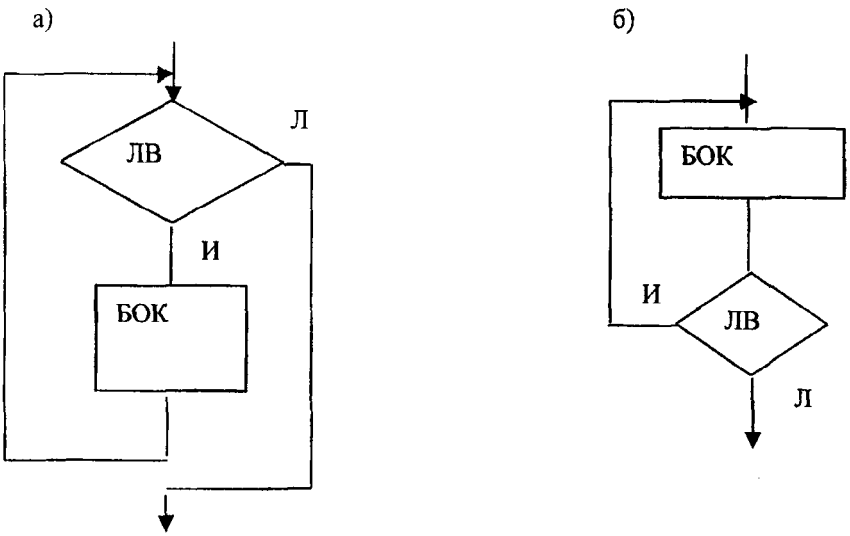


Рис. 6.2. Циклы "пока" и "до":
а – цикл "пока"; б – цикл "до"

6.4.3. Цикл "до"

В цикле "до" (рис. 6.2), вначале идет вычисление БОК, затем – проверка истинности условия прерывания цикла.

Синтаксис:

[имя:] Do
БОК
If (.not. ЛВ) Exit
End Do [имя]

При работе с Do-циклами запрещается:

1) изменять переменную *P* цикла операторами этого цикла;

2) переходить внутрь цикла посредством выполнения оператора Goto или альтернативного возврата из подпрограммы.

6.4.4. Управляющие операторы прерывания цикла

Любой из Do-циклов может быть прерван операторами Goto, Exit, Cycle и оператором *Return, обеспечивающим возврат из процедуры.*

Синтаксис:

Exit [имя] - передает управление из Do-конструкции на первый следующий за конструкцией оператор;

Cycle [имя] - передает управление на начало Do-конструкции; при этом операторы, расположенные после Cycle и до End Do, опускаются, – происходит переход к следующей итерации (т.е. управление передается операторам Do или Do While).

Операторы **Exit** и **Cycle** используются только в конструкциях **If**.

Пример:

```
Integer a(100), sum
<ввод массива a>
sum=0
do i=1, 100
if (a(i) .eq. 0) exit          ! Досрочный выход из цикла
if (a(i) .gt. 0) cycle         ! Суммирование не выполняется
sum = sum+a(i)
enddo
print*, sum
```

Оператор **Goto** безусловного перехода осуществляет передачу управления по метке.

Синтаксис:

Goto *метка*

Пример:

```
Integer a (100), sum
<ввод массива a>
sum=0
do i=1, 100
if (a(i) .eq. 0) goto 15        ! Переход управления к оператору с меткой 15
if (a(i) .gt. 0) cycle          ! Суммирование не выполняется
sum = sum+a(i)
enddo
15 print*, sum
```

Помимо вышеописанных операторов прерывания цикла в FPS существует ряд других опреаторов управления.

Оператор **Stop** прекращает выполнение программы.

Синтаксис:

Stop [сообщение]

Сообщение – символьная или целочисленная константа в диапазоне от 0 до 99999. При отсутствии *сообщения* на дисплее появится строка:

Stop – program terminated

Оператор **Pause** временно приостанавливает выполнение программы и позволяет пользователю выполнить команды операционной системы.

Синтаксис:

Pause [сообщение]

Если *сообщение* отсутствует, на дисплее появится строка:

Please enter a blank line (to continue) or a system command.

После выполнения оператора Pause возможны действия:

1) если пользователь вводит пробел, управление возвращается программе;

2) если пользователь вводит команду, выполняется команда, и управление возвращается программе (максимальный размер задаваемой на одной строке команды составляет 128 байт);

3) если введено слово Command (прописными или строчными), пользователь может выполнить последовательность команд операционной системы; для возврата в программу потребуется ввести Exit (прописными или строчными буквами).

Оператор **End** – последний в программе; сообщает транслятору, что в программе операторов больше нет.

6.4.5. Циклические списки ввода-вывода

Список ввода-вывода (см. подраздел 6.2) может содержать и циклический список.

Синтаксис:

(объект цикла, $p = p_s, p_e$ [,s])

объект цикла – переменная при вводе, выражение – при выводе

Пример:

```
Write (*, '(1x, a)') ('_', i=1,80)      ! Выводит горизонтальную линию
print*,'vvedite masiv A'
read (*,*) ((A(i,j), i=1,2), j=1,3)     ! Ввод с клавиатуры двухмерного мас-
                                        ! сива A(2,3)
```

6.5. Массивы

Массив – это именованный упорядоченный набор переменных одинакового типа.

Доступ к элементу или группе элементов массива обеспечивается при помощи ***имени*** *массива.*

Объектами (элементами) массива могут быть данные как базовых, так и производных типов. Массив является составной переменной.

Массивы могут быть *статическими* и *динамическими.*

Число измерений массива называется его ***рангом***, число элементов массива – ***размером***. Ранг массива ≤ 7.

Массив характеризуется ***формой***, которая определяется его рангом и протяженностью (экстентом) массива вдоль каждого измерения.

Каждая размерность массива может быть задана *нижней* и *верхней границей*, разделенной двоеточием:

```
real c (2:5, -1:1,0:9)
real b (4, 3, 10)
```

Если ранг, форма и размер массивов c и b совпадают, такие массивы называются *согласованными*.

6.5.1. Объявление массивов

Объявление массивов выполняется:
1) при объявлении типа переменных;
2) операторами Dimension, Allocatable, Common.

Пример:

```
Real a(15), asd(3,5)          ! Двухмерный массив с именем asd: 3 строки,
                              ! 5 столбцов
integer b(12)

integer a
dimension a(2,3)

real mm [allocatable] (:,:)   ! Измерения двухмерного динамического мас-
                              ! сива задаются двоеточием
real a
common a(15)
```

6.5.2. Инициализация массива

Примеры:
1) в разделе объявления типов данных:
real a (7) /1., 0., -5., 10., 6., 6., 6./
с использованием коэффициента повторения:
real a (7) /1., 0., -5., 10., 3*6/
2) с использованием оператора Data:

```
real a (7)
data a /1., 0., -5., 10., 6., 6., 6./
```

Во всех приведенных примерах после инициализации массива а:

а (1) = 1,0; а (2) = 0,0; а (3) = -5,0; а (4) = 10,0; а (5) = 6,0; а (6) = 6,0; а (7) = 6,0.

6.5.3. Размещение элементов массива в памяти ЭВМ. Доступ к элементам массива

Память компьютера является одномерной. Двумерный массив располагается в памяти ЭВМ по ***столбцам***, т.е. в FPS быстрее изменяется первый индекс массива. Например, массив а (3,2) расположится в памяти ЭВМ следующим образом.

1	0	5	7	8	6
a(1,1)	a(2,1)	a(3,1)	a(1,2)	a(2,2)	a(3,2)

Для инициализации ***по строкам*** можно воспользоваться циклическим списком ввода:

```
Integer a(3,2)
data ((a(i,j), j=1,2), i=1,3) /1, 0, 5, 7, 8, 6/
```

Доступ к элементу массива обеспечивается путем задания имени массива с указанием в круглых скобках номера строки и столбца, на пересечении которых данный элемент находится.

Пример. Дан двумерный вещественный массив а (3,3). Необходимо исправить третий элемент, добавив к нему число –4,5.

```
.....
Real a(3,3)
<ввод массива любым из известных способов>
a(3,1)=a(3,1)-4.5
print*, 'a(3,1)=', a(3,1)
print*, ((a(i,j), i=1,3), j=1,3)          ! Вывод на экран всего массива
end
```

Доступ ко всем элементам массива или к группе его элементов осуществляется с помощью Do-конструкции.

Пример. Необходимо преобразовать массив a (3,3), добавив к его отрицательным элементам число π = 3,14.

```
.....
Do i=1,3
do j=1,3
if (a(i,j) .lt. 0.) then
a(i,j)=a(i,j)+3.14
write(*,3) 'a(', i,  ',' , j, ')=', a(i,j)
endif
enddo
enddo
3  format (4x, a, i1, a, i1, a, f5.2)
   write (*,'(1x, a, i1, a, i1, a, 1x, f5.2)') (( 'a(',  i,  ','  , j, ')=', a(i,j), i=1,3), j=1,3)
   stop 'we have done it!'
   end
```

6.6. Внешние процедуры

В алгоритмическом языке FORTRAN могут быть определены два типа внешних процедур: ***подпрограммы*** и ***функции***.

Функция отличается от подпрограммы тем, что вызывается непосредственно из выражения и возвращает результат, который затем используется в этом выражении (аналогично встроенным внутренним функциям Cos, Sin и т.д.). Тип возвращаемого результата определяет тип функции. При задании внешней функции необходимо объявлять ее тип в разделе описаний вызывающей программной единицы по тем же правилам, что и для других объектов данных.

Процедуру следует оформлять в виде функции, если ее результат можно записать в одну переменную; в противном случае следует применять подпрограмму.

6.6.1. Структура подпрограмм

Структура подпрограмм (п/п) имеет следующий вид:

```
Subroutine имя п/п (список формальных параметров)
операторы описания
исполняемые операторы
end
```

Имя п/п является глобальным. Оно не должно совпадать с другим глобальным именем, а также использоваться для локального имени в вызывающей программной единице. Создается по общепринятым правилам (см. подраздел 6.1).

Обмен данными между процедурой и головной программой осуществляется через ***параметры процедур***.

Параметры, используемые при вызове процедуры, называются *фактическими*.

Параметры, используемые в процедуре, называются *формальными*.

Между фактическими и формальными параметрами должно быть установлено *строгое соответствие по числу, типу и порядку следования*.

Фактическими параметрами могут быть выражения, буквальные и именованные константы, простые переменные, массивы, элементы массивов, записи, элементы записей, строки, подстроки, процедуры и встроенные функции.

Формальными параметрами могут быть переменные (полные объекты), процедуры и звездочка (*).

Вызов подпрограммы выполняется оператором Call имя п/п (список фактических параметров).

Пример. Найти сумму отрицательных элементов двухмерного массива:

```
Program ku_ku
integer A(3,2), rez
data a((a(i,j), j=1,2), i=1,3)/1, -2, 2*3, 5, 6/
call ku_ku_ku (a, 3, 2, rez)
write (*, '(1x, a, 2x, i6)') ' rez=', rez
end
subroutine ku_ku_ku (x, m, n, rez_1)
integer x(m,n), rez_1
rez_1=0
do i=1,m
do j=1,n
if (x(i,j) .lt. 0) then
rez_1=rez_1+x(i,j)
else
print*, 'calculation impossible at this step'
```

```
endif
enddo
enddo
end
```

6.6.2. Структура функции

Структура функции имеет вид:

[*тип*] [recursive] Function *имя функции* (спис. форм. параметров) [result (имя результата)]
операторы описания
исполняемые операторы
end

При активизации опции Recursive функция будет обращаться к самой себе необходимое количество раз в соответствии с заданным алгоритмом.

Функция должна содержать ***результирующую переменную***, в которую помещается возвращаемый функцией результат. Эта переменная получает значение в результате присваивания в конце проведенных вычислений.

Имя результирующей переменной задается опцией result или совпадает с именем функции, если эта опция опущена.

Задаваемое предложением result имя результата не должно совпадать с именем функции.

Тип результирующей переменной определяет тип функции и может быть задан одним из способов:

1) посредством указания типа в заголовке функции;

2) явно в операторах описания типов данных, используемых в этой функции;

3) оператором Implicit;

4) неявно согласно правилам умолчания.

Пример:

```
Function 0p_0p (a, m, n)   ! Тип результирующей переменной задан неявно
integer A(m, n)            ! Согласно правилам умолчания (см. подраздел 7.1)
.....
real function 0p_0p (a, m, n)
integer A(m, n)
.....
```

```
function 0p_0p (a, m, n)
integer A(m, n)
real 0p_0p
……
function 0p_0p (a, m, n) result(GOP_gop)
real gop_GOP
integer A(m, n)
……
```

Кроме того, тип используемой функции должен быть отражен в операторах описания головной программы.

Передача управления в головную программу происходит после выполнения операторов End или Return, причем оператор Return может стоять в любом месте среди исполняемых операторов.

Пример. Найти сумму отрицательных элементов двухмерного массива:

```
Program kot
real  A(3,2), kis
<ввод массива a(3,2)>
write (*,1) rezult=', kis(a,3,2)
1  format (1x, a, 2x, f4.1)
end
real function kis (x, m, n)          ! m, n – соответственно число строк и
                                     ! столбцов массива X
real x(m,n)
kis=0.
do i=1,m
do j=1,n
if (x(i,j) .ge. 0.0) print*, 'no'
if (x(i,j) .lt. 0.0) then
kis=kis+x(i,j)
print *, 'find the element'
endif
enddo
enddo
return
end
```

7. ОСНОВЫ АЛГОРИТМИЧЕСКОГО ЯЗЫКА PASCAL

7.1. Классификация данных языка PASCAL

7.1.1. Алфавит и словарь языка PASCAL

Программа на языке PASCAL формируется с помощью конечного набора знаков, образующих алфавит языка, и состоит из букв, цифр и специальных символов. В качестве букв используются прописные и строчные буквы латинского алфавита:

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
a b c d e f g h i j k l m n o p q r s t u v w x y z,
цифры и знак подчеркивания:
1 2 3 4 5 6 7 8 9 0 _
специальные символы:
+ {} - . * , / : = ;
< ' > # [] $ () ^

Также используется пробел (не имеет обозначения).

Комбинации специальных символов могут образовать составные символы:

:= <=
<> >=
.. (..)
(* *)

Неделимые последовательности знаков алфавита образуют ***слова***, отделенные друг от друга ***разделителями***. Разделителем может служить пробел, символ конца строки, комментарии.

Слова подразделяются на:

1) зарезервированные слова;

2) стандартные идентификаторы (служат для обозначения заранее определенных разработчиками языка типов данных, констант, процедур и функций);

3) идентификаторы пользователя (применяются для обозначения меток, констант, переменных, процедур и функций, определенных самим программистом).

Зарезервированные слова TURBO-PASCAL:

and	end	nil	shr
absolute	file	not	string
asm	for	object	then
array	forward	of	to
begin	function	or	type
case	goto	packed	unit
const	if	procedure	until
constructor	implementation	program	uses
destructor	in	record	var
div	inline	repeat	virtual
do	interface	set	while
downto	label	with	xor
else	mod	shl	

Зарезервированные слова составляют основу языка и имеют строго фиксированное написание. Их значение не может быть переопределено пользователем.

7.1.2. Правила написания идентификаторов

Существуют общие правила написания идентификаторов:

1. Он может начинаться только с буквы или знака подчеркивания (исключение составляют метки, которые могут начинаться и с цифры, и с буквы).
2. Он может состоять только из букв, цифр и знака подчеркивания.
3. Между двумя идентификаторами должен быть по крайней мере один пробел.
4. Его максимальная длина – 127 символов.

При написании идентификаторов можно использовать прописные и строчные буквы. Компилятор не делает различий между ними.

Пример.

Metka12
1rew – ошибка
Blok_56
Nomer.Doma – ошибка

7.1.3. Константы и переменные

В программе каждый элемент данных является либо константой, либо переменной. Константы и переменные определяются идентификаторами.

Константами называются элементы данных, значения которых известны заранее и в процессе выполнения программы не изменяются. Тип констант автоматически распознается компилятором. В PASCAL для определения констант служит зарезервированное слово ***Const***.

Формат:

Const
<идентификатор> = <значение константы>;

Переменные могут менять свои значения в процессе выполнения программы. Тип переменной должен быть описан перед тем, как с переменной будут выполняться какие-либо действия. В PASCAL для определения переменных служит зарезервированное слово ***Var***.

Формат:

Var
<идентификатор>: <тип>;

7.1.4. Типы данных

Объекты, которыми оперирует программа, относятся к определенному типу. ***Тип*** – это множество значений, которые могут принимать объекты программы, и совокупность операций, допустимых над этими значениями.

PASCAL наряду со стандартными типами позволяет программисту образовать собственные типы.

Все допустимые в языке PASCAL типы подразделяются на две группы:

1) скалярные;
2) структурированные.

Скалярные типы подразделяются на:

1) стандартные;
2) описанные пользователем.

Структурированные типы базируются на скалярных и могут содержать их различные комбинации.

К ***стандартным скалярным типам*** относятся данные следующих типов:

1) целочисленного;
2) байтового;
3) вещественного;
4) булевского;
5) литерного.

Целочисленный тип определяет все целые числа в диапазоне от -32768 до 32767. Для его описания служит стандартный идентификатор **Integer**.

Формат:

<идентификатор>: integer;

Для размещения в памяти переменной типа Integer требуется 2 байта.

Байтовый тип аналогичен целочисленному, но охватывает более узкий диапазон значений от 0 до 255.

Формат:

<идентификатор>: byte;

Для размещения в памяти переменной типа **Byte** требуется 1 байт. В арифметических и логических выражениях допустимо смешение типов Integer и Byte.

Вещественный тип данных включает все положительные числа от 1E-38 до 1E+38, соответствующие отрицательные числа и 0. Мантисса может содержать 11 значащих цифр. Описывается стандартным идентификатором **Real**. Данные этого типа могут записываться в формате с плавающей и фиксированной точками.

Значение:	С плавающей точкой:	С фиксированной точкой:
0	0.0000000000E+00	0
134	1.3400000000E+02	134
-7611	-7. 610000000E+01	-7611

Формат:

<идентификатор>: real;

Для размещения в памяти переменной типа Real требуется 6 байт. Данные этого типа нельзя определять как базовый тип множества, применять при индексировании массивов.

Булевский тип данных описывается стандартным идентификатором **Boolean**. Переменные и константы этого типа могут принимать только одно из 2-х значений, определяемых стандартными константами True (истина) и False (ложь).

Формат:

<идентификатор>: boolean;

Для размещения в памяти переменной типа Boolean требуется 1 байт. Переменные этого типа используются для управления порядком выполнения операторов программы.

Литерный тип данных описывается стандартным идентификатором **Char**. Константы и переменные этого типа могут принимать одно из значений кодовой таблицы ПЭВМ.

Формат:

<идентификатор>: char;

Для размещения в памяти переменной типа Char требуется 1 байт. Значения переменных и констант этого типа должны быть заключены в апострофы. Например: 'A' представляет букву A, ' ' - пробел. Использование данных типа Char в арифметических выражениях запрещено.

К ***скалярным типам пользователя*** относятся следующие типы:

1) перечисляемый;

2) интервальный.

Данные этих типов занимают в памяти 1 байт, поэтому не могут содержать более 256 элементов.

Синтаксис данных типов подробно описан в 5.3.1 [1, с. 42].

Структурированные типы данных определяют упорядоченную совокупность скалярных переменных и характеризуются типом своих компонентов. В языке PASCAL допускаются следующие структурированные типы данных:

1) строки;

2) массивы;

3) множества;

4) записи;
5) файлы;
6) указатели.

Строка – это последовательность символов. При использовании в выражениях она заключается в апострофы. Для определения данных строкового типа используется идентификатор **String**, за которым следует заключенное в квадратные скобки значение максимально допустимой длины строки данного типа.

Формат:

```
Type
имя типа> = string [максимальная длина строки];
var
идентификатор, ... >: <имя типа>;
```

Строковыми выражениями называются выражения, в которых операндами служат строковые данные. Они состоят из строковых констант, переменных, указателей функций и знаков операций. Над строковыми данными допустимы операция сцепления и операции отношения.

Операция сцепления (+) применяется для сцепления нескольких строк в одну результирующую строку.

Пример:

Выражение:	Результат:
'E'+'C'+' 18'+'40'	'EC 1840'
'Дом'+' номер'+' 5'	'Дом номер 5'

Операции отношения (=, <>, >, <, >=, <=) проводят сравнение двух строковых операндов и имеют приоритет более низкий, чем операции сцепления. Сравнение строк производится слева направо до первого несовпадающего символа. Результат выполнения операций отношения над строковыми данными всегда имеет булевский тип и принимает значение True, если выражение истинно, иначе – False.

Большей считается строка, в которой первый несовпадающий символ имеет больший номер в стандартной таблице обмена информацией.

Пример:

Выражение:	Результат:
'Cosm2' > 'Cosm1'	True
'Dos1.0' < 'Dos1.0'	False

Если строки имеют различную длину, но в общей части символы совпадают, считается, что более короткая строка меньше более длинной. Строки считаются равными, если они полностью совпадают по длине и содержат одни и те же символы.

Для присваивания строковой переменной результата строкового выражения используется оператор присваивания (:=).

Пример:

```
Str1:= 'Группа студентов';
Str2:= Str1 + ' первого курса';
```

Для обработки строковых данных используются стандартные процедуры и функции.

7.1.5. Общие сведения о структуре программ

Программа на языке PASCAL состоит из строк. Максимальная длина строки не должна превышать 127 символов. Все лишние символы компилятором игнорируются. Набор текста программы осуществляется с помощью встроенного редактора текстов системы программирования PASCAL или любого другого редактора.

Размер программы имеет предел. Редактор текстов и компилятор позволяют обрабатывать программы размером до 64 кбайт. Если программа требует большего количества памяти, следует воспользоваться средствами включения файлов.

Синтаксически программа состоит из необязательного ***заголовка*** и ***блока***. Заголовок состоит из зарезервированного слова Program, имени программы и параметров, с помощью которых программа взаимодействует с операционной системой. После заголовка следует программный блок, состоящий в общем случае из 6 разделов. В общем случае структура PASCAL-программы имеет следующий вид:

```
{Заголовок}
program имя;
uses список используемых модулей;
{блок}
label                    – раздел описания меток;
const                    – раздел описания констант;
type                     – раздел описания типов данных;
var                      – раздел описания переменных;
procedure, function      – раздел описания процедур и функций;
begin
оператор 1;
оператор 2;
.....                    – раздел операторов
оператор n
end.
```

Любой раздел, кроме последнего, может отсутствовать. Разделы описаний могут встречаться в программе любое количество раз и следовать в любом порядке. Главное, чтобы все описания объектов программы были сделаны до того, как они будут использованы.

7.1.6. Модули

Модуль представляет собой набор констант, типов данных, переменных, процедур и функций. Каждый модуль по своей структуре аналогичен отдельной программе. Вместе с тем, структура модуля позволяет использовать его как своеобразную библиотеку описаний. Модули являются достаточно гибким и удобным средством при разработке больших программных комплексов. При выполнении программы каждому модулю отводится свой отдельный сегмент оперативной памяти.

TURBO-PASCAL располагает 6-ю стандартными (встроенными) модулями: System, Dos, Overlay, Graph, Crt, Printer. Все перечисленные стандартные модули (кроме Graph) объединены и хранятся в файле TURBO.tpl.

Модуль System – поддерживает все стандартные процедуры и функции, обеспечивает ввод-вывод данных, обработку строк, динамическое распределение оперативной памяти и ряд других возмож-

ностей TURBO-PASCAL. Подключается к любой программе автоматически.

Модуль Dos – содержит многочисленные стандартные процедуры и функции, многие из которых по своему действию эквивалентны командам MS-DOS.

Модуль Overlay – обеспечивает поддержку системы оверлеев.

Модуль Crt – поддерживает ряд стандартных процедур и функций, которые обеспечивают работу с экраном дисплея в текстовом режиме, управление звуком и работу с клавиатурой.

Модуль Printer – содержит драйвер печатающего устройства и позволяет организовать вывод информации на принтер.

Модуль Graph – обеспечивает работу с экраном дисплея в графическом режиме.

Наряду с использованием стандартных модулей каждый программист имеет возможность организации собственных модулей.

Для того, чтобы использовать модули в программах, их имена следует указать в предложении Uses. Например, разработана программа, которая наряду со стандартным модулем Crt использует и оригинальный модуль с именем **Modul**. В этой программе следует указать список всех используемых модулей в следующем виде:

```
program Pr;
uses Crt, Modul;
.....
```

Модули транслируются отдельно. В отличие от основных программ, результатом трансляции которых будут файлы с расширением exe, модули получают расширение tpu. Полученные в результате трансляции tpu-файлы можно подсоединить к стандартному файлу Turbo.tpu (с помощью утилиты Tpumover). Если этого не делать, при трансляции самой программы все используемые в ней модули (tpu-файлы) подсоединятся автоматически. Если в какой-нибудь из используемых модулей были внесены изменения, при трансляции программы пользователя все модифицированные модули будут автоматически перетранслированы (эту функцию реализует интегрированная среда программирования TURBO-PASCAL).

7.1.7. Раздел описания меток

Перед любым оператором языка PASCAL можно поставить метку, что позволяет выполнить прямой переход на этот оператор с помощью оператора Goto из любого места программы.

Метка состоит из имени и следующего за ним двоеточия. Именем может служить идентификатор или цифра. Максимальная длина метки ограничена 127 символами.

Перед употреблением метка должна быть описана. Раздел описания меток начинается зарезервированным словом **Label** (метка), за которым следуют имена меток, разделенные запятыми. За последним именем ставится точка с запятой.

Формат:

Label < имя, ... >;

Пример:

```
Label metka1, metka2, 111;
.....
111: <оператор>;
.....
metka1: <оператор>;
```

7.1.8. Раздел описания констант

В разделе описания констант производится присваивание идентификаторам констант постоянных значений. Раздел описания начинается зарезервированным словом **Const**, за которым следуют выражения, присваивающие идентификаторам постоянные числовые или строковые значения. Выражения присваивания отделяются друг от друга точкой с запятой.

Формат:

Const <идентификатор> = <значение>;

Пример:

```
Const
max = 100;
vin = 7;
```

```
kods = #124;
Vxod = 'начало';
```

После того как константа определена, ей нельзя присвоить другое значение.

7.1.9. Раздел описания типов данных

Тип данных может быть либо описан непосредственно в разделе описания переменных, либо определяться идентификатором типа. Стандартные типы (Integer, Byte, Real, Char, Boolean) не требуют описания в отличие от типов, образованных пользователем. Раздел описания типов данных начинается зарезервированным словом Type, за которым следуют одно или несколько определений типов, разделенных точкой с запятой.

Формат:

Type <имя типа> = <значение типа>;

Пример:

```
Type
matr = array [1..5] of real;    {Одномерный вещественный массив}
dni  = 1 .. 31;                 {Интервальный скалярный тип пользователя}
```

Каждое описание задает множество значений и связывает с этим множеством некоторое имя типа.

7.1.10. Раздел описания переменных

Каждая переменная перед использованием должна быть описана.

Раздел описания типов переменных начинается зарезервированным словом **Var**, за которым через запятую перечисляются имена переменных и через двоеточие следует их тип.

Формат:

Var <идентификатор>: <тип>;

Пример:

```
Var
A,B,C: integer; bykva, simvol: char;
result, summa: real; Aa,Bb,Cc,Dd: boolean;
```

Раздел описания процедур и функций.

Подпрограммой называется программная единица, имеющая имя, по которому она может быть вызвана из других частей программы. Роль подпрограмм выполняют ***процедуры*** и ***функции***. Для описания подпрограмм используются зарезервированные слова **Procedure** и **Function**, которые записываются в начале подпрограммы.

Формат процедуры:

Procedure <имя процедуры> {<параметры>};
<разделы описаний>
<раздел операторов>

Формат функции:

Function <имя функции> {<параметры>};
<разделы описаний>
<раздел операторов>

Процедуры и функции подразделяются на *стандартные* и *определенные пользователем*. Стандартные процедуры и функции являются частью языка и могут вызываться без предварительного описания. Описание процедур и функций пользователя обязательно.

Раздел операторов является основным, так как именно в нем выполняются действия, позволяющие получить результат, ради которого создавалась программа.

Раздел операторов начинается зарезервированным словом **Begin** (начало); далее следуют операторы языка, отделенные друг от друга точкой с запятой; завершают раздел зарезервированное слово **End** (конец) и точка.

Формат:

Begin
<оператор>;
.....
<оператор>
end.

Операторы выполняются строго в том порядке, в каком они записаны в тексте программы в соответствии с правилами синтаксиса и пунктуации.

Комментарий - это пояснительный текст, который можно записать в любом месте программы, где разрешен пробел. Текст комментария ограничен символами { } или (* *). Его текст может содержать любые символы.

Пример.

{Комментарий к программе}
(* Программа вычисления полинома Лагранжа*)

В ограничителях (* *) пробелы между скобкой и звездочкой запрещены. В тексте не должны находиться знаки ограничителей, с которых начинается комментарий. Ограничения на длину комментария нет. Он игнорируется компилятором, и поэтому никакого влияния на программу не оказывает.

7.2. Выражения. Операции

Переменные и константы всех типов используются в выражениях. ***Выражение*** задает порядок выполнения действий над элементами данных и состоит из операндов (констант, переменных, обращений к функциям), круглых скобок и знаков операций.

Операции определяют действия, которые надо выполнить над операндами.

Операции в языке PASCAL подразделяются на:

1) арифметические;
2) отношения;
3) логические (булевские);
4) строковые и др.

Выражения соответственно называются арифметическими, отношения – булевскими, строковыми и др. в зависимости от того, какого типа операнды и операции в них используются.

Арифметические выражения и операции.

Арифметическое выражение порождает целое или действительное значение. ***Арифметические операции*** выполняют арифметические действия в выражениях над значениями операндов типа Real, Integer, Byte. Арифметические операции представлены в табл. 7.1.

Таблица 7.1

Знак	Выражение	Тип операнда	Операция
+	A + B	real integer real, integer	сложение
-	A - B	real integer real, integer	вычитание
*	A * B	real integer real, integer	умножение
/	A / B	real integer real, integer	деление
div	A div B	integer	целочисленное деление
mod	A mod B	integer	деление с остатком
and	A and B	integer	арифметическое И
shl	A shl B	integer	целочисленный сдвиг влево
shr	A shr B	integer	целочисленный сдвиг вправо
or	A or B	integer	арифметическое Или
hor	A hor B	integer	исключающая дизъюнкция
-	-A	integer real	изменение знака

Приоритет операций в порядке убывания: /, *, Div, Mod, And, Or, Shl, Shr, -, +. Порядок выполнения операций регулируется с помощью скобок.

Выражения и операции отношения.

Выражение отношения определяет истинность или ложность результата. ***Операции отношения*** выполняют сравнение 2-х операндов и определяют, истинно значение выражения или ложно. В табл. 7.2 приведены операции отношения.

Таблица 7.2

Операции отношения

Знак	Пример	Результат	Операция
=	A = B	true, если A равно B	равно
<>	A <> B	true, если A не равно B	не равно
>	A > B	true, если A больше B	больше
<	A < B	true, если A меньше B	меньше
>=	A >= B	true, если A больше или равно B	больше или равно
<=	A <= B	true, если A меньше или равно B	меньше или равно

Все операции являются бинарными. Приоритет операций в порядке убывания: =, <>, <, >, >=, <=.

Сравниваемые величины могут принадлежать к любому скалярному типу или перечисляемому типу данных. Результат имеет булевский тип и принимает одно из двух значений: True (истина) или False (ложь). При объединении в одном выражении арифметических операций и операций отношения первыми всегда выполняются арифметические.

Логические выражения и операции.

Результатом выполнения ***логического выражения*** является логическое значение True и False.

Простейшими видами логических выражений являются следующие:

1) логическая константа;
2) логическая переменная;
3) элемент массива логического типа;
4) логическая функция;
5) выражение отношения.

Другие логические выражения строятся из выше перечисленных путем применения логических операций и круглых скобок. Список ***логических операций*** приведен в табл. 7.3.

Таблица 7.3

Логические операции

Операция	Пример	Значение A	Значение B	Результат	Название
not	not A	true false		false true	логическое отрицание
and	A and B	true true false false	true false true false	true false false false	логическое И
or	A or B	true true false false	true false true false	true true true false	логическое Или
xor	A xor B	true true false false	true false true false	false true true false	исключаемое Или

Операции And, Or, Xor – бинарные, операция Not – унарная. Приоритет операций в порядке убывания: Not, And, Or, Xor. Приоритет логических операций выше приоритета операций отношения.

Стандартные функции.

Наиболее часто используемые функции TURBO-PASCAL приведены в табл. 7.4. Необходимо отметить, что в тригонометрических функциях синуса и косинуса аргумент должен быть задан только в радианах. Если аргумент задан в градусах, для перевода его в радианы используется формула

$$Y = X \cdot \pi / 180.$$

Таблица 7.4

Арифметические встроенные функции

Функция	Назначение
1	2
Abs(X)	вычисление абсолютного значения X
Sqr(X)	вычисление квадрата X (X · X)
Sin(X)	вычисление синуса X

1	2
Cos(X)	вычисление косинуса X
Arctg(X)	вычисление арктангенса X
Exp(X)	вычисление экспоненты X
Ln(X)	вычисление натурального логарифма X
Sqrt(X)	вычисление квадратного корня из X
Trunc(X)	вычисление целой части X
Round(X)	округление X в сторону ближайшего целого:
Odd(X)	true, если X – нечетное; false, если X – четное

В TURBO-PASCAL определены только 3 тригонометрические функции (Sin, Cos, Arctg). Для вычисления остальных тригонометрических функций необходимо использовать известные соотношения:

$Tg(X) = Sin(X) / Cos(X)$
$Ctg(X) = Cos(X) / Sin(X)$
$Csc(X) = 1 / Sin(X)$
$Sc(X) = 1 / Cos(X)$
$Arcsin(X) = Arctg(X/(1-X^2))^{1/2}$
$Arccos(X) = \pi/2 - Arcsin(X)$
$Arcctg(X) = \pi/2 - Arctg(X)$.

Для вычисления логарифма с основанием A используется соотношение:

$$\log_A(X) = \ln(X) / \ln(A).$$

В TURBO-PASCAL нет операции возведения в степень. При необходимости ее использования применяют стандартные функции:

X^A соответствует EXP(A*ln(X)).

7.3. Операторы языка PASCAL

7.3.1. Операторы

Основная часть программы на языке PASCAL представляет собой последовательность ***операторов***, каждый из которых произво-

дит некоторое действие над данными. Операторы выполняются последовательно в том порядке, в котором они записаны в тексте программы. Разделителем операторов служит точка с запятой.

Все операторы подразделяются на три группы:

1) простые;

2) структурные;

3) ввода-вывода.

Кроме операторов, входящих в перечисленные группы, имеется еще оператор With.

Простыми называются операторы, не содержащие в себе никаких других операторов. К ним относятся операторы:

1) присваивания;

2) безусловного перехода;

3) вызова процедуры или функции;

4) пустой.

Оператор присваивания (:=) предписывает выполнить выражение, заданное в его правой части, и присвоить результат переменной, идентификатор которой расположен в левой части. Переменная и выражение должны иметь один тип, за исключением случая, когда переменная имеет вещественный тип, а выражение - целочисленный. Допустимо присваивание любых типов данных, кроме файловых.

Формат:

<Идентификатор> := <выражение> ;

Пример:

```
Sort:= 1;
cena:= 15.754;
resultat:= sin(A)+cos(B);
```

Оператор безусловного перехода (Goto) означает "перейти к" и применяется в случаях, когда после выполнения некоторого оператора надо выполнить не следующий по порядку оператор, а какой-либо другой, отмеченный меткой.

Формат:

Goto <метка>;

Пример:

```
...
Label metka1, metka2;
...
metka1:
goto metka2;
...
metka2:
...
```

При записи оператора Goto нужно помнить следующее:

1) метка должна быть описана в разделе описания меток того блока процедуры, функции, основной программы, в котором она используется;

2) областью действия метки является тот блок, в котором она описана; переход возможен только в пределах блока;

3) при передаче управления внутрь другого блока происходит программное прерывание.

Обычно оператор Goto используется для преждевременного выхода из цикла или при обработке ошибок.

Оператор вызова процедуры служит для активизации предварительно определенной пользователем или стандартной процедуры.

Формат:

<имя процедуры> {(параметры)};

Пример:

```
Program prim;
procedure v1;
begin
...  {тело процедуры}
end;
procedure v2;
begin
...  {тело процедуры}
end;
```

```
begin
v1;   {вызов процедуры v1}
v2;   {вызов процедуры v2}
end.
```

Пустой оператор не содержит никаких символов и не выполняет никаких действий. Он может быть расположен в любом месте программы, где синтаксис языка допускает наличие оператора.

Пример:

```
Begin
goto m1;  {переход в конец блока}
...
m1:  {пустой оператор помечен меткой}
end.
```

Структурный оператор представляет собой структуру, построенную из других операторов по строго определенным правилам.

Все структурные операторы подразделяются на три группы:

1) составные;
2) условные;
3) повтора.

Составной оператор представляет собой группу из произвольного числа операторов, отделенных друг от друга точкой с запятой и ограниченную операторными скобками Begin и End.

Формат:

```
Begin
<оператор>;
<оператор>;
...
<оператор>;
end;
```

Пример:

```
Begin
t:=A*B+C;
rez:=A+sin(C);
writeln(rez:12:6)
end;
```

Составной оператор воспринимается как единое целое и может находиться в любом месте программы. Обычно он используется при написании условных операторов.

Условные операторы If и Case обеспечивают выполнение или невыполнение некоторого оператора, группы операторов или блока в зависимости от заданных условий. PASCAL допускает использование двух условных операторов: If и Case.

Оператор условия If является одним из самых популярных средств, изменяющих естественный порядок выполнения операторов программы. Он может принимать одну из следующих форм:

1) if <условие> then <оператор1> else <оператор2> ;

2) if <условие> then <оператор>;

Условие – выражение булевского типа. Оно может быть простым или сложным. Результат выражения всегда имеет булевский тип.

В первом случае, если значение выражения истинно, выполняется <оператор1>, если ложно, – <оператор2>.

Пример:

```
if A>B then writeln('A больше B')
else writeln('A меньше или равно B');
```

Во втором случае, если результат выражения – True, выполняется <оператор>, если False, – оператор, следующий сразу за оператором If.

Один оператор If может входить в состав другого оператора If. При вложенности операторов каждое Else соответствует тому Then, которое непосредственно ему предшествует.

Оператор выбора Case является обобщением оператора If и позволяет сделать выбор из произвольного числа имеющихся вариантов. Он состоит из выражения, называемого *селектором*, и списка параметров, каждому из которых предшествует список констант выбора. Как и в операторе If, здесь может присутствовать слово Else, имеющее тот же смысл.

Формат:

```
Case <выражение-селектор> of
<список 1>: <оператор 1>;
<список 2>: <оператор 2>;
...
```

<список N>: <оператор N>;
else <оператор>
end;

Оператор Case работает следующим образом. Сначала вычисляется значение выражения-селектора, затем обеспечивается реализация того оператора, константа выбора которого равна текущему значению селектора. Если ни одна из констант не равна текущему значению селектора, выполняется оператор, стоящий за словом Else. Если слово Else отсутствует, активизируется оператор, стоящий за словом End. Селектор может иметь любой скалярный тип, кроме вещественного.

Пример:

```
Case I of
1: Z:=I+10;
2: Z:=I+100;
3: Z:=I+1000
end;
```

Операторы ввода-вывода.

Ввод данных – это передача информации от внешнего носителя в оперативную память для обработки. ***Вывод*** – обратный процесс, когда данные передаются после обработки из оперативной памяти на внешний носитель. Внешним носителем может служить:

1) терминал ввода-вывода;
2) печатающее устройство;
3) гибкий (дискета) или жесткий (винчестер) магнитный диск и др.

Для выполнения операций ввода-вывода служат 4 оператора:

1) Read;
2) Readln;
3) Write;
4) Writeln.

Оператор чтения ***Read*** обеспечивает ввод числовых данных, символов, строк и т.д. для последующей их обработки программой.

Формат:

```
read (X1,X2,...,Xn);
```

или

```
read(FV,X1,X2,...,Xn),
```

где X1,X2,...,Xn – переменные допустимых типов данных;

FV – переменная, связанная с файлом, откуда будет выполняться чтение.

В данном разделе рассматривается, в основном, первый вариант формата.

Значения X1,X2,...,Xn набираются пользователем на клавиатуре минимум через один пробел и высвечиваются на экране. После набора данных для одного оператора Read нажимается клавиша ввода Enter. Значения переменных должны вводиться в строгом соответствии с синтаксисом языка PASCAL.

Пример:

```
...
Var
I: real;
J: integer;
K: char;
begin
read(I,J,K);
...
```

Первый вариант ответа:	Второй вариант ответа:
235.6 100 'G' <Enter>	'G' 235.6 100

Первый вариант обеспечивает нормальный ввод данных; второй вызовет ошибку.

Оператор чтения Readln аналогичен оператору Read; единственное отличие – в том, что после считывания последнего в списке значения для одного оператора Readln данные для следующего оператора чтения будут считываться с начала новой строки.

Оператор записи Write производит вывод числовых данных, символов, строк и булевых значений.

Формат:

```
write(Y1,Y2,...,Yn);
```

или

```
write(FV,Y1,Y2,...,Yn);
```

где Y1,Y2,...,Yn – выражения типа Integer, Real, Byte, Char, Boolean и т.д.;

FV – имя файла, куда выполняется вывод.

Пример:

```
write(234);
write(A+B-2);
```

Форматы оператора вывода Write. В TURBO-PASKAL предусмотрен вывод данных с форматом. В общем случае формат имеет следующий вид:

```
P:M,
```

где P – имя переменной;

M – целая константа, указывающая на число позиций для выводимой величины P (в качестве параметра M может указываться не число, а имя константы, описанное в разделе Const). Вывод осуществляется в крайние правые позиции поля шириной M.

Для вещественных переменных формат может быть задан в виде:

```
P:M:N,
```

где M – общее число позиций для выводимой переменной P, включая знак числа, целую часть, точку и дробную часть;

N – число позиций дробной части (0<= N <=24).

В этом случае переменная P выводится в виде константы с фиксированной точкой. Если N = 0, ни дробная часть, ни десятичная точка не выводятся. Если N > 24, при выводе используется формат с плавающей точкой.

Если параметры M и N опущены, вещественная переменная выводится в виде константы с плавающей точкой. В этом случае значения M и N устанавливаются транслятором по умолчанию.

Пример. Используем форматный вывод переменных I, J, K (см. выше пример с использованием оператора ввода Read):

```
.....
writeln('I=', I:6:2, 'j=', j:4, 'k=', k:2)
end.
```

В результате получим:

I=235.60 j=100 k=G

Оператор записи Writeln аналогичен оператору Write; единственное отличие заключается в том, что после вывода последнего в списке значения для текущего оператора Writeln происходит перевод курсора к началу следующей строки. Оператор Writeln, записанный без параметров, вызывает перевод строки.

7.3.2. Файлы

Файл – это поименованная область памяти на внешнем носителе, предназначенная для хранения информации (любого набора элементов одного и того же типа). Число элементов, называемое *длиной файла*, не фиксировано. Файл может быть связан с любым источником или потребителем информации:

1) клавиатурой;
2) принтером;
3) магнитным диском;
4) коммутационным каналом и др.

Файлы по методу доступа к их элементам подразделяются на файлы последовательного и прямого доступа. В файлах ***последовательного доступа*** каждый элемент становится доступным только после перебора всех предыдущих элементов. Файлы ***прямого доступа*** позволяют обращаться к каждому элементу непосредственно по его порядковому номеру в файле.

По отношению к программе файлы могут быть внешними и внутренними. ***Внутренними*** являются файлы, которые создаются, используются и существуют только во время работы данной программы, ***внешними*** – файлы, которые существуют вне программы. В качестве носителей внешних файлов обычно используют магнитные диски.

Каждому файлу пользователя должно быть присвоено уникальное ***имя***, которое используется при обращении к этому файлу. Имя состоит из собственно имени (1-8 символов: букв и цифр) и необязательного типа файла (3 символа). Если тип файла присутствует, он отделяется от первой части имени точкой. Внешние файлы должны быть описаны в разделе описаний программы.

Формат:

```
Type
<имя типа> = <базовый тип>;
var
<имя файла>: file of <имя типа>;
```

или

```
var
<имя файла>: file of <базовый тип>;
```

В качестве ***базового типа*** элементов файла можно использовать любой тип данных (как простой, так и сложный).

Пример:

```
Type
dlina: array [1..10] of real;
ddr: set of 1..50;
end;
var
kar: file of integer;
karta: file of dlina;      {Каждый элемент файла – массив}
rrr: file of ddr;          {Каждый элемент файла – множество}
kot: text;                 {Текстовый файл }
```

Примечание. Текстовый файл содержит текст, состоящий из обычных символов (букв алфавита и цифр), разбитых на строки (до 256 символов в строке). Текстовые файлы являются файлами последовательного доступа.

Доступ к компонентам (элементам) файла осуществляется через ***указатель файла***. При чтении или записи этот указатель перемещается к следующему компоненту и делает его доступным для обработки.

Процедуры и функции обработки файлов. При описании процедур и функций используются следующие обозначения:

FV – файловая переменная;

Str – строковое выражение;

P – переменные p1, p2,..., pn того же типа, что и компоненты переменной FV;

n – целочисленное выражение.

Процедуры выполняют следующее:

Assign (FV, Str) – связывает файловую переменную FV в программе с внешним файлом, имя которого указано в Str. Строковое выражение Str имеет вид: 'диск:\ имя каталога\ имя подкаталога\...\имя файла'.

Rewrite (FV) – открывает новый файл для записи. Имя файла должно быть предварительно определено в процедуре Assign. Если на диске уже был файл с таким именем, он уничтожается. Указатель файла устанавливается на первый компонент файла. Файл не содержит ни одного компонента, он только подготовлен для загрузки.

Reset (FV) – открывает файл для чтения. При этом указатель файла устанавливается на его первый элемент. Если эта процедура применена к несуществующему файлу, возникает ошибка ввода-вывода.

Read (FV, P) – производит чтение из внешнего файла, определенного файловой переменной FV, значений p1, p2, pn. После завершения выполнения процедуры указатель перемещается на следующий компонент.

Write (FV, P) – записывает переменные p1, p2, pn во внешний файл. После завершения выполнения процедуры указатель перемещается на следующий компонент.

Seek (FV,n) – позволяет осуществить прямой доступ к элементам внешнего файла. При этом указатель файла перемещается к элементу с порядковым номером n. Первый элемент файла имеет номер n = 0, второй – n = 1 и т.д.

Close (FV) – закрывает внешний файл. При этом разрывается связь, установленная процедурой Assign для FV. Если файл был открыт, никогда не следует выходить из программы, предварительно не закрыв его.

Функции выполняют следующее:

Eof (FV) – возвращает True, если указатель файла находится сразу за последним компонентом файла, в другом случае – False.

FileSize (FV) – возвращает объем файла в байтах.

FilePos (FV) – возвращает текущую файловую позицию.

Пример. Ввод/вывод данных с использованием внешнего файла:

```
Var
f, q: text;
y: array[1..6] of real; h, kon, koff, E0, alfa, beta, gamma, md, mmt: real;
```

```
begin
{Ввод данных с внешнего файла kot.dat}
assign(f, 'd:\users\301110\kot.dat);
reset(f);
readln(f, h, kon, koff);
readln(f, E0, alfa, beta, gamma);
for i=1 to 6 do read (f, y[i]);          {Чтение одномерного массива}
close(f);
.....
{Вывод данных во внешний файл kot.rez}
assign(q, 'd:\users\301110\kot.rez);
rewrite(q);
writeln(q, ' результат имитационного моделирования процесса трогания');
writeln(q);
writeln(q, 'w2=', y[2]:6:2, 'мдв=', md:5:1)
writeln(q, 'w4=', y[4]:6:2, 'ммт=', mmt:5:1)
close(q)
end.
```

7.3.3. Программирование линейных и разветвляющихся алгоритмов

Пример. Составить программу для вычисления выражения

$$W = \lg\left|a^{7}\right| + \operatorname{arctg}(x^{3})\frac{\pi \cdot a}{\sqrt{|a+x|}}\left(b - \cos\frac{a}{b}\right),$$

где a = 1,03; b = 2,91 E-3; x = 5,27 E+2.

```
{Линейный алгоритм}
program Lin;
var A,B,C:real;
W,X:real;
begin A:=1.03; B:=2.91E-03; X:=527;
W:=Exp(7*Ln(A)); W:=Abs(W);
W:=Ln(W)/Ln(10);
C:=Exp(3*Ln(X)); C:=Arctan(C);
```

```
C:=C*Pi*A/Sqrt(Abs(A+X));
C:=C*(B-Cos(A/B));
W:=W+C;
writeln (' A=',A,' B=',B,' X=',X);
write (' W=',W)
end.
```

Пример. Составить программу для вычисления выражения

$$m = \begin{cases} e^{-\frac{|a|+|b|}{a^2+b^2}} & \text{при } a \cdot b > 0.5; \\ |a+b| & \text{при } 0.4 < a \cdot b \le 0.5, \end{cases}$$

где a = 0,0256; b = 1,39 E-2.

```
{Разветвляющийся алгоритм}
program Raz;
label 10;
var A,B,M:real; M1,M2:real;
begin
read (A,B);
M1:=-(Abs(A)+Abs(B))/(A*A+B*B);
M1:=Exp(M1);
M2:=Abs(A+B);
A:=A*B;
if A>0.5
then M:=M1
else if A>0.4        {0.4 < A <= 0.5}
then M:=M2
else begin           {A <= 0.4}
write ('результат не определен');
goto 10
end;
write (' M=',M);
10:;
end.
```

Другой вариант этой же программы:

```
{Разветвляющийся алгоритм – вариант 2}
program Raz;
label 10;
var A,B,M,AB:real;
begin
A:=0.0256; B:=1.39E-02;
AB:=A*B;
if AB>0.5 then begin
M:=-(Abs(A)+Abs(B))/(A*A+B*B);
M:=Exp(M)
end;
if (0.4<AB)and(AB<=0.5) then M:=Abs(A+B);
if AB<=0.4 then begin
write ('M не определена');
goto 10
end;
write (' M=',M);
10:;
end.
```

7.4. Операторы организации циклов. Операторы повтора

Операторы повтора используются при организации циклов. Если количество повторов известно заранее, используется оператор For, если оно неизвестно, применяются операторы Repeat или While.

Оператор повтора FOR состоит из заголовка и тела цикла. Он может быть представлен в двух форматах:

```
for <параметр цикла>:=<s1> to <s2> do <оператор>;
for <параметр цикла>:=<s1> downto <s2> do <оператор>;
```

где s1 и s2 – выражения, определяющие соответственно начальное и конечное значения параметра цикла;

for .. do – заголовок цикла;

<оператор> – тело цикла.

Оператор For обеспечивает выполнение тела цикла до тех пор, пока не будут перебраны все значения параметра цикла от начального до конечного.

Пример:

```
for i:=1 to 20 do write ('*');
```

Параметр цикла, его начальное и конечное значения должны принадлежать к одному и тому же типу. Допустим любой скалярный тип, кроме вещественного.

Значение параметра цикла последовательно увеличивается (при For.. To) или уменьшается (при For .. Downto) на единицу при каждом повторе.

В операторе For PASCAL не допускает изменения параметра цикла на величину, отличную от 1.

Для выхода из цикла до момента достижения параметром цикла конечного значения можно воспользоваться оператором Goto.

В теле оператора For могут находиться другие операторы For.

Оператор повтора Repeat состоит из заголовка (Repeat), тела и условия окончания (Until).

Формат:

```
Repeat
<оператор>;
<оператор>;
...
<оператор>
until <условие>;
```

Условие – выражение булевского типа. При написании условия допустимы булевские операции и выражения. Операторы, заключенные между словами Repeat и Until, являются телом цикла. Вначале выполняется тело цикла, затем проверяется условие выхода из него. Если результат булевского выражения – False, тело цикла активизируется еще раз; если результат – True, – происходит выход из цикла.

Оператор Repeat имеет особенности:

1) выполняется по крайней мере 1 раз;

2) тело цикла выполняется, пока условие равно False;

3) в теле может находиться произвольное число операторов без операторных скобок Begin...End.

По крайней мере, один из операторов тела цикла должен влиять на значение условия, иначе цикл будет выполняться бесконечно.

Пример:

```
D:=1;                    D:=1;
S:=0;                    S:=0;
repeat                   repeat
S:=S+D                   S:=S+D;
until (D<100);           D:=D+1;
{бесконечный цикл}       until (D<100)
                         {цикл имеет завершение}
```

Оператор повтора While аналогичен оператору Repeat, но проверка условия выполнения тела цикла производится в самом начале оператора.

Формат:

while <условие> do <тело цикла>;

Условие – булевское выражение, тело цикла – простой или составной оператор. Перед каждым выполнением тела цикла вычисляется значение выражения условия. Если результат равен True, выполняется тело цикла, и снова вычисляется значение выражения условия. Если результат равен False, происходит выход из цикла и переход к первому после While оператору. Программист сам должен позаботиться об изменении переменных, определяющих условие выхода, иначе цикл получится бесконечным.

Пример:

```
I:=5;
while (I<100) do I:=I+1;
write(I);
```

7.5. Массивы

7.5.1. Определение массива

Массив – это структурированный тип данных, состоящий из фиксированного числа элементов, имеющих один и тот же тип. Элементами массива могут быть данные любого типа, включая

структурированные. Тип элементов массива называется *базовым*. Число элементов массива фиксируется при описании и в процессе выполнения программы не меняется.

Доступ к каждому отдельному элементу осуществляется путем индексирования элементов массива. ***Индексы*** представляют собой выражения любого скалярного типа, кроме вещественного. Тип индекса определяет границы изменения значений индекса.

Для описания массива предназначено словосочетание ***Array of***.

Формат:

```
Type
<имя типа> = array[тип индекса] of <тип компонент>;
var
<идентификатор, ... >: <имя типа>;
```

Массив может быть описан без представления типа в разделе описания типов данных:

```
Var
<идентификатор, ... >: array[тип индекса] of <тип компонент>;
```

Пример:

```
Type
Klass = (k1,k2,k3,k4);
znak  = array[1..25] of char;
var
M1: znak; M2: array[1..50] of integer;
M3: array[1..4] of klass;
M4: array[1..4,1..4] of integer;
```

Если при описании массива задан один индекс, массив называется *одномерным*, если два индекса, – *двумерным*, если n индексов,– *n-мерным*. Размерность ограничена только объемом памяти конкретной ПЭВМ. Одномерные массивы обычно используются для представления векторов, двухмерные – матриц.

Для описания массива можно использовать предварительно определенные константы:

```
Const
G1 = 4;G2 = 6;
var
mass: array[1..G1,1..G2] of real;
```

Элементы массива располагаются в памяти последовательно. Элементы с меньшими значениями индекса хранятся в более низких адресах памяти. Многомерные массивы располагаются таким образом, что самый правый индекс возрастает самым первым. Например, если имеется массив:

```
A: array[1..5,1..5] of integer;
```

то в памяти элементы массива будут размещены по возрастанию адресов:

A[1,1] A[1,2] ... A[1,5] A[2,1] A[2,2] ... A[5,5]

7.5.2. Действия над массивами

Для работы с массивом как с единым целым используется идентификатор массива без указания индекса в квадратных скобках. Массив может участвовать только в операциях отношения "равно", "не равно" и в операторе присваивания. Массивы, участвующие в этих действиях, должны быть идентичны по структуре, то есть иметь одинаковые типы индексов компонентов.

Выражение:	Результат:
A = B	True, если значение каждого элемента массива A равно соответствующему значению элемента массива B
A <> B	True, если хотя бы одно значение элемента массива A не равно значению соответствующего элемента массива B
A := B	Все значения элементов массива B присваиваются соответствующим элементам массива A

7.5.3. Действия над элементами массива

После объявления массива каждый его элемент можно обработать, указав имя массива и индекс элемента в квадратных скобках. Например: Mas[2], Vektor[5]. При работе с двумерными массивами указываются 2 индекса, с n-мерными массивами – n индексов.

Рассмотрим типичные ситуации, возникающие при работе с данными типа Array. Для этого опишем 2 массива и 2 вспомогательные переменные:

```
Var
A,D: array[1..4] of real;
i: integer;
Vs: real;
```

Инициализация массива заключается в присваивании каждому элементу массива одного и того же значения, соответствующего базовому типу.

Пример:

```
A[1] := 0; A[2] := 0; A[3] := 0; A[4] := 0;
```

Удобнее получить тот же результат, используя оператор For:

```
for i:=1 to 4 do A[i]:=0;
```

PASCAL не имеет средств ввода-вывода элементов массива сразу, поэтому ввод и вывод значений производится поэлементно. Значения вводятся с экрана с помощью оператора Read и Readln с использованием оператора For:

```
for i:=1 to 4 do read(A[i]);
```

Можно ввести значения отдельных элементов, а не всего массива:

```
read(A[3]);
```

Вывод значений элементов массива выполняется аналогичным образом, но с использованием операторов Write и Writeln:

```
for i:=1 to 4 do write(A[i]:4:1);
```

Копированием массивов называется присваивание значений всех элементов одного массива всем соответствующим элементам другого массива. Копирование можно выполнить:

```
for i:=1 to 4 do A[i]:= D[i];   или  A:= D;
```

В обоих случаях значение элементов массива D не изменяется, а значения элементов массива A становятся равными значениям соответствующих элементов массива D. Оба массива должны быть идентичны по структуре.

Перестановка значений элементов массива осуществляется с помощью дополнительной переменной того же типа, что и базовый тип массива. Например, требуется поменять значения 1 и 5 элементов массива A (Vs – вспомогательная переменная):

```
Vs:= A[5]; A[5]:= A[1]; A[1]:= Vs.
```

7.5.4. Программирование алгоритмов с использованием массивов

Рассмотрим примеры программ с использованием массивов.

Пример 1. Ввести двумерный массив M размерностью 10x10, содержащий целые числа, и вывести на печать сумму четных элементов этого массива.

```
Program mas1;
const N=10;
var M: array[1..N,1..N] of integer;
I,J,Sm: integer;
begin for I:=1 to N do
for J:=1 to N do read(M[I,J]);
Sm:=0;
for I:=1 to N do
for J:=1 to N do
if odd(M[I,J]) then
else Sm:=Sm+M[I,J];
write ('сумма четных чисел =', Sm:4)
end.
```

Пример 2. Дан массив N размерностью 10x20. Нужно вывести на печать наибольшее из всех значений элементов этого массива, а также номера строки и столбца, на пересечении которых находится данный элемент.

```
Program mas2;
var N: array[1..10,1..20] of real;
maxn: real;
I,J,Im,Jm: integer;
begin for I:=1 to 10 do
for J:=1 to 20 do read(N[I,J]);
maxn:=N[1,1]; Im:=1; Jm:=1;
for I:=1 to 10 do
for J:=1 to 20 do
if N[I,J]>maxn then
begin
maxn:=N[I,J];
Im:=I; Jm:=J
end;
write(' maxn=',maxn:5:1,' Im=',Im,' Jm=',Jm)
end.
```

7.6. Подпрограммы

Подпрограммой называется именованная логически законченная группа операторов языка, которую можно вызвать для выполнения по имени любое количество раз из различных мест программы. В языке PASCAL для организации подпрограмм используются процедуры и функции.

7.6.1. Процедуры и функции

Процедура – это независимая поименованная часть программы, предназначенная для выполнения определенных действий. Она состоит из заголовка и тела. После однократного описания процедуру можно вызывать по имени из последующих частей программы. Когда процедура выполнит свою задачу, программа продолжится с оператора, следующего за оператором вызова процедуры. Имя процедуры не может находиться в выражении в качестве операнда.

Функция аналогична процедуре, но имеет 2 отличия:

1) она передает в точку вызова скалярное значение (результат своей работы);

2) ее имя может входить в выражение как операнд.

Все процедуры и функции языка PASCAL подразделяются на 2 группы:

1) встроенные;

2) определенные пользователем.

Встроенные (стандартные) процедуры и функции языка PASCAL являются частью языка и могут вызываться по имени без предварительного определения в разделе описаний.

Процедуры и функции пользователя организуются самим программистом в соответствии с синтаксисом языка и представляют собой локальный блок. Предварительное описание обязательно.

PASCAL предоставляет программисту широкий набор встроенных процедур и функций, которые реализуют наиболее часто встречающиеся при написании программ действия. В соответствии с областями применения различают ряд основных групп встроенных процедур и функций:

1) арифметические;

2) скалярные;

3) преобразования типов;

4) специальные;

5) обработки строк;

6) обработки файлов;

7) графики.

Арифметические процедуры и функции реализуют наиболее часто встречающиеся в практике математические действия и операции, включая генерацию случайных чисел.

Скалярные функции обрабатывают данные скалярного типа, кроме вещественного.

Функции преобразования типов используются для преобразования значений одного скалярного типа в значения другого скалярного типа.

Процедуры, определенные пользователем.

Процедура представляет собой именованную часть программы, вызываемую при необходимости для выполнения из любой позиции раздела операторов. Описание процедуры включает заголовок и тело процедуры. Заголовок состоит из зарезервированного слова **Procedure**, имени (идентификатора) процедуры и необязательного заключенного в круглые скобки списка формальных параметров с указанием типа каждого параметра.

Формат:

Procedure <имя> {(формальные параметры)};

Пример:

```
Procedure Sort ( A:integer; B:real );
procedure summa ;
```

Имя процедуры – идентификатор, уникальный в пределах программы. Тело процедуры представляет собой локальный блок, по структуре аналогичный программе:

```
Procedure <имя> {(формальные параметры )};
<раздел описаний>
begin
<раздел операторов>
end;
```

Для обращения к процедуре используется ***оператор вызова процедуры***. Он состоит из имени процедуры и списка фактических параметров, отделенных друг от друга запятыми и заключенных в круглые скобки. Список параметров может отсутствовать, если процедуре не передается никаких значений.

Формат:

<имя процедуры> {(параметр, ...)};

Пример:

```
Sort (a1,b1);
summa;
```

Параметры обеспечивают механизм замены, который позволяет выполнять процедуру с различными начальными данными. Количество, порядок следования и тип формальных параметров строго соответствует количеству, порядку следования и типу фактических параметров.

Если процедура возвращает в программу какие-то значения, соответствующие переменные должны быть описаны как параметры переменной с использованием слова Var (см. ниже).

Функции, определенные пользователем.

Функция состоит из заголовка и тела функции. Заголовок содержит зарезервированное слово **Function**, идентификатор (имя) функции, заключенный в круглые скобки необязательный список формальных параметров и тип возвращаемого функцией значения.

Формат:

Function <имя> {(формальные параметры)}: <тип результата>;

Пример:

```
Function prov (X,Y: integer): real;
function Zx: boolean;
```

Имя функции – уникальный в пределах блока идентификатор. Возвращаемый результат может иметь любой скалярный тип, тип String и тип "Указатель". Тело функции представляет собой локальный блок, по структуре аналогичный программе.

```
Function <имя> {(формальные параметры)}: <тип результата>;
<раздел описаний>
begin
<раздел операторов>
end;
```

В разделе операторов должен находиться по крайней мере один оператор, присваивающий идентификатору функции значение. Если таких присваиваний несколько, результатом выполнения функции будет значение последнего оператора присваивания.

Обращение к функции осуществляется по имени с обязательным указанием списка аргументов. Каждый аргумент должен соответствовать формальным параметрам, указанным в заголовке, и иметь тот же тип.

7.6.2. Параметры

Параметры могут иметь любой тип, включая структурированный. PASCAL поддерживает 2 различных метода передачи параметров:

1) по значениям;
2) по ссылкам.

При передаче по значениям формальный параметр является переменной, локальной в блоке. Фактический параметр может быть любым выражением того же типа, что и соответствующий ему формальный. Такие параметры называются ***параметрами-значениями***. Их главная отличительная черта – то, что изменение формальных параметров не влечет за собой изменения фактических. Приведем ***пример*** типичной записи параметров-значений в описании процедур и функций:

```
Procedure konc (A,B,C: integer; D: real);
function docs (S1,S2: real): real;
```

При передаче по ссылкам фактический параметр является переменной. Формальный параметр обозначает эту фактическую переменную в течение всего времени активизации блока. Такие параметры называются ***параметрами-переменными***. Их характерный признак – то, что любое изменение формального параметра означает изменение фактического. Для описания параметров-переменных в секции формальных параметров служит зарезервированное слово **Var**.

При использовании параметров-переменных формальные и фактические параметры должны совпадать по количеству и типу. Возможны два исключения:

1) можно передавать значение параметров строкового типа, не совпадающее по длине с описанной в подпрограмме переменной; для этого директиву компилятора V надо перевести из активного состояния по умолчанию {$V+} в пассивное состояние {$V-};

2) допускается отсутствие типа формального параметра в секции формальных параметров заголовка подпрограммы.

Пример:

```
Procedure (var F1);
```

Формальные параметры без типа несовместимы ни с каким типом. Допустимо сочетание параметров-значений и параметров-переменных в одной секции формальных параметров:

```
Procedure dok (var A,B,C: real; S: integer);
```

Пример. Функция предназначена для вычисления tg(x), а подпрограмма – для вычисления суммы n-членов арифметической прогрессии по следующей формуле:

$$Sn = (n+1)*(A0+An)/2,$$

где A0 и An – первый и последний члены арифметической прогрессии.

```
Program test;
var S, X, A0, An: real;
N: integer;
function Tg(X: real): real;     {Функция}
begin
Tg:=Sin(x)/Cos(x);
end;
procedure Sumar (A0, An: real; N: integer; var Sn: real);   {Подпрограмма}
begin
Sn:=(n+1)*(A0+An)/2;
end;
begin
writeln('Введите значение x');
readln(X);
writeln('Значение Tg(x)=', Tg(x):8:5);
writeln('Введите значение A0');
readln(A0);
writeln('Введите значение An');
readln(An);
writeln('Введите значение N');
readln(N);
sumar(A0, An, N, S);
writeln('Сумма арифметической прогрессии при A0=', +A0,
' An=', An,' N=', N,' равна', S:8:1);
end.
```

Эта простая программа запрашивает число x и вычисляет tg(x), затем требует ввести три числа: первый и последний члены арифметической прогрессии и количество всех ее членов. На основании введенных значений вычисляется сумма арифметической прогрессии.

Содержание