Министерство образования Республики Беларусь
БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Кафедра «Электропривод и автоматизация промышленных установок
и технологических комплексов»

С.Н.Павлович

ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ ИНЖЕНЕРНЫХ ЗАДАЧ НА ПАСКАЛЕ

Учебно-методическое пособие
по дисциплине «Информатика»
для студентов специальности 1-53 01 05
«Автоматизированные электроприводы»
заочной формы обучения

Учебное электронное издание

Минск 2010

УДК 004(076.5)
ББК 32.81
П 12

А в т о р :
С.Н. Павлович

Р е ц е н з е н т ы :
В.П. Беляев, доцент кафедры «Полиграфическое оборудование и системы обработки информации» БГТУ, кандидат технических наук, доцент;
А.А. Гончар, доцент кафедры «Электроснабжение» БНТУ, кандидат технических наук, доцент

В пособии приведены теоретические сведения и методика программирования на алгоритмическом языке Паскаль отдельных блоков и структур алгоритмов решения инженерных задач. Рассмотрены схемы типовых алгоритмов, встречающихся при программировании инженерных задач.

Пособие предназначено для студентов заочной формы обучения по специальности 1-53 01 05 «Автоматизированные электроприводы». Оно может быть использовано и студентами дневной формы обучения.

Белорусский национальный технический университет
пр-т Независимости, 65, г. Минск, Республика Беларусь
Тел.(017)292-77-52 факс (017)292-91-37
Регистрационный № БНТУ/ФИТР46-11.2010

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В настоящее время в обучении все шире используется **модульный подход**, в основе которого лежат такие основные принципы как системность, структуризация и модульность. Понятие «**модуль**» в рамках одной учебной дисциплины означает **блок информации**, *включающий в себя логически завершенную одну, две или более единиц учебного материала.*

При модульном подходе обучения учебная программа дисциплины (или ее отдельные разделы, темы) представляются в виде определенного количества конкретных модулей, с которыми студент может работать самостоятельно или под руководством преподавателя. При этом *каждый модуль – это автономный учебный материал, предназначенный для освоения некоторых элементарных единиц знаний или умений.*

В данном учебно-методическом пособии приведены теоретические сведения и методика программирования на Паскале отдельных блоков и структур алгоритмов решения инженерных задач в виде следующих восьми модулей дисциплины «Информатика»:

М1 – Алгоритмизация инженерных задач;

М2 – Базовые элементы алгоритмического языка Паскаль;

М3 – Программирование линейных алгоритмов;

М4 – Программирование разветвляющихся алгоритмов;

М5 – Программирование циклических алгоритмов;

М6 – Программирование алгоритмов с использованием массивов данных;

М7 – Программирование алгоритмов с использованием подпрограмм;

М8 – Методика программирование алгоритмов сложных задач.

Модуль М1 – «Алгоритмизация инженерных задач»

1.1. Методика программирования и решения инженерных задач на ПК

Методика подготовки (программирования) и решения инженерных задач на ПК включает в себя выполнение следующих основных этапов:

1. Математическую формулировку задачи.
2. Выбор метода вычисления.
3. Разработку схемы алгоритма решения задачи.
4. Написание программы на алгоритмическом языке.
5. Ввод текста программы в память ПК и отладку ее.
6. Ввод исходных данных и выполнение вычислений по программе.

На ***первом этапе*** условие задачи записывается в виде последовательности формул, которые в дальнейшем будут использоваться в вычислениях.

На ***втором этапе*** выбирается такой метод решения математически сформулированной задачи, который позволяет свести поиск результата к выполнению последовательности элементарных математических операций. Для большинства практических задач разработаны численные методы, обеспечивающие приближенное решение с требуемой точностью.

На ***третьем этапе*** на основе выбранного метода разрабатывается схема алгоритма решения задачи.

На ***четвертом этапе,*** используя схему алгоритма, составляют программу решения задачи на определенном алгоритмическом языке в виде последовательности соответствующих операторов.

На ***пятом этапе*** с помощью системы программирования вводят программу в память ПК, транслируют, редактируют, проверяют правильность ее написания и ввода.

На ***шестом этапе*** вводят исходные данные и выполняют вычисления по программе.

1.2. Разработка алгоритма решаемой задачи

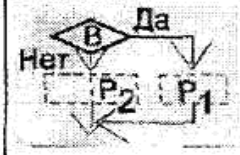
Под ***алгоритмом*** понимается определенная последовательность предписаний (инструкций, действий, операций) по переработке исходных и промежуточных данных в результат решения задачи, т.е. *алгоритм* представляет собой общую схему решения конкретной задачи. При разработке алгоритмов сложных задач выделяют составные части, различные по назначению. Каждая часть может представлять собой отдельный алгоритм. Иногда этот алгоритм можно разбить на ряд еще более простых алгоритмов и т.д. Глубина разработки алгоритма (детализация) зависит от наличия разработанных ранее стандартных алгоритмов решения типовых задач и от условий решаемой задачи. В итоге алгоритм решения задачи сводится к определенной последовательности таких действий: начало решения, ввод данных, вычисления по формулам, обращение к подпрограммам (другим мини-программам), проверка условий, влияющих на ход вычислительного процесса, организация повторяющихся участков вычислений, вывод данных (исходных, промежуточных, результатов решения задачи), конец решения.

Из различных способов описания алгоритмов (словесный, операторный, схемный) наиболее распространен (из-за большей наглядности, облегчающий затем написание по алгоритму самой программы на каком-нибудь алгоритмическом языке) ***схемный способ,*** при котором алгоритм представляется в виде ***символов*** (блоков), выполняющих определенные действия, и связей между ними с помощью ***линий,*** указывающих очередность выполнения этих символов при вычислениях. Форму, размеры, наименование символов и выполняемые ими функции, а также правила построения схем алгоритмов определяет

ГОСТ 19.701-90 [2]. В табл. 1.1 приведены основные, чаще употребляемые символы схем алгоритмов и форма их записи (программирование в общем формате) на алгоритмическом языке Паскаль.

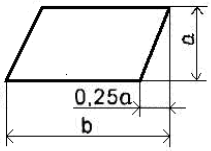
Таблица 1.1

Некоторые символы и форма их записи на Паскале

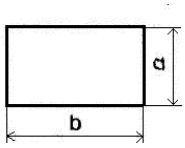
Название и форма обозначения символа	*Выполняемая функция*	*Общая форма записи символа на Паскале*	*Примечание*
1	2	3	4
Терминатор Начало или Конец	Начало выполнения программы Конец программы	Заголовок программы, описательные разделы и операторная скобка Begin раздела операторов End.	
Данные Ввод:S или Вывод:S	Ввод данных Вывод данных	Read(S); или ReadLn(S); Write(S); или WriteLn(S);	S – список вводимых переменных S – список выводимых элементов
Процесс a=b	Выполнение вычислений по формуле	a:=b;	a – имя переменной; b – арифметическое или логическое выражение
Решение 	Выбор дальнейших вычислений в зависимости от значения B	If B then P_1; или If B then P_1 else P_2;	B – логическое выражение P_1, P_2 – операторы (простые, составные или вложенные)
Подготовка 	Начало цикла с известным числом повторения и шагом изменения параметра цикла *i*, равным 1	For i:=n_1 to n_2 do p; или (если $n_1 > n_2$) For i:=n_1 downto n_2 do p;	i – параметр цикла n_1, n_2 – начальное и конечное значение i p – тело цикла (простой, составной или вложенный оператор)

Приведем условные обозначения чаще используемых символов (блоков) и некоторые правила выполнения схем алгоритмов из ГОСТа 19.701-90.

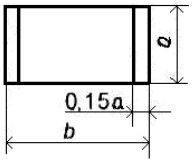
1. Данные. Символ отображает данные, носитель данных не определен (символы данных во многих случаях представляют способы ввода/вывода данных).

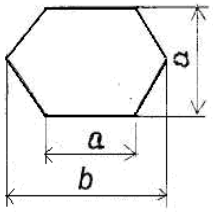


2. Процесс. Символ отображает обработку данных (вычисления по формулам).

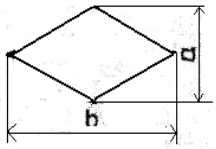


3. Предопределенный процесс. Символ отображает подпрограмму, модуль.

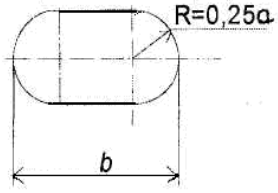




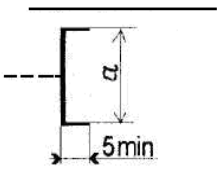
4. Подготовка. Символ отображает модификацию команды или группы команд с целью воздействия на некоторую последующую функцию (начало цикла с заданным числом повторения).



5. Решение. Символ отображает выбор направлений выполнения алгоритма в зависимости от некоторых условий.



6. Терминатор. Символ отображает выход во внешнюю среду и вход из внешней среды (начало или конец схемы программы), прерывание (остановка, пуск).



7. *Линия.* Поток данных или управления.

8. Комментарий. Символ отображает связь между элементом схемы и пояснением (используется, если текст внутри символа не помещается).

Символы (блоки) в схеме алгоритма должны быть расположены равномерно, быть по возможности одного размера; *не должны изменяться углы и другие параметры, влияющие на соответствующую форму символов* (размер ***a*** выбирается из ряда 10, 15, 20, … мм, а размер $\boldsymbol{b = 1{,}5a}$ или $\boldsymbol{2a}$ согласно ГОСТ 19.003-80).

Помещаемый внутри символа текст записывается слева направо и сверху вниз независимо от направления потока выполнения символов. Текст должен быть минимальным и достаточным для понимания выполняемых функций данного символа; если объем текста превышает размеры символа, то следует использовать символ комментария.

Для большей ясности на линиях потока в схеме алгоритма используются стрелки. Направление линий потока (данных или управления) слева направо и сверху вниз считается ***стандартным*** и стрелкой может не обозначаться. Линии, отличные от стандартного направления или имеющие изломы, должны обязательно обозначаться стрелками.

В схемах алгоритмов следует избегать пересечения линий. Пересекающиеся линии не имеют логической связи между собой, поэтому изменение направления в точках пересечения не допускается. Две или более входящих линий могут объединяться в одну исходящую линию, при этом место объединения должно быть смещено (рис. 1.1).

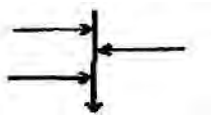


Рис. 1.1

Линии в схемах алгоритмов должны подходить к символу либо слева, либо сверху, а исходить либо справа, либо снизу (по центру символа).

Несколько выходов из символа (рис. 1.2) можно показывать:

- несколькими линиями от данного символа к другим символам;

- одной линией от данного символа, которая затем разветвляется на соответствующее число линий. Каждый выход из символа должен сопровождаться соответствующим значением условия разветвления.

Пример.

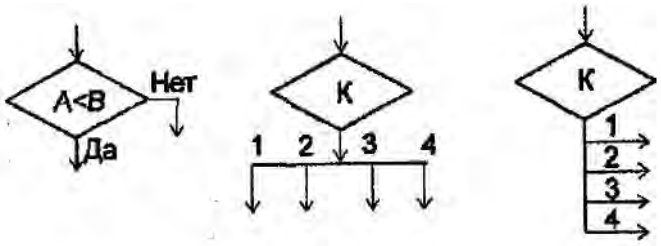


Рис. 1.2

Символы в схеме алгоритма могут помечаться идентификаторами, например, для ссылки на них в других частях документации. Идентификатор располагается слева над символом.

При разработке схемы алгоритма и написании Паскаль-программы целесообразно использовать некоторые простые приемы, позволяющие уменьшить затраты времени на вычисления (при решении задачи по программе):

- вместо операции возведения в целую степень (для низких степеней) использовать операцию умножения (например, $a^3 = a \cdot a \cdot a$);

- выражение, вычисляемое в программе многократно с одними и теми же данными, необходимо вычислять один раз, присваивая его значение промежуточной переменной, используемой затем в дальнейших вычислениях как известной;

- в качестве границ параметров цикла использовать не выражения, а промежуточные переменные, вычисляемые до начала выполнения цикла;

- все повторяющиеся внутри цикла вычисления с одинаковыми данными выносить за пределы цикла, выполняя их один раз до начала цикла.

Рассмотрим в следующем разделе некоторые характерные приемы алгоритмизации типовых задач на конкретных примерах.

1.3. Стандартные схемы алгоритмов

1.3.1. Линейный алгоритм

Линейным называется алгоритм, в котором все блоки выполняются последовательно один за другим. При разработке алгоритма решаемой задачи следует выбирать наилучший вариант в смысле некоторого критерия, в качестве которого используют либо оценку точности решения задачи, либо затраты времени на решение, либо некоторые интегральные критерии.

Пример.

Разработаем линейный алгоритм для определения высот: h_a, h_b , h_c треугольника по заданным длинам его сторон *a, b, c:*

$$h_a = 2 / a\sqrt{p\cdot(p-a)\cdot(p-b)\cdot(p-c)}\ ;$$

$$h_b = 2 / b\sqrt{p\cdot(p-a)\cdot(p-b)\cdot(p-c)}\ ;$$

$$h_c = 2 / c\sqrt{p\cdot(p-a)\cdot(p-b)\cdot(p-c)}\ ,$$

где $p = (a + b + c)/2$.

При решении данной задачи, с целью уменьшения затрат на вычисления, определять высоты будем не по приведенным выше формулам, а с использованием промежуточной переменной:

$$x = 2\sqrt{p\cdot(p-a)\cdot(p-b)\cdot(p-c)}\ .$$

Тогда $h_a = x/a$; $h_b = xlb$; $h_c = x/c$ и блок-схема алгоритма решения данной задачи будет иметь вид, представленный на рис. 1.3.

Рис. 1.3

1.3.2. Разветвляющийся алгоритм

На практике часто, в зависимости от исходных данных или от результатов промежуточных вычислений, бывает необходимо организовать дальнейшие вычисления по одним или другим формулам, т.е. вычислительный процесс в зависимости от выполнения некоторых логических условий должен идти по нескольким направлениям (ветвям). Алгоритм такого вычислительного процесса называют ***разветвляющимся.***

Пример.

Разработаем разветвляющийся алгоритм вычисления корней квадратного уравнения $ax^2 + bx + c = 0$ с вещественными коэффициентами ***a, b , c***.

Алгоритм решения данной задачи (рис. 1.4) содержит три ветви (в зависимости от значения дискриминанта $d = b^2 - 4ac$):

1. Если $d > 0$, то корни вещественные и определяются по формулам:

$$x_1 = (- b + \sqrt{d}\ ) / (2a); \; x_2 = (-b - \sqrt{d}\ ) / (2a). \qquad (1.1)$$

2. Если $d < 0$, то корни комплексно-сопряженные вида $x_1 \pm jx_2$, а их вещественная (x_1) и мнимая (x_2) части вычисляются по формулам:

$$x_1 = - b /(2a); \quad x_2 = \sqrt{-d}\ /(2a). \qquad (1.2)$$

3. Если d = 0, то корни вещественные и равные между собой:

$$x_1 = x_2 = - b /(2a).$$

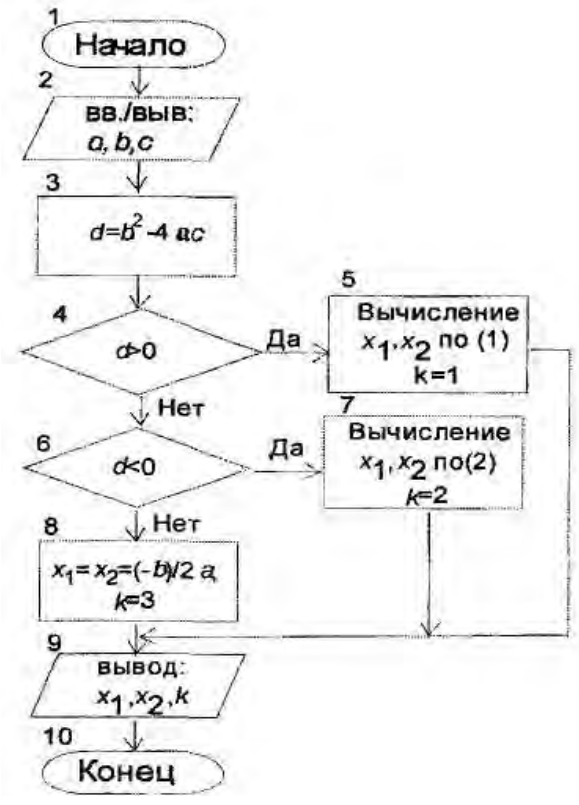


Рис. 1.4

В данном алгоритме вид корней при выводе результатов вычисления определяется значением переменной ***к:*** если ***к*** = 1, то корни вещественные и разные; если ***к*** = 2, то корни комплексно-сопряженные; если ***к*** = 3, то корни вещественные и равные между собой.

1.3.3. Циклические алгоритмы

Циклическим алгоритмом называют вычислительный процесс, в котором решение задачи или ее части сводится к многократному вычислению по одним и тем же формулам при различных значениях входящих в них некоторых переменных. Многократно повторяющиеся участки такого вычислительного процесса называются ***циклами.***

Различают циклы с заданным (известным) и неизвестным числом повторений. К последним относятся так называемые ***итерационные циклы,*** характеризующиеся последовательным приближением к искомому значению результата с заданной точностью (например, при вычислении суммы членов сходящегося бесконечного ряда чисел).

Переменную, изменяющуюся в цикле, называют ***параметром цикла.***

В одном цикле могут быть один или несколько параметров. *Цикл с несколькими одновременно изменяющимися параметрами* организуется по схеме построения цикла с одним каким-либо параметром, а для остальных параметров перед началом цикла задаются их начальные значения, а внутри цикла вычисляются их текущие значения для очередных повторений цикла.

Пример 1.

Составим схему алгоритма по вычислению функции ***S***, равной сумме членов y_i, т.е.

$$S = \sum_{i=1}^{20} y_i \text{ , } yi = x_i^2 / i; \; x_i - \text{элементы массива } (x_1, x_2, \ldots, x_{20}).$$

Эта задача относится к циклическому вычислительному процессу с заданным числом повторения цикла по накапливанию суммы членов y_i.

Схема алгоритма решения данного примера представлена на рис. 1.5, в котором блок 3 задает начальное значение суммы *S* перед циклом, а блок 5 вычисляет значение слагаемых y_i и накапливает искомую сумму.

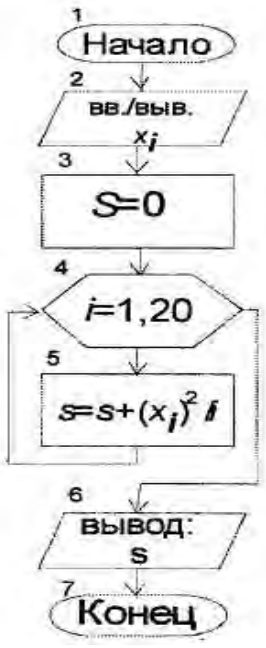


Рис. 1.5

Примечание. Алгоритм по вычислению функции ***P***, равной произведению членов y_i , т.е.

$$P = \prod_{i=1}^{20} (x_i^2 / i)$$

аналогичен алгоритму на рис. 1.5 с той лишь разницей, что для накапливания произведения будет использоваться в блоке 5 формула $P = P \cdot y_i$, а начальное значение произведения P в блоке 3 должно быть равно единице.

Пример 2.

Разработаем алгоритм по вычислению суммы членов сходящегося бесконечного ряда чисел**:**

$$S = x + x^2/2! + x^3/3! + \ldots = \sum_{n=1} (x^n / n!)$$

с точностью до члена ряда, меньшего e.

Здесь имеет место итерационный цикл, так как заранее не известно, при каком n выполняется условие $(x^n/n!) < e$.

Сравнивая два соседних члена ряда, видим, что $y_n / y_{n-1} = x / n$.

Поэтому для уменьшения затрат времени на вычисление текущего члена ряда целесообразно в цикле использовать ***рекуррентную*** формулу: $y_n = y_{n-1} \cdot x / n$, т.е. формулу, использующую при вычислении члена y_n уже вычисленное значение предыдущего члена y_{n-1} . А чтобы использовать эту формулу для вычисления первого члена ряда $y_1 = y_0 \cdot x / 1$, необходимо положить начальное значение $y_0 = 1$. Тогда схема алгоритма решения данного примера будет иметь вид, представленный на рис. 1.6.

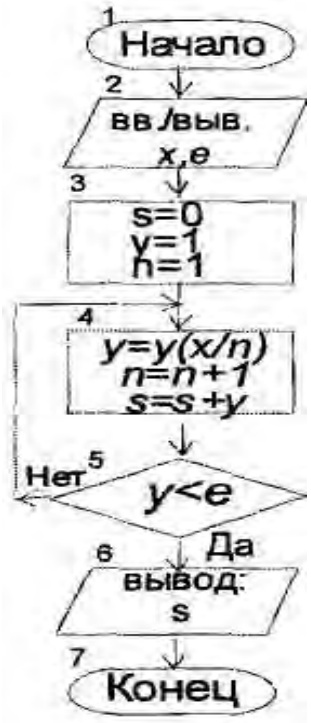


Рис. 1.6

Пример 3.

Разработаем алгоритм по вычислению функции $z_i = x \cdot y_i$, если x изменяется одновременно с i от начального значения a с шагом h, а y_i - элементы массива $(y_1, y_2, \ldots, y_{20})$.

Здесь в цикле, повторяемом 20 раз, изменяются одновременно 2 параметра: простая переменная x и i - индекс массива y. Схема алгоритма решения данной задачи представлена на рис. 1.7, где блок 4 задает закон изменения параметра i , блок 3 (перед циклом) - начальное значение параметра x, а блок 5 (внутри цикла) вычисляет (кроме z_i) новое значение параметра x для очередного выполнения цикла.

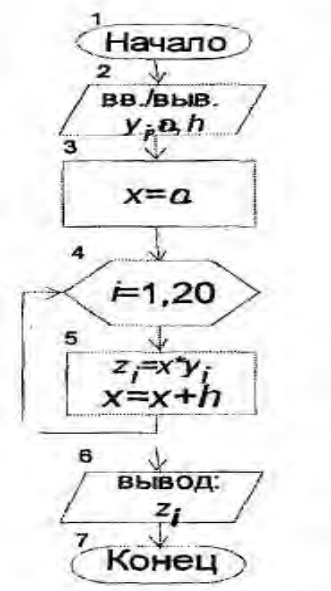


Рис. 1.7

Пример 4.

Разработаем алгоритм по вычислению суммы положительных элементов каждой строки матрицы ***a (10 ·15).***

Для вычисления суммы элементов одной строки заданной матрицы необходимо организовать цикл с целью перебора всех элементов строки, поэтому параметром этого цикла следует выбрать номер столбца ***к.*** Перед циклом нужно задать начальное значение суммы $S_i = 0$.

Для вычисления суммы положительных элементов каждой строки матрицы необходимо организовать другой цикл (с параметром номера строки ***i)***, охватывающий первый цикл. Здесь мы имеем алгоритм со структурой вложенного цикла. ***Вложенным*** называется цикл, содержащий внутри себя один или несколько других циклов, при этом цикл, охватывающий другие циклы, называется ***внешним***, а остальные циклы - ***внутренними.*** Параметры этих циклов изменяются не одновременно, так как при одном каком-либо значении параметра внешнего цикла параметр внутреннего цикла принимает по очереди все свои значения. Схема алгоритма данного примера приведена на рис. 1.8.

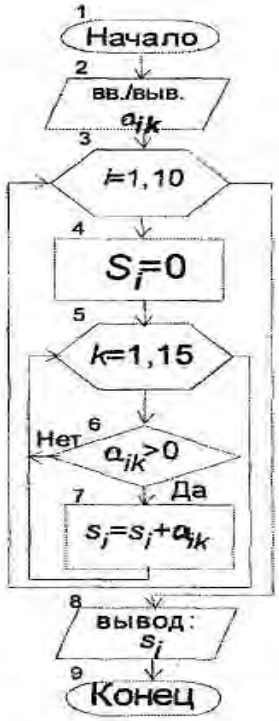


Рис. 1.8

1.3.4. Вычисление полинома

Для вычисления полинома n-й степени

$$y = a_1x^n + a_2x^{n-1} + ... + a_nx + a_{n+1}$$

удобно использовать формулу Горнера:

$$y = ((a_1x + a_2)x + ... + a_n)x + a_{n+1}.$$

Если выражение в самых внутренних скобках обозначить y_i, то значение выражения в следующих скобках можно вычислять по рекуррентной формуле: $y_{i+1} = y_ix + a_{i+1}$, а значение полинома *y* получим после повторения этого процесса в цикле ***n*** раз. Начальное значение y_i целесообразно взять равным a_1, а цикл начать с **i = 2**. Тогда алгоритм для вычисления полинома ***n-й*** степени по формуле Горнера можно представить в виде схемы рис. 1.9. Все коэффициенты полинома сводятся в массив из **n+1** элементов, при этом следует иметь в виду, что если полином не содержит членов с некоторыми степенями ***x***, то в массиве коэффициентов a_i на соответствующих местах необходимо помещать коэффициенты, равные нулю.

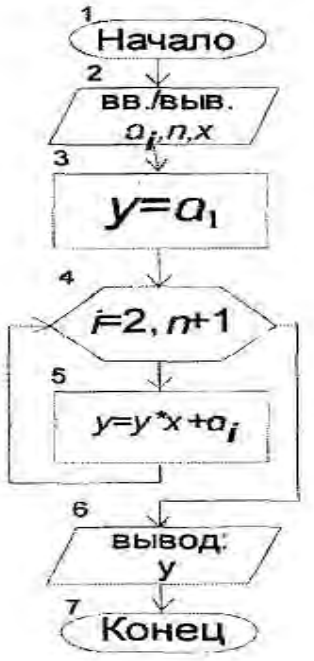


Рис. 1.9

1.3.5. Нахождение наибольшего или наименьшего значения функции

Нахождение наибольшего (или *наименьшего*) значения функции $y = f(x)$ выполняется по циклу, в котором вычисляется текущее значение функции и сравнивается с наибольшим (или с *наименьшим*) из всех предыдущих значений этой функции. Если текущее значение функции оказывается больше из наибольшего (или *меньше из наименьшего*) из предыдущих значений, то его считают новым наибольшим (или *наименьшим*) значением.

При этом в качестве начального значения y_{max} необходимо взять очень малое значение (например, число - $1 \cdot 10^{10}$), а в качестве начального значения y_{min} – очень большое число (например, + $1 \cdot 10^{10}$).

Пример.

Разработаем алгоритм по нахождению наибольшего элемента массива x_i $(x_1, x_2, ..., x_{20})$ и его порядкового номера.

Здесь нет необходимости вычислять сравниваемые значения, так как они уже имеются в массиве x_i . Поэтому в качестве начальных значений x_{max} и номера n_{max} примем $x_{max} = x_1$ и $n_{max} = 1$ и сравнение будем производить по циклу, начиная со второго элемента массива x_i.

Схема алгоритма решения данного примера представлена на рис. 1.10.

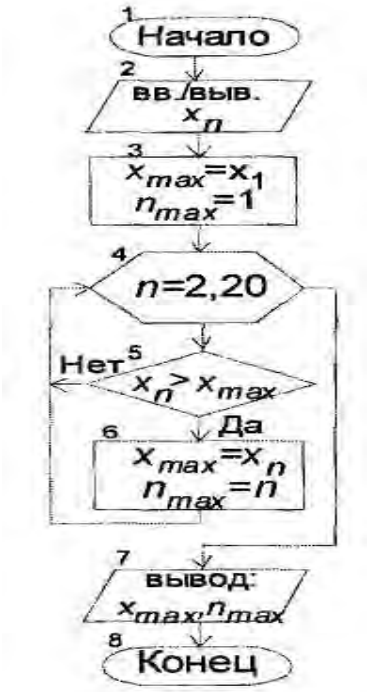


Рис. 1.10

Модуль М2 – «Базовые элементы алгоритмического языка Паскаль»

2.1. Вводные сведения о Паскале и системе программирования Турбо-Паскаль

Алгоритмический язык Паскаль является одним из популярных и широко распространённых языков программирования инженерных задач. Впервые его описание было опубликовано в 1971 г. создателем этого языка Никлаусом Виртом – профессором Цюрихского технологического института (Швейцария). Название язык получил в честь французского математика и философа Блеза Паскаля (1623-1662). Первый транслятор с языка Паскаль был разработан в 1973 г. Этот язык лёгок в изучении и удобен для программирования (набор его операторов относительно мал), является наиболее совершенным по сравнению с Бейсиком, Фортраном и др. языками и в 1982 г. утверждён в качестве международного стандарта.

В начале 80-х годов появилась первая версия языка Паскаль как составная часть системы программирования для ПК. В настоящее время одной из наиболее популярных систем программирования для ПК является Турбо-Паскаль (ТП), представляющая собой интегрированную среду и включающая: экранный редактор, компилятор, редактор связей и отладчик. Интегрированность среды проявляется не только в единой концепции построения её составляющих частей, но и в связи их друг с другом. Так, при возникновении ошибки трансляции система ТП автоматически переходит в режим экранного редактирования и ставит курсор в точку возникновения ошибки. Аналогичные действия выполняются отладчиком при возникновении ошибки во время выполнения программы.

Система программирования ТП создана американской фирмой Borland International. В первой версии этой системы был объединён очень быстрый компилятор с редактором текста. В 1985г. на рынке ПК появилась версия 3.0 системы программирования ТП с компилятором стандартного Паскаля, которая благодаря простоте её использования получила широкое применение. В пакете ТП версии 4.0 было устранено большинство подвергавшихся критике ограничений компилятора и была повышена производительность системы программирования, что дало возможность разработки в этой системе крупных программных продуктов. В ТП версии 5.0 в среду программирования был встроен интегрированный отладчик, позволяющий повысить производительность труда программистов. В версии ТП 5.5 улучшены технические характеристики: наряду с внутренними улучшениями реализована концепция объектно-ориентированного программирования. В ТП версии 6.0 чисто теоретическая концепция объектно-ориентировочного программирования реализована практически с полным набором объектов, которые могут использоваться для решения прикладных задач. Кроме того, реализация системы меню приведена в соответствие со стандартом SAA (Turbo Vision); реализован текстовый редактор, встроенный в IDE (интегрированную инструментальную оболочку). В 1992 г. фирмой Borland International разработана была очередная версия 7.0 системы программирования ТП, в которой унаследованы преимущества предыдущей версии и произведены некоторые изменения и улучшения: появилась возможность выделять определённым цветом различные элементы исходного текста программы (ключевые слова, идентификаторы, числа и т.д.), улучшен компилятор (коды программ стали более эффективными), улучшен интерфейс пользователя и др.

2.2. Базовые элементы языка Паскаль

Алфавит и словарь. При записи алгоритма решаемой задачи на языке Паскаль используется конечный набор знаков (символов), образующих ***алфавит*** этого языка.

Он состоит из

прописных и строчных ***букв*** *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

знака подчёркивания _

цифр **0 1 2 3 4 5 6 7 89**

и *специальных символов*

+	плюс	{ }	фигурные скобки
-	минус	**.**	точка
*****	звёздочка	**,**	запятая
/	дробная черта	**:**	двоеточие
=	равно	**;**	точка с запятой
>	больше	'	апостроф
<	меньше	#	номер
[ ]	квадратные скобки	**$**	номер денежной единицы
( )	круглые скобки	^	тильда
_	пробел		

Комбинации специальных символов образуют *составные символы:*

: =	присвоить	<=	меньше или равно
<>	не равно	>=	больше или равно
..	диапазон значений	**(. .)**	альтернатива квадратных скобок

(* *) альтернатива фигурных скобок.

В комментариях к программе и в символьных константах могут использоваться любые другие знаки, например буквы русского алфавита. Комментарий представляется следующей конструкцией: **{Любой текст}** и может быть помещен в любом месте программы.

Неделимые последовательности знаков алфавита образуют *слова,* которые несут определённый смысл в программе и отделяются друг от друга *разделителями* (пробелом, символом конца строки, комментарием). Слова делятся на ключевые (зарезервированные) слова, стандартные идентификаторы и идентификаторы пользователя.

Ключевые слова имеют фиксированное написание и определённый смысл.

Идентификатором *называется последовательность букв и цифр, начинающаяся с буквы или знака подчёркивания.* Идентификаторы применяются для обозначения имён переменных, констант, функций и процедур (подпрограмм).

Стандартные идентификаторы - это обозначения заранее определённых разработчиками языка Паскаль типов данных, констант, процедур и функций. Например, стандартный идентификатор **sin(x)** вызывает функцию по вычислению синуса заданного угла **x.**

Идентификаторы пользователя применяются для обозначения меток, констант, переменных, процедур и функций, которые задаются самим пользователем. Например, дату удобнее обозначать идентификатором **Data,** чем просто буквой **D** или каким-либо другим символом. Существуют *общие правила написания идентификаторов пользователя:*

- идентификатор должен начинаться только с буквы или знака подчёркивания, исключение, составляют метки, которые могут начинаться также и с цифры;

- для обозначения идентификаторов используются только буквы, цифры и знак подчёркивания;

- между двумя идентификаторами должен быть, по крайней мере, один пробел;

- идентификаторы могут быть любой длины (до 126 символов), но сравнение их между собой производится по первым 63 символам;

- при написании идентификаторов можно использовать как прописные, так и строчные буквы, так как компилятор не делает различий между ними. Поэтому на практике для более простого чтения и понимания целесообразно, например, написать ***NomerOtdela*** (вместо ***nomerotdela),*** выделив прописными буквами каждую из двух смысловых частей этого идентификатора.

Константы и переменные. При решении любой задачи по программе используются конкретные данные, над которыми выполняются определённые действия для получения результата решения. В программе каждый элемент данных является или константой, или переменной.

Константами *называются элементы данных, значения которых заданы перед решением задачи и которые в процессе выполнения программы не изменяются. Формат их описания:*

Const *идентификатор = значение константы*;

Имеется ряд стандартных констант, к значениям которых можно обращаться без предварительного описания, например:

Maxint = 32767;

True = истинно;

False = ложно;

Pi =3,14159…

Переменная *- это именованный объект, который в процессе выполнения программы может принимать различные значения.*

Переменные имеют следующий формат описания:

Var идентификаторы: *тип;*

В Турбо-Паскале используются три вида констант:

* числовые (целые или вещественные);

* логические (или булевские);

* символьные и строковые.

Целые константы *– это положительные или отрицательные целые числа (без десятичной точки).* Знак «+» в положительных числах можно опускать.

Турбо-Паскаль позволяет использовать также шестнадцатеричные целые значения. При использовании шестнадцатеричной константы перед ее значением указывается знак доллара **$**. Например, **$27** определяет десятичное число **39**.

Вещественные константы могут быть представлены в двух формах: с фиксированной и плавающей точкой. *Константы с фиксированной точкой* – это числа, содержащие точку, разделяющую целую и дробную части. *Константы с плавающей точкой* – это числа, представленные с десятичным порядком **mEp** (без пробелов), где **m** – мантисса (как целые, так и вещественные числа с фиксированной точкой); **E** – признак записи числа с десятичным порядком; **p** – порядок числа (только целые числа).

Пример.

Значение константы	Целая константа	Константа с фиксированной точкой	Константа с плавающей точкой
-257	-257	-257.0	-2.57E2
16,4	--	16.4	1.64E1
0,0032	--	0.0032	0.32E-2

Константы с фиксированной точкой обязательно должны содержать как целую, так и дробную часть: 2.0; 0.5; -13.0.

Логические константы *- это константы, которые принимают только два значения: True (истинно) или False (ложно).*

Символьные константы *- это какой-либо один символ, заключенный в апострофы: 'A', '1'.*

Строковые константы *- это ряд символов, заключенных в апострофы: '+9CL', 'A и B'.*

Кроме констант и переменных существуют ещё так называемые ***типизированные константы,*** которые являются промежуточными данными между константами и переменными. Слово *константа* означает, что типизированная константа описывается в разделе **Const,** а слово *типизированная* - что должен указываться ***тип,*** как у переменных. Формат описания их следующий:

Const идентификатор: ***тип*** = значение;

Типы данных. Константы, переменные, функции, выражения, которые используются в программе, относятся к определённому типу. ***Тип*** – это множество значений элементов программы и совокупность операций над ними. Например, значения 1 и 5 относятся к целочисленному типу, над ними можно выполнять различные арифметические операции.

В Паскале типы данных делятся на *скалярные* (стандартные и пользовательские) и *структурированные* (содержащие различные комбинации скалярных типов).

К стандартным скалярным типам данных относятся: целочисленные, байтовые, вещественные, символьные и булевские типы.

Целочисленный тип данных включает все целые числа в диапазоне от -32768 до +32767. Для размещения в памяти значения переменной целочисленного типа требуется 2 байта. Формат описания целочисленных переменных следующий:

Var идентификаторы: *integer*;

Байтовый тип данных аналогичен целочисленному, но охватывает меньший диапазон значений: от нуля до 255. Переменная байтового типа занимает в памяти 1 байт и имеет следующий формат описания:

Var идентификаторы: *byte*;

Вещественный тип данных включает положительные и отрицательные числа от $1 * 10^{-38}$ до $1 * 10^{+38}$, при этом мантисса может содержать до 11 значащих цифр. Данные этого типа могут записываться с *фиксированной* (целая часть числа от дробной отделяется точкой) и *плавающей точкой* в виде **mE ± p,** где **m** - мантисса, представляющая собой целое или дробное число с десятичной точкой; **E** означает «десять в степени»; **p** - порядок в виде двухзначного целого числа.

Переменная вещественного типа в памяти занимает 6 байт и имеет следующий формат описания:

Var идентификаторы: *rea1*;

Переменная *булевского типа* в памяти занимает 1 байт, может принимать одно из двух значений (констант): **True** (истина) или **False** (ложь) и имеет следующий формат описания:

Var идентификаторы: *boolean*;

Константы и переменные ***символьного (литерного) типа*** принимают одно из значений кодовой таблицы ПК. Переменная этого типа в памяти занимает 1 байт. Конкретные значения переменных и констант символьного типа записываются в апострофах. Например, 'A' представляет собой букву A, ';' - точку с запятой. Формат описания переменной символьного типа следующий:

Var идентификаторы: *char;*

К скалярным пользовательским типам данных относятся перечисляемый и интервальный типы. Данные этих типов в памяти занимают по 1 байту.

Перечисляемый тип данных задаётся перечислением всех значений в круглых скобках через запятую. Форматы описания их следующие:

1) **Type** имя типа = (значение 1,... , значение n);
   Var идентификаторы: имя типа;

или

2) **Var** идентификаторы: (значение 1, ... , значение n**);**

Пример:

```
Type Gaz = (C,O,N,F);
          Metall = (Fe,Co,Na,Cu,Zn);
     Var Gl,G2:Gaz;
   Metl,Met2:Metall;
Ses : (Winter, Spring, Summer, Autumn);
```

В этом примере явно описаны 2 типа данных пользователя - Gaz и Metall (перечислены некоторые газы и металлы периодической таблицы Д.И. Менделеева). Переменные Gl, G2 и Metl, Met2 могут принимать только одно из перечисленных значений. Третий тип перечисления (Ses) анонимный, так как не имеет имени типа. Он задаётся перечислением значений переменных в разделе описаний **Var** (т.е. записан по второму формату).

Интервальный тип данных задаётся двумя граничными константами диапазона значений для данной переменной. Обе константы должны быть одного из стандартных типов, кроме вещественного, при этом значение первой константы должно быть меньше значения второй. Формат описания:

Type имя типа = константа 1 .. константа 2;
Var идентификаторы: имя типа;

Пример:

```
Type Dni=l ..31;
Var RabDni, VolnDni: Dni;
```

В этом примере переменные RabDni и VolnDni имеют интервальный тип Dni и могут принимать любые значения из диапазона 1 ..31. Выход из диапазона вызывает программное прерывание.

Границы диапазона можно задавать не значениями констант, а их именами:

Const Min = I; Max = 31;

Type Dni = Min .. Max;

Var RabDni, VolnDni: Dni;

Структурированные типы данных (строки, массивы, записи, множества, файлы и указатели) представляют собой упорядоченные определённым образом совокупности скалярных переменных и характеризуются типом своих элементов. Приведём лишь краткую характеристику этих данных (данные типа массивов, которые часто используются в вычислительных алгоритмах инженерных задач, более подробно рассмотрены в модуле № 6).

Строка *– это последовательность символов, заключённая в апострофы.*

Массив *– это данные, состоящие из фиксированного количества элементов, имеющих один и тот же тип и объединённых под общим именем.*

Множество *– это набор выбранных по какому-то признаку (или группе признаков) объектов, которые можно рассматривать как единое целое.*

Запись *– это данные, состоящие из фиксированного числа элементов разного типа.*

Файл *– это данные, состоящие из последовательности элементов одного типа и одной длины.* Чаще всего элементами файла являются записи.

Указатель *– это структурированный тип данных, состоящий из неограниченного множества указывающих на однотипные элементы значений.* Используется при работе с динамическими структурами данных.

Стандартные арифметические функции. Они реализуют наиболее часто встречающиеся математические действия и операции. Приведём их описание и примеры использования, обозначив через **x** целочисленные и вещественные типы.

Abs(x) - вычисление абсолютной величины выражения **x**. Тип результата совпадает с типом параметра **x**.

Abs (4 - 6) = 2

ArcTan(x) - вычисление угла, тангенс которого равен **x**; значение угла представляется в радианах в диапазоне от -Pi/2 до Pi/2. Результат имеет вещественный тип.

ArcTan (l) = Pi/4

Sin(x), Cos (x) - вычисление синуса или косинуса выражения **x**; **x** - в радианах; результат имеет вещественный тип.

Cos (60*Pi/180) = 0.5

Exp(x) - вычисление экспоненты **x**, т.е. значение **e** в степени **x** (**e** - основание натурального логарифма, равное 2,718282). Результат имеет вещественный тип.

Exp (1) = 2.718282

Frac(x) - вычисление дробной части выражения **x**; результат имеет вещественный тип.

Frac (0.25*11) = 0.75

Int(x) - вычисление целой части **x**.

Int (123.448) = 123

Int (-345.555) = -345

Sqr(x) - возведение в квадрат значения **x**. Тип результата совпадает с типом аргумента **x**.

Sqr (5) = 25

Sqrt(x) - вычисление квадратного корня из **x**. Тип результата вещественный.

Sqrt (25) = 5.0

Ln(x) - вычисление натурального логарифма по основанию **e**. Результат имеет вещественный тип.

Ln (3) = 1.0986

Для вычисления логарифма с основанием **a** используется соотношение:

$$\log_a(x) = \ln(x)/\ln(a)$$

Для вычисления других тригонометрических функций следует использовать известные соотношения:

Tan(x) = sin(x)/cos(x) **Ctg(x)** = cos(x)/sin(x)

Csc(x) = l/sin(x) **Sc(x)** = l/cos(x)

ArcSin(x) = $\text{ArcTan}(x/(l-x^2))^{1/2}$

ArcCos(x) = Pi/2-ArcSin(x)

ArcCtg(x) = Pi/2-ArcTan(x)

В языке Паскаль нет операции возведения в степень. Поэтому эту операцию заменяют другими с применением стандартных функций **Exp** и **Ln**:

$$\mathbf{X^a = Exp(a*Ln(x)).}$$

Вычисление выражения $\mathbf{(-x)^n}$, если **n** - целочисленная константа, организуют по циклу путём умножения **(-x)** само на себя **n** раз.

Выражения, операнды и операции. ***Выражение*** задаёт порядок выполнения действий над элементами данных и состоит из *операндов* (констант, переменных, обращений к функциям), *знаков операций* и *круглых скобок*. ***Операции*** определяют действия, которые необходимо выполнить над операндами. В простейшем случае выражение может состоять из одного операнда. Круглые скобки ставятся так же, как в обычных арифметических выражениях для управления порядком выполнения операций. Их использование также вполне приемлемо и полезно для более чёткого и понятного визуального определения порядка вычислений.

Операции подразделяются на арифметические, отношения, логические, строковые и др. Выражения соответственно бывают арифметические, отношения, логические, строковые и др. в зависимости от того, какого типа операнды и операции в них используются.

Операции могут быть *унарными* (относятся к одному операнду и записывается перед ним) и *бинарными* (выражают действия над двумя операндами и записываются между ними). Например, в выражении **-A** символ "-" является унарной операцией, а в выражении **A-B** - бинарной.

Под ***арифметическим выражением*** понимают совокупность констант, переменных и функций, объединённых знаками арифметических операций и круглыми скобками. Результатом вычисления арифметического выражения всегда является число. Арифметические операции языка Паскаль представлены в табл. 2.1.

Операции сложения (+), вычитания (-), умножения (*) и деления (/) выполняется так же, как и в обычных арифметических выражениях.

Целочисленное деление (div) отличается от обычной операции деления тем, что результатом является целая часть частного, при этом дробная часть отбрасывается. Результат целочисленного деления всегда равен нулю, если делимое меньше делителя.

Примеры.	**Выражение**	**Результат**
	10 div 3	3
	2 div 3	0

Деление по модулю (mod) восстанавливает остаток, полученный при выполнении целочисленного деления.

Примеры.	**Выражение**	**Результат**
	10 mod 3	1
	2 mod 3	2

В выражениях для *арифметических операций* **and, shr, shl, or, xor** операнды записываются в десятичной форме, а при выполнении операций они предварительно переводятся в двоичную систему счисления. Результаты ***поразрядного*** вычисления затем представляются в десятичной форме.

Примеры.	**Выражение**	**Результат**
	12 and 22	4
	2 shl 7	256
	160 shr 2	40
	12 or 22	30
	12 xor 22	26

Таблица 2.1

Арифметические операции

Знак	Опрерация	Выражение
+	Сложение	A+B
-	Вычитание	A - B
*	Умножение	A*B
/	Деление	A/B
Div	Целочисленное деление	A div B
Mod	Деление с остатком	A mod B
And	Арифметическое И	A and B
Shr	Целочисленный сдвиг вправо	A shr B
Shl	Целочисленный сдвиг влево	A shl B
Or	Арифметическое ИЛИ	A or B
Xor	Исключающее ИЛИ	A xor B
-	Изменение знака	- A

Операция (унарная) изменения знака восстанавливает значение операнда с противоположным знаком.

Примеры.	**Выражение**	**Результат**
	-(-256)	256
	-(+39)	-39

При написании арифметических выражений рекомендуется соблюдать следующие ограничения и правила:

1. Запрещено последовательное написание двух операций, поэтому запись **A/-B** запрещается, **A/(-B)** разрешается.
2. Приоритет операций в порядке убывания следующий:

1) ***, /, div, mod, and, shl; shr;***

2) **+, -, or, hor.**

3. Часть выражения, заключённая в круглые скобки, выполняется в первую очередь.

4. Операции одинакового приоритета в выражении выполняются последовательно слева направо.

Выражение отношения состоит из двух или более арифметических выражений, соединённых операциями отношения. Оно определяет истинность или ложность результата, который имеет логический (булевский) тип и принимает одно из двух значений: **True** (истина) или **False** (ложь). Операции отношения на Паскале записывается так: **=** (равно), **<>** (не равно), **>** (больше), **<** (меньше), **>=** (больше или равно), **<=** (меньше или равно), **in** (принадлежность). Все операции являются бинарными.

При объединении в одном выражении арифметических операций и операций отношения первыми всегда выполняются арифметические операции. *Сравниваемые данные должны быть одинакового типа.*

Логическое выражение образуется из операндов логического типа и логических операций: **not** (логическое отрицание), **and** (логическое умножение, И), **or** (логическое сложение, ИЛИ), **xor** (исключающее ИЛИ). При этом операндами могут быть: логические константы, логические переменные, выражения отношения.

Старшинство логических операций в порядке убывания следующее: 1) **not,** 2) **and,** 3) **or, xor.** Приоритет логических операций выше операций отношения (в других алгоритмических языках наоборот).

При вычислении логического выражения, содержащего различные операции (арифметические, логические и отношения), выполнение каждой операции осуществляется с учётом её приоритета (табл. 2.2).

Таблица 2.2

Порядок выполнения операций

Операция	Приоритет	Вид операции
not	Первый (высший)	Унарная операция
***, /, div, mod, and, shl, shr**	Второй	Операция типа умножения
+, -, or, xor	Третий	Операция типа сложения
=, < >, <, >, <=, >=, in	Четвертый (низший)	Операция отношения

Для выполнения операций не по старшинству применяются круглые скобки. Например, в выражении **(A<C) and (B=D)** операции отношения будут выполняться раньше, чем логическая операция **and**.

Результатом вычисления логического выражения является константа логического типа.

Вычисления различных функций (переменных) по формулам программируются с помощью *операторов присваивания*. Общий вид оператора присваивания следующий:

Идентификатор переменной : = выражение;

Здесь *идентификатор переменной* – имя переменной, текущее значение которой заменяется новым значением, определяемым данным *выражением*.

Пример.

```
Y : = Sqrt (x)+1;
b : = M and N;
```

В операторе присваивания *идентификатор переменной* и *выражение* должны иметь один и тот же тип, кроме *одного исключения*: идентификатору переменной типа **real** разрешается присваивать выражение типа **integer**.

Модуль М3 – «Программирование линейных алгоритмов»

3.1. Структура и общие правила написания программы на Паскале

Программа реализует алгоритм решения задачи и представляет собой последовательность действий над определёнными данными с помощью математических операций. При разработке программы следует руководствоваться *основными принципами структурного программирования:*

- не используйте сложных методов там, где можно обойтись простыми;
- без крайней необходимости не используйте оператор перехода **goto**;
- исключайте переходы извне внутрь рассматриваемой разветвляющейся структуры;
- циклические структуры задавайте в явном виде, избегая операторов **goto**;
- большие программы разбивайте на логически завершённые сегменты (процедуры и функции);
- выбирайте имена констант, переменных, процедур, функций по смыслу, с учётом их назначения.

Программа на языке Паскаль состоит из строк. Набор текста программы осуществляется с помощью встроенного редактора текстов системы программирования Турбо-Паскаль. Существуют различные схемы написания программ на Паскале, которые отличаются количеством отступов слева в каждой строке и различным использованием прописных букв. Строка может начинаться с любой колонки, т.е. величина отступа от левой границы для каждой строки устанавливается самим программистом с целью получения наиболее ясного текста программы. Количество операторов в строке произвольно. Один оператор может записываться на нескольких строках. Такое разбиение является условным из соображения удобства и чёткости, так как никаких знаков переноса в Паскале не используется.

Синтаксически программа состоит из необязательного заголовка и программного блока. Заголовок в общем случае состоит из ключевого слова Program и имени программы.

Программный блок может содержать в себе другие блоки. Блок, который не входит ни в какой другой блок, называется *глобальным*. Другие блоки, находящиеся в глобальном блоке, называются *локальными*. ***Глобальный блок*** - это основная программа, локальные блоки - это процедуры и функции. Отдельные элементы программы (типы, переменные, константы и др.) соответственно называются глобальными или локальными и областью действия их являются: блок, в котором они описаны, и все вложенные в него блоки. Блочная структура обеспечивает структуризацию программ. В идеальном случае программа на Паскале состоит из подпрограмм (процедур и функций), которые вызываются для выполнения из раздела операторов основной программы.

Программный блок состоит из двух частей: описательной и исполнительной. Описательная часть в общем случае включает в себя 6 разделов: список имён подключаемых модулей (он определяется ключевым словом **Uses**), описание меток, описание констант, определение типов данных, описание переменных, описание процедур и функций. Исполнительная часть (её ещё называют разделом операторов) начинается ключевым словом **Begin** (начало), далее следуют операторы, записанные согласно алгоритму решаемой задачи и отделенные друг от друга точкой с запятой.

Завершается исполнительная часть программного блока ключевым словом **End** (конец) с точкой. Слова **Begin** и **End** являются аналогом открывающей и закрывающей ско-

бок в обычных арифметических выражениях, поэтому их называют ещё **операторными скобками.**

Структуру программы на Паскале в общем случае можно представить следующем образом (рис. 3.1).

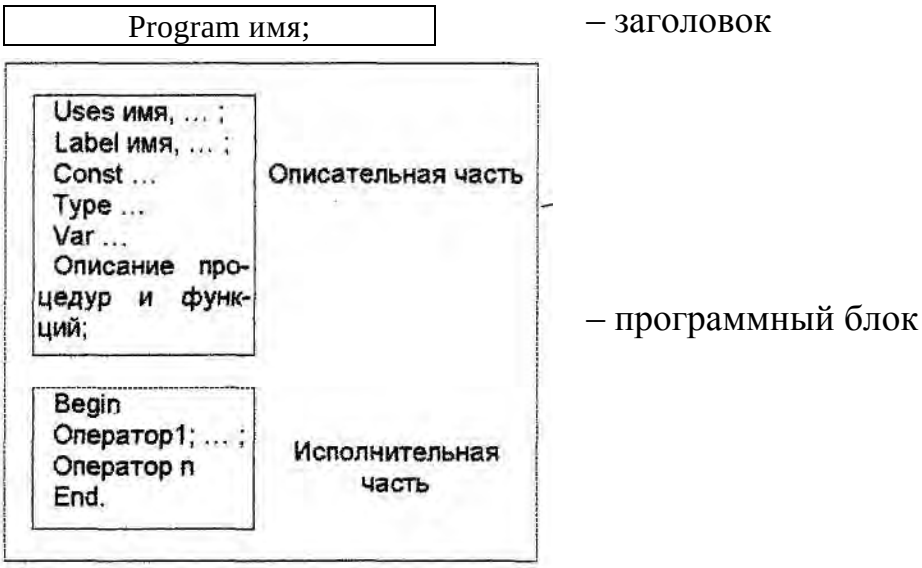


Рис. 3.1

В программе любой описательный раздел может отсутствовать. Разделы описания могут следовать в любом порядке (кроме Uses, который всегда располагается после заголовка программы). Главное, чтобы все описания элементов были бы сделаны до того, как они будут использоваться.

При компиляции программы процессор ПК рассматривает содержащиеся перед операторами описания переменных и отводит в памяти соответствующие места для размещения каждой из переменных. При выполнении программы во время вычисления значения выражения производятся обращения за значениями переменных в отведённые для них места памяти, а полученное новое значение для переменной помещается в закреплённое за данной переменной место в памяти, а предыдущее значение этой переменной стирается.

Раздел Uses. Он состоит из ключевого слова **Uses** и списка имён подключаемых стандартных и пользовательских модулей.

Пример: **Uses** Crt, Dos, MyLib;

Раздел описания меток. Перед любым оператором можно поставить метку, состоящую из имени и следующего за ним двоеточия. Именем метки может служить идентификатор или число. Все метки должны быть описаны. Раздел описания меток начинается ключевым словом **Label** (метка), за которым следуют имена меток, разделённые запятыми. За последним именем ставится точка с запятой.

Пример: **Label** Blok, Ml, 5, 15;

Раздел описания констант. В этом разделе производится присвоение идентификаторам констант постоянных значений. Раздел начинается ключевым словом **Const**, за которым следуют выражения, присваивающие идентификаторам (через =) постоянные числовые или строковые значения. Эти выражение отделяются друг от друга точкой с запятой.

Пример. **Const** A=50; B2='Блок1';

Раздел описания типов данных. Тип данных может быть описан либо в разделе описания переменных, либо определяться идентификатором типа. Раздел описания типов данных начинается ключевым словом **Type,** за которым следует определение типов, разделяемых точкой с запятой.

Пример. **Type** Matr = **array** [1..10] **of** real;
Dni = 1..31; LatBukva = ('a'..'2');

Раздел описания переменных. Каждая встречающаяся в программе переменная должна быть описана. Раздел описания переменных начинается ключевым словом **Var,** затем через запятые перечисляются имена переменных и через двоеточие указывают их тип, а после типа ставится точка с запятой.

Пример. **Var** A,B,C:integer; Res,Sum:real;
Ll, Vhod:boolean;

Раздел описания процедур и функций. В этом разделе размещаются тела подпрограмм. ***Подпрограммой*** *называется программная единица, имеющая имя, по которому она может быть вызвана из других частей программы.* В Паскале роль подпрограмм выполняют процедуры и функции, которые подразделяются на стандартные и определённые пользователем. Стандартные процедуры и функции являются частью языка Паскаль и могут вызываться без предварительного описания. А процедуры и функции пользователя должны описываться обязательно. В общем случае подпрограмма имеет ту же структуру, что и основная программа. При описании подпрограмм их заголовки начинаются ключевыми словами: **Procedure** или **Function.** Более подробное рассмотрение описания процедур и функций пользователя приведено позже (в модуле № 8).

Комментарии. ***Комментарий*** *— это пояснительный тест, который можно записать в любом месте программы, где размещён пробел.* Текст комментария ограничивается символами { } или (* *) и может использовать любые комбинации латинских и русских букв, цифр и других символов алфавита языка Паскаль. Комментарии игнорируются компилятором.

В процессе отладки программы часто требуется временно исключить выполнение какой-либо части программы. Это удобно выполнить путём заключения временно этой части программы в символы { } или (* *), которые после отладки программы можно убрать, и программа будет выполняться в полном объёме.

3.2. Программирование блоков линейных алгоритмов на Паскале

Линейный алгоритм использует такие символы-блоки (рис. 3.2): *терминатор* (начало, конец), *данные* (ввод/вывод данных), *процесс* (блоки 4 - 6). Рассмотрим программирование данных символов-блоков на языке Паскаль.

Блок 1 (начало). Этот блок включает заголовок программы на Паскале, все описательные разделы и операторную скобку **Begin** раздела операторов.

Блоки ввода/вывода данных. Для ввода и вывода данных в Турбо-Паскале существуют стандартные процедуры ввода/вывода, вызываемые, соответственно, операторами **Read** и **Write.**

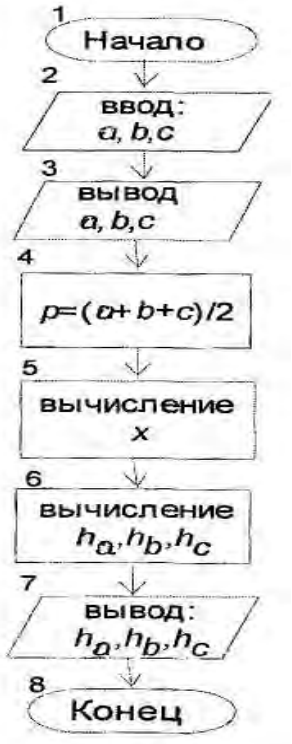


Рис. 3.2

Для вызова процедуры ввода используются три оператора:

1) **Read** (список переменных)**;** - каждое вводимое значение последовательно присваивается переменным из списка;

2) **ReadLn** (список переменных)**;** - то же, что и оператор 1), но после ввода всех данных происходит переход на новую строку, т.е. следующий оператор ввода будет вводить данные с новой строки;

3) **ReadLn ;** - происходит переход (после нажатия оператором клавиши **<Enter>** во время выполнения программы на ПК) на новую строку без ввода данных (без набора данных на клавиатуре).

Последовательно расположенные два оператора 1) и 3) эквивалентны одному оператору 2).

Блоки ввода/вывода данных программируются с помощью операторов ввода/вывода данных: **Read, ReadLn, Write, WriteLn.**

Операторы **Read** и **ReadLn** обеспечивают ввод данных с клавиатуры для последующей их обработки программой. Их форматы:

Read (xl,x2, ...,xn)**;** **ReadLn(**xl,x2,... ,xn)**;** ,

где xl, x2, ..., xn - вводимые переменные. Значения переменных xl,x2, ...,xn при вводе набираются на клавиатуре минимум через один пробел (но не через запятую) и высвечиваются на экране. После набора данных для одного оператора **Read** или **ReadLn** нажимается клавиша **Enter .** Значения вводимых переменных должны соответствовать этим переменным по очерёдности и типам. Если соответствие по типам будет нарушено, то возникнет ошибка ввода и появится сообщение об этом. Если соответствие будет нарушено по очерёдности, то будут неверными результаты вычислений по программе.

Если в программе имеется несколько операторов **Read,** данные для них можно набирать в одной строке потоком, так как после считывания значений переменных по первому оператору **Read** курсор остаётся в той же строке вводимых данных.

Оператор ввода **ReadLn** аналогичен оператору **Read,** за исключением того, что после считывания значения последней переменной одного оператора **ReadLn** курсор перейдёт на начало следующей строки и данные для очередного оператора **ReadLn** должны набираться с начала новой строки.

Операторы **Read** и **ReadLn** требуют *обязательного* ввода данных. Если вы их не введёте, а просто нажмёте клавишу **Enter ,** то работа оператора ввода не закончится и он будет ожидать ввода конкретной информации, а вы не сможете перейти к выполнению следующего оператора программы.

Значения переменных xl,x2,...,xn в операторах **Read** и **ReadLn** при вводе можно набирать и в нескольких строках, нажимая клавишу **Enter** после набора каждой строки данных, так как в данном случае после каждого нажатия клавиши **Enter** курсор переходит в начало следующей строки.

Оператор **ReadLn;,** записанный в программе без списка водимых переменных, приводит к остановке её дальнейшего выполнения до тех пор, пока вы не нажмёте клавишу **Enter .** Поэтому можно использовать оператор **ReadLn;** для просмотра выводимых результатов вычисления по программе, например:

```
… … …
WriteLn (' x = ', x , ' y = ' , y);
Write (‘ Нажмите Enter ’);
ReadLn;
End.
```

В Турбо-Паскале допускается вводить значения следующих данных: целых, вещественных, символьных и строковых переменных.

С помощью оператора ввода нельзя ввести:

1) значение логической переменной;

2) значение переменной перечисляемого типа;

3) значения структурированных переменных (массивов, множеств, записей).

Пример 1.

```
Var A, B, C: real;
    I, K: integer;
… … …
ReadLn (A, B, C);
Read (I, K);
```

В данном примере значения переменных вводятся в следующем порядке:

```
0.5 _ 6.25 _ -7.1E-1
1_5
```

или

```
0.5
6.25
-7.1E-1
1
5
```

Но нельзя все числа записать в одной строке, так как используется оператор **ReadLn.** После выполнения операции ввода переменным будут присвоены такие значения: A = 0,5; B = 6,25; C = -0,71; I=1; K = 5.

Пример 2. Пусть имеются переменные следующих типов: R: real; Cl, C2, C3 : char, которым необходимо присвоить соответственно значения: 1,5; 'A' ; 'B' ; 'C. Для этого используется оператор **Read (**R**,** Cl, C2, C3); . При вводе значения переменных можно расположить следующим образом:

1.5ABC или **1.5EOABC** (без апострофов), но нельзя после 1.5 поместить пробел, так как он воспримется как значение символьной константы.

Оператор вывода данных имеет формы записи:

1) **Write** (список переменных)**;** – выводит последовательно значения переменных из списка;

2) **WriteLn** (список переменных)**;** – то же, что и оператор Write, но после вывода значения последней переменной из списка осуществляется переход на новую строку;

3) **WriteLn ;** - осуществляет переход на новую строку (без вывода данных).

Транслятор по умолчанию отводит определенное число позиций для выводимых величин каждого типа. Все элементы вывода печатаются в строку в заданном порядке, при этом ***пробелы*** *между ними автоматически не ставятся. Их при желании необходимо учитывать самим при программировании операторов вывода.*

Пример. Пусть в результате выполнения программы переменные получили такие значения: I = -5, R= 3.52, C = '+' , B = True.

Выведем их на печать:

```
Program Pr;
Var I: integer;
    R:real;
    C:char;
    B:Boolean;
Begin  …
Write (‘ Пример ‘); WriteLn;
WriteLn (’ I= ’, I,’_ _R=’,  R );
WriteLn (‘ C= ’, C );
WriteLn (‘ B= ’, B );  End.
```

В результате выводимые значения примут вид:

```
I = -5 _ _R= _ 3.5200000000E+00
C= +
B = True.
```

Здесь знак «_» означает *пробел*.

Форматный вывод данных. В ТП предусмотрен вывод данных с форматами. В общем случае формат имеет вид: **P: M**,

где **P** – имя переменной;

M – целая константа, указывающая на число позиций для выводимой величины **P**.

Для вещественных переменных формат может быть задан и в таком 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.

Если параметры M и N опущены, то вещественная переменная выводиться в виде константы с плавающей точкой.

Пример. Используем форматный вывод переменных из предыдущего примера:

```
WriteLn (‘ I= ’, I:3, ’_ _ R=’, R:5:2);
Write (‘ C=’, C:2, ’_ _B = ’, B:6);
```

В результате получим:

```
I = _ _ 5_ _R = _ 3.52
C =_ +_  _B = _ _  True.
```

Необходимо помнить, что все символы (включая пробелы, запятые), заключенные между открывающим и закрывающим апострофом в списке переменных операторов **Write** и **WriteLn,** выводятся на экран как элементы текста. *Если при выводе значения некоторой переменной выводимое число не будет помещаться в указанный формат, то часть значения переменной, расположенная перед десятичной точкой, будет выведена на экран полностью.* При этом число позиций, предназначенных для вывода дробной части числа остается равным указанной в формате величин (дробная часть, не укладывающаяся в заданное число позиций, округляется; округление не изменяет самого значения переменной, а касается только процесса вывода этого значения).

Если в выводимом числе дробная часть отсутствует, оно выводится в экспоненциальной форме с достаточным для точного изображения числом позиций. Но если же вы хотите выдать значение вещественного числа без дробной части (и без экспоненты), то необходимо указать следующий формат:

WriteLn (P :M :O);

Использование форматного вывода позволяет корректно оформлять различного рода таблицы.

Ввод данных с клавиатуры можно осуществлять и по запросам. В этом случае необходимо запрограммировать соответствующие запросы на ввод данных, используя операторы **Write**, а ввод численных значений - по операторам **Read** или **ReadLn**. Например, запрограммируем ввод переменных x = 37,5; y = $-0{,}7\cdot10^2$; z = -2,73 по следующим запросам:

Введите значение x = 37.5

1-й запрос, ответ пользователя на запрос (набор на клавиатуре);

Введите значение y = -0.7E+02

2-й запрос, ответ пользователя на запрос;

Введите значение z = -2.73

3-й запрос, ответ пользователя на запрос.

Тогда на Паскале такой ввод данных будет иметь вид:

Write (' Введите значение x = ');
ReadLn (x);
Write (' Введите значение y = ');
ReadLn (y);
Write (' Введите значение z = ');
ReadLn (z);

Блоки 4 - 6 (рис. 3.2) линейного алгоритма программируются с помощью *операторов присваивания*. Общий вид оператора присваивания следующий:

Идентификатор переменной : = выражение;

Оператор присваивания предписывает выполнить *выражение*, записанное в его правой части, и результат вычисления по этому выражению присвоить (**:=**) *переменной*, идентификатор которой расположен в левой части оператора. Допустимо присваивание любых типов данных, кроме *файловых*.

Пример.

Y : = Sqrt (x)+1;
b : = M and N;

В операторе присваивания идентификатор переменной и выражение должны иметь один и тот же тип, кроме одного исключения: *идентификатору переменной типа* **real** *разрешается присваивать выражение типа* **integer**.

Блок 8 (рис. 3.2) линейного алгоритма программируется оператором **End .** (с точкой).

3.3. Интегрированная среда программирования Турбо-Паскаль

Интегрированная среда программирования Турбо-Паскаль (в дальнейшем ТР) включает в себя: экранный редактор, компилятор, редактор связей и отладчик. ТП позволяет набирать тексты программ с использованием внутреннего редактора текстов, компилировать их, выполнять программы и проводить их отладку. Управление всеми этими функциями возможно и в режиме *меню*, и с помощью соответствующих *функциональных клавиш*. Так, для выбора необходимой функции нужно подвести курсор к требуемой команде и нажать клавишу *ввода* (или *нажать выделенную в команде заглавную букву*).

При возникновении ошибки трансляции ТП автоматически переходит в режим экранного редактирования и ставит курсор в точку возникновения ошибки. Аналогичные действия выполняются и отладчиком при возникновении ошибки во время выполнения программы.

Запуск системы программирования Турбо-Паскаль осуществляется командой **Turbo**, после выполнения которой на экране появляется *главное меню* системы.

Для выхода из ТР можно нажать клавиши «**Alt + X**».

ТП использует следующие основные расширения файлов:

com и e*xe* – выполнимые файлы (программы, готовые для выполнения);

pas – файл с исходным текстом программы на Паскале;

bak – резервная копия *pas*- файла;

tp1 – файл, содержащий стандартные модули ТР.

Для работы с ТР обязательными являются два файла: *Turbo.exe* (компилятор с интегрированной средой программирования) и *Turbo.tp1* (библиотека стандартных модулей).

Главное меню (*первая строка экрана*) содержит команды: ***File*** (файл), ***Edit*** (редактор), ***Run*** (выполнение), ***Compile*** (компилирование), ***Options*** (опции), ***Debug*** (отладка), ***Break/Watch*** (прерывание/просмотр). Все они, кроме *Edit* , имеют собственные подменю, а некоторые – и несколько вложенных подменю.

Для входа в главное меню можно нажать клавишу **F10**, для выхода из него – **Esc** .

Команда ***File*** содержит функции, управляющие работой с файлами: *Load* – загрузка файла с диска и переход в режим экранного редактирования; *New* – удаление текущей программы из памяти и очистка экрана; *Save* - сохранение на диске текущего редактируемого файла и продолжение редактирования и др.

Команда ***Edit*** активизирует экранный редактор (эта команда не имеет собственного меню).

Команда ***Run*** объединяет функции и команды, управляющие трассировкой и выполнением программы. В этот режим входят следующие функции:

Run – запуск программы на выполнение (при необходимости выполняется трансляция программы). По завершении работы программы происходит возврат в ТР. Синоним этой функции – *Ctrl+F9*;

Go to cursor – выполнение программы (без трассировки) от текущей строки (в режиме отладки текущая строка выделяется голубым цветом) до строки, в которой находится курсор. Синоним – *F4*;

Trace into – покомандное выполнение (трассировка) программы. Синоним *F7*;

Step over – пооператорное выполнение программы. Синоним *F8*;

User screen – показ результатов выполнения программы, выведенных на экран. Для возврата достаточно нажать любую клавишу. Синоним – *Alt +F5*.

Команда ***Compile*** содержит команды для управления процессом трансляции программы, например *Compile* – трансляция программы (синоним *Alt +F9*) и др.

Команда ***Options*** обеспечивает управление режимами ТР.

Команда ***Debug*** позволяет определять и изменять значения переменных и используется при отладке программы.

Команда ***Break/Watch*** позволяет управлять точками прерывания и переменными в окне просмотра (в этом окне отражаются текущие значения заданных переменных и выражений; для переключения в окно просмотра из окна редактирования служит клавиша F6).

Модуль М4 – «Программирование разветвляющихся алгоритмов»

В разветвляющихся алгоритмах кроме блоков, применяемых в линейных алгоритмах (программирование которых мы рассмотрели в модуле М3 для рис. 3.2), применяются управляющие блоки «*решение*», определяющие нужную ветвь дальнейших вычислений в зависимости от выполнения или невыполнения конкретных условий. Так как при этом нарушается естественный порядок выполнения блоков алгоритма, то при программировании разветвляющихся алгоритмов могут использоваться условные (**If** или **Case**) и безусловные (**Goto**) операторы управления. Рассмотрим сначала эти операторы.

Оператор безусловного перехода **Goto** означает «перейти к» и применяется в случаях, когда после выполнения некоторого оператора необходимо выполнять дальше не следующий по порядку оператор, а какой-либо другой, помеченный меткой.

Формат написания оператора безусловного перехода следующий:

Goto Метка;

В соответствии с принципами структурного программирования этот оператор следует применять как можно реже, так как его частое употребление усложняет понимание логики программы.

Условный оператор **If** (*если*). Он может использоваться в двух формах:

1) **If** B **then** P1 **else** P2 ;

Здесь ключевые слова **If, then , else** означают соответственно: *если, то, иначе;* **B** – любое логическое (булевское) выражение; **P1, P2** – операторы, которые могут быть простыми, составными или условными.

Управление по этому оператору осуществляется следующим образом (рис. 4.1): если выражение **B** *истинно,* то выполняется оператор **P1**, а оператор **P2** пропускается; если **B** *ложно,* то пропускается оператор **P1**, а оператор **P2** выполняется.

2) **If** B **then** P1;

Управление по этому оператору осуществляется следующим образом (рис. 4.2): если выражение **B** *истинно,* то выполняется оператор **P1** и далее оператор, записанный после оператора **If**; если **B** *ложно,* то оператор **P1** пропускается (не выполняется).

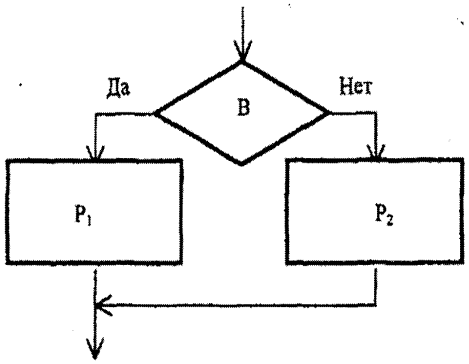


Рис 4.1

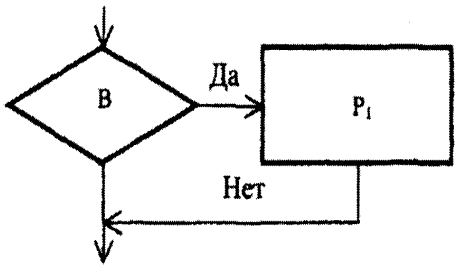


Рис. 4.2

Один оператор **If** может входить в состав другого оператора **If**. В таком случае говорят о вложенности операторов **If**:

If Bl **then If** B2 **then** P1 **else** P2 ;

При вложенности операторов **If** каждое **else** соответствует тому **then** , которое непосредственно ему предшествует (рис. 4.3). Конструкций со степенью вложения более 2 – 3 стараются избегать (из-за сложности их анализа при отладке программы).

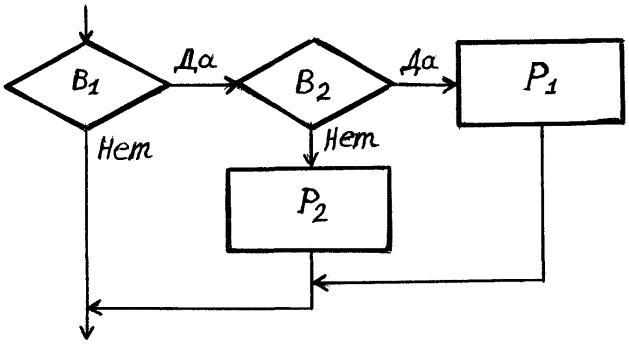


Рис. 4.3

Простыми операторами *называются такие операторы, которые не содержат в себе никаких других операторов.* К ним относятся операторы присваивания, безусловного перехода, вызова процедуры (этот оператор рассмотрен позже - в модуле М7) и пустой оператор.

Пустой оператор *не содержит в себе никаких символов и не выполняет никаких действий.* Он может быть расположен в любом месте программы, где синтаксис языка допускает наличие оператора. Как и все другие операторы, пустой оператор может быть помечен меткой. Чаще всего пустой оператор используется для организации выхода из середины программы или составного оператора:

```
Begin  … ..
Goto Ml; {Переход в конец программы}
… .
M1:     {Пустой оператор с меткой Ml}
End.
```

Составной оператор – это группа из произвольного числа операторов, отделенных друг от друга точкой с запятой и ограниченных операторными скобками **begin** и **end** . Составной оператор воспринимается как единое целое и может находиться в любом месте программы (чаще всего он используется в условных операторах и операторах повтора).

Оператор выбора Case. Этот оператор является обобщением оператора **If** и позволяет сделать выбор из произвольного числа вариантов. Он состоит из *выражения – селектора* и *последовательности операторов*, каждому из которых предшествует *список констант выбора* (список может состоять и из одной константы). Как и оператор If, оператор **Case** может использоваться в двух формах: со словом **else**, имеющим тот же смысл, как и в операторе **If**, и без него. Их форматы записи следующие:

1.
```
Case K of
S1:P1;
S2: P2;
…
Sn:  Pn
else  Pn+1
end;
```

2. Case К **of**
SI:PI :
S2: P2;
…
Sn: Pn
end;

Здесь **К** – *выражение-селектор*, которое может быть одним из скалярных типов (кроме **real**), т.е. целого, логического, символьного или пользовательских типов.

Оператор **Case** работает следующим образом (рис. 4.4): сначала выполняется тот из операторов **Pi**, константа выбора которого равна текущему значению *выражения-селектора* **К**. Если ни одна из констант **Si** не равна текущему значению **К**, то выполняется оператор, стоящий за словом **else.** Во втором случае (рис. 4.5), когда слово **else** в операторе **Case** отсутствует, если ни одна из констант **Si** не равна текущему значению **К**, то выполняется первый оператор, за границей **end;** (оператор **Case** в этом случае пропускается).

Список констант выбора состоит из произвольного количества значений или диапазонов. Тип констант должен совпадать с типом выражения-селектора **К**.

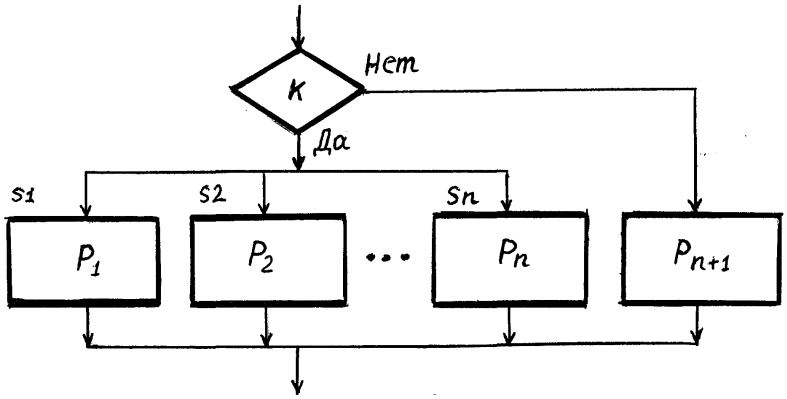


Рис. 4.4

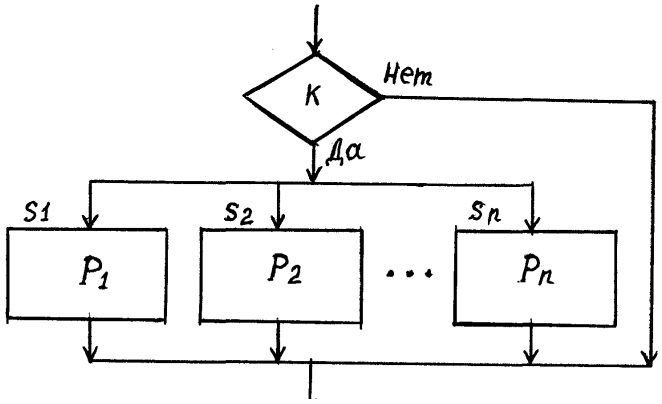


Рис. 4.5

Пример. Составим программу разветвляющегося алгоритма (рис. 4.6) по вычислению функции

$$D = Ln(ab)^2, \quad если \ ab<0 \ ;$$
$$D = Ln(ab), \quad если \ ab>0 \ ;$$
$$D = (a+b), \quad если \ ab=0 \ .$$

Пусть исходные переменные ***a*** и ***b*** вещественного типа. Запрограммируем, например, разветвляющуюся структуру с блоком «решение» 3 с помощью оператора **If**, а - с блоком «решение» 5 с помощью оператора **Case** . Тогда программа будет иметь вид:

```
Program RazAlg;
 Var a, b, d :real;
  Begin
   ReadLn (a, b);
   If a*b<0 then
    d : = Ln (sqr (a*b))
    else Case a*b>0 of
         True: d: =Ln(a*b)
         else d: = a+b
         end;
   WriteLn ( ' d = ', d );
    End.
```

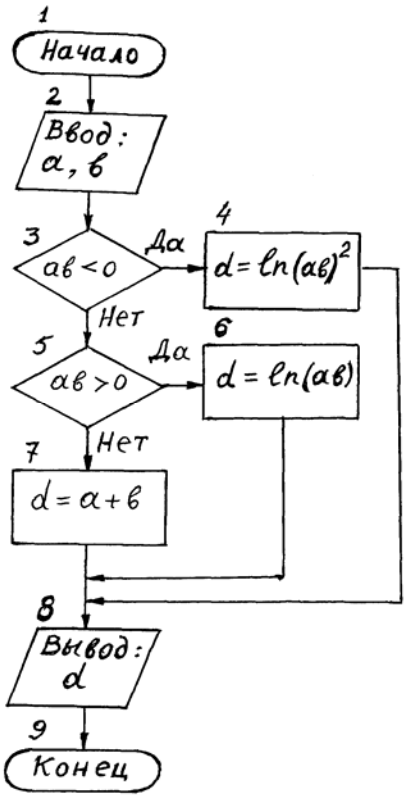


Рис. 4.6

Модуль М5 – «Программирование циклических алгоритмов»

При программировании циклических алгоритмов используются операторы повтора: **For** (для), **Repeat** (повторять) и **While** (пока).

Оператор повтора **For** состоит из заголовка и тела цикла. Тело цикла представляет собой один или несколько операторов (согласно алгоритму), которые могут выполняться более одного раза. Оператор **For** используется обычно тогда, когда количество повторов известно заранее. Он может представляться в двух формах:

1. **For** I:= nl **to** n2 **do** P**;**
2. **For** I:= nl **downto** n2 **do** P**;** ,

где nl, n2 – *выражения*, определяющие начальное и конечное значения *параметра цикла* **I**;

For ... **do** – заголовок цикла;

P – тело цикла, которое может быть простым или составным оператором.

Оператор **For** обеспечивает выполнение цикла (рис. 5.1) до тех пор, пока не будет перебраны все значения параметра цикла **I** от начального до конечного значения включительно. Параметр цикла, его начальное и конечное значения должны принадлежать к одному и тому же типу, при этом допустим любой скалярный тип, кроме вещественного. Если используется тип **integer, byte** и интервальный, то значения параметра цикла последовательно увеличиваются (при **For** ... **to)** или уменьшаются (при **For** ... **downto)** на единицу при каждом повторе.

В теле цикла нельзя использовать операторы, изменяющие значение параметра цикла. После нормального (если не используется преждевременный выход из цикла с помощью оператора **goto**) завершения оператора значение параметра цикла равно его конечному значению.

В теле оператора **For** могут находиться другие операторы **For** (при вложенных циклах, рис. 5.2).

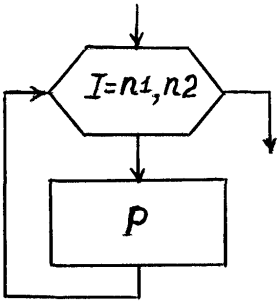


Рис. 5.1

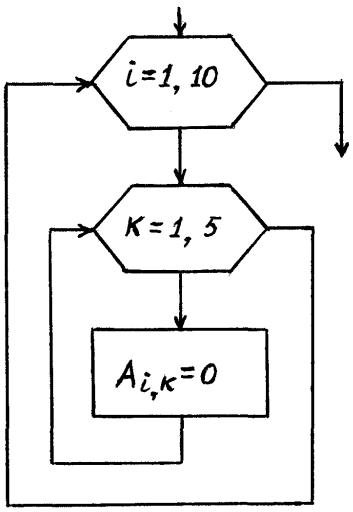


Рис. 5.2

Оператор повтора **Repeat** записывается в виде:

Repeat P1; …; Pn **until** B ; ,

где операторы P1; ... ; Pn - тело цикла;

B - выражение булевского типа.

Этот оператор означает: *повторять* (**Repeat**) выполнять тело цикла *пока не* (**until**) станет выражение B истинным, после чего осуществляется выход из цикла (рис. 5.3).

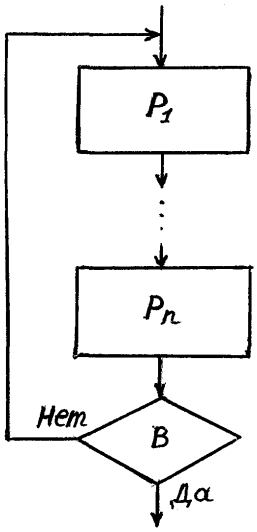


Рис. 5.3

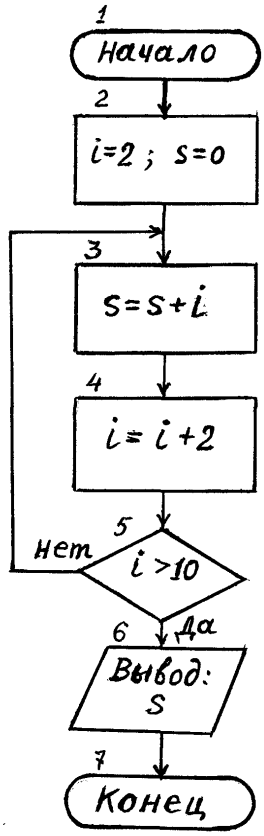


Рис. 5.4

Оператор **Repeat** имеет такие характерные особенности: тело цикла выполнятся по крайней мере один раз; оно выполняется до тех пор, пока условие **B** равно **False**; в теле цикла может находиться несколько операторов, при этом операторные скобки **begin** и **end** не используются; по крайней мере один оператор из операторов тела цикла должен влиять на значение условия **B**, а иначе цикл может повторяться бесконечно.

Пример. Программа вычисления суммы четных чисел в интервале (0 - 10) включительно согласно алгоритму рис. 5.4, составленная с использованием оператора **Repeat**, имеет вид:

```
Program CklAlg ;
Var I, S : integer ;
Begin
   I : = 0 ;  S : = 0 ;
    Repeat  S : = S + I ;
            I : = I + 2
            Until ( I > 10 ) ;
     Writeln ( ‘ S = ‘ , S )
End .
```

Оператор повтора **While** похож на оператор **Repeat** , но проверка логического условия **B** выполнения тела цикла осуществляется в самом начале оператора. Он записывается в виде:

While B **do** P ;

и означает: ***пока*** (While) выражение **B** истинно, то необходимо ***выполнять*** (do) тело цикла **P** (рис. 5.5).

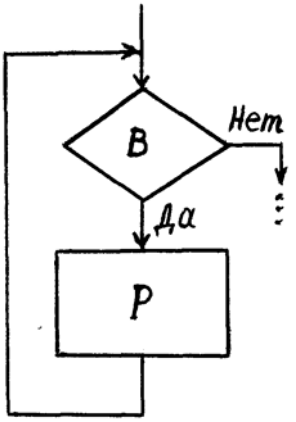


Рис. 5.5

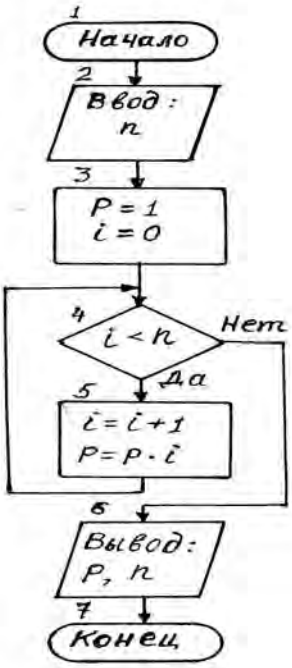


Рис. 5.6

Перед каждым выполнением тела цикла вычисляется значение выражения **B**. Если результат равен **True** , происходит выполнение тела цикла **P**, а если значение выражения **B** равно **False** , то тело цикла **P** не выполняется и осуществляется переход к следующему после цикла оператору. Но если перед первым выполнением цикла значение выражения **B** равно **False** , то тело цикла **P** не выполняется ни разу.

Как и в операторе ***Repeat***, в теле цикла должен присутствовать оператор, изменяющий переменную, входящую в условие **B**, иначе цикл может оказаться бесконечным.

Оператор *While*, как и операторы *For* и *Repeat*, может быть вложенным.

Пример. Программа, в которой с помощью оператора *While* вычисляется **n!** согласно алгоритму рис. 5.6, имеет вид:

```
Program Fakt ;
Var n , I , p : integer ;
 Begin
    Readln (n) ;
     p : = 1 ; I : = 0 ;
     While I < n do
         Begin
           I : = I + 1 ;
           p : = p * I
         End ;
     Write ( ‘ n! = ‘ , p , ' _ _ n = ' , n )
   End .
```

Модуль М6 – «Программирование алгоритмов с использованием массивов данных»

6.1. Массивы данных и их описание

Под **массивом данных** понимается упорядоченная совокупность конечного числа данных одного типа под одним именем. Имена массивов образуются так же, как и имена простых переменных. Элементами массива данных могут быть данные любого типа, включая структурированные типы. Число элементов массива фиксируется при описании и в процессе выполнения программы не меняется. Доступ к каждому отдельному элементу массива осуществляется путем указания имени массива и конкретных индексов этого элемента в *квадратных* скобках. Индексы могут представлять собой выражения любого скалярного типа, кроме вещественного. Тип индекса определяет границы изменения его значений. Если задан один индекс, массив называется одномерным, если задано два индекса - двумерным, если n индексов - n-мерным.

Возможны два способа описания массивов:

1) **Type** *имя типа* = **array** [t1, t2,…, tN] **of** *тип данных* ;
var *имя массива* : *имя типа*;

2) **var** *имя массива* : **array** [t1, t2,…, tN] **of** *тип данных* ;

Здесь t1, t2,…, tN – типы индексов массива (количество индексов N определяет размерность массива).

Пример. Пусть в программе необходимо описать следующий двумерный массив M:

[1 2 5]
[6 7 2]

Описание этого массива в соответствии с первым способом выглядит так:

```
Type  Mas =  array [1.. 2, 1.. 3] of  integer ;
var  M : Mas;
```

Для второго способа имеем:

```
var  M : array [1.. 2, 1.. 3] of  integer ;
```

В самой программе, задав конкретные значения индексов, можно выбрать определенный элемент массива:

```
N : = M [1, 3 ] ;     {т.е. N будет присвоено значение 5} .
```

Для описания массива можно использовать предварительно определенные константы, например:

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

Элементы массива располагаются в памяти последовательно. Элементы одномерного массива 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.. 2, 1.. 3] of  integer ; ,
```

то в памяти элементы массива будут размещены по возрастанию адресов в такой последовательности:

A[1, 1], A[1, 2], A[1, 3], A[2, 1], A[2, 2], A[2, 3].

6.2. Действия над массивами

Для работы с массивом как *единым целым* используется имя массива без указания индексов. Массив может участвовать только в операциях отношения «равно», «не равно» и в операторе присваивания. При этом участвующие в этих действиях массивы должны быть одинаковыми по структуре, т.е. иметь одинаковые типы индексов и количество элементов. Например,

Var A, B : **array** [1..20] **of** real;

Выражение отношения **A = B** дает результат *True* , если значения каждого элемента массива **A** равны значениям соответствующих элементов массива **B**. Выражение отношения **A<> B** дает результат *True* , если хотя бы один элемент массива **A** по значению не равен соответствующему элементу массива **B**.

По оператору **A : = B** все значения элементов массива **B** присваиваются соответствующим элементам массива **A,** при этом значения элементов массива **B** остаются неизменными.

6.3. Действия над элементами массивов

Индексированные элементы массива называются *индексированными переменными* и могут быть использованы точно так же, как и обычные (простые) переменные. Например, они могут находиться в выражениях в качестве операндов; использоваться в операторах повтора; входить в качестве параметров в операторы ввода и вывода данных; им можно присваивать любые значения, соответствующие их типу.

Рассмотрим *типичные ситуации*, возникающие при работе с массивами данных. Пусть имеем три массива и четыре вспомогательные переменные:

```
Var A, D : array [1..4] of real;
    B : array [1..10, 1..15] of integer;
    I, J, K : integer;  Vs : real;
```

Инициализация массива заключается в присваивании каждому элементу массива одного и того же значения, соответствующего базовому типу массива. Наиболее эффективно эта операция выполняется с помощью оператора цикла **for** (рис. 6.1):

```
For I : = 1 to 4 do
    A[I] : = 0 ;
```

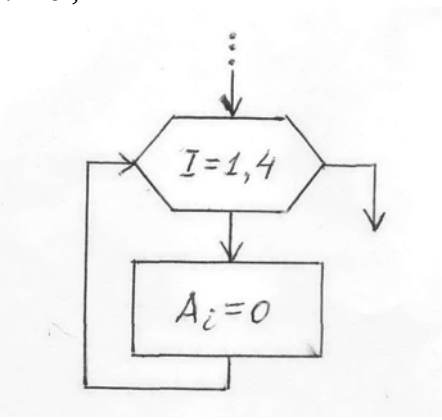


Рис. 6.1

Для инициализации двумерного массива используется вложенный оператор **for** (рис. 6.2):

```
for I : = 1 to 10 do
        For J : = 1 to 15 do
            B[I, J] : = 0 ;
```

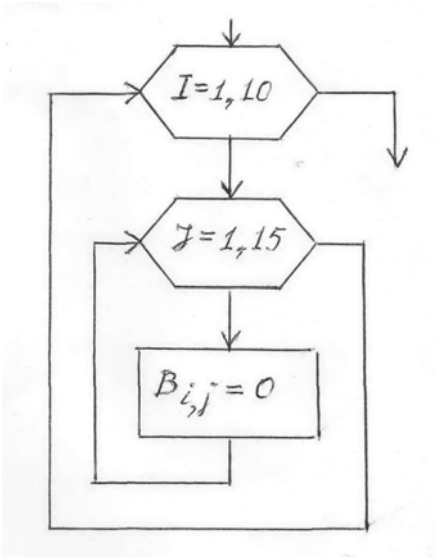


Рис. 6.2

Паскаль не имеет средств ***ввода/вывода элементов массива*** сразу, поэтому вод/вывод их значений производится поэлементно, чаще всего с помощью оператора **Read** или **ReadLn** с использованием оператора организации цикла **for**, например (рис. 6.3, 6. 4):

```
For I : = 1 to 4 do
  ReadLn (A[I]);
```

```
For I : = 1 to 10 do
  for J : = 1 to 15 do
    ReadLn (B[I, J]);
```

В этих примерах использовался оператор **ReadLn**, поэтому каждое вводимое значение элементов массива будет набираться с новой строки.

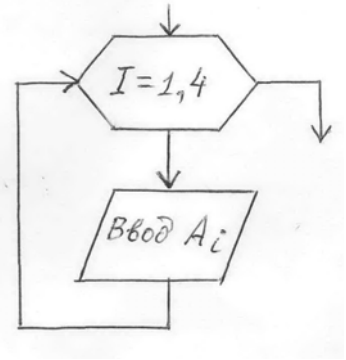


Рис. 6.3

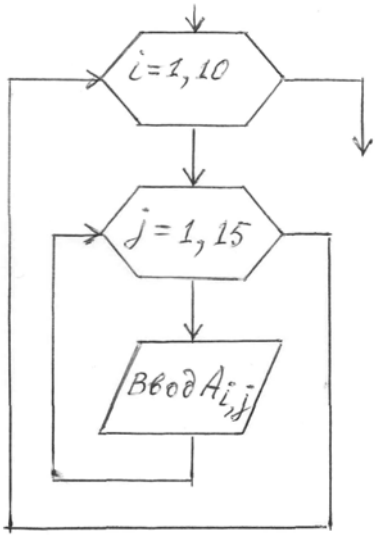


Рис. 6.4

Можно ввести и значения отдельных элементов, а не всего массива, например: **Read** (A[3]), B[6, 9]); - оба значения набираются на одной строке экрана, начиная с текущей позиции расположения курсора.

Вывод значений элементов массива выполняется аналогичным образом, но используются операторы **Write** или **WtiteLn.**

Копированием массивов называется присваивание значений всех элементов одного массива всем соответствующим элементам другого массива. Копирование можно выполнить одним оператором присваивания, например **A : = D;** или с помощью оператора **for** :

```
For I : = 1 to 4 do
    A[I] : = D[I];
```

В обоих случаях значения элементов массива **D** не изменяются. Очевидно, что оба массива должны быть одинаковыми по структуре.

Иногда требуется осуществить **поиск** *в массиве каких-либо элементов, удовлетворяющих некоторым известным (заданным) условиям*. Пусть, например, надо определить, сколько элементов массива **A** имеет нулевое значение. Для ответа на этот вопрос введем дополнительную переменную ***K*** и определим ее значение по алгоритму рис. 6.5:

```
K : = 0;
For I : = 1 to 4 do
  If A[I] = 0 then
    K : = K + 1;
```

После выполнения данного цикла переменная **K** будет иметь значение, равное числу элементов массива **A** с нулевыми значениями.

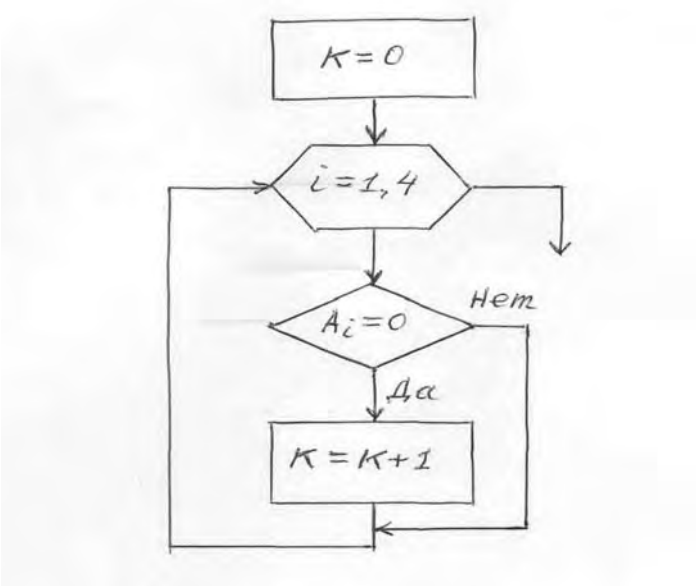


Рис. 6.5

Перестановка значений элементов массива осуществляется с помощью дополнительной переменной того же типа, что и базовый тип массива. Например, требуется поменять значения первого и четвертого элементов массива **A**:

```
Vs : = A[4]; { Vs – вспомогательная переменная }
A[4] : = A[1];
A[1] : = Vs;
```

6.4. Программирование с использованием динамического распределения оперативной памяти под массивы

При выполнении любой программы каждая используемая в ней переменная получает свой адрес в оперативной памяти *автоматически*, так как все переменные программы объявляются в разделе их описания. В Турбо-Паскале (ТП) имеются два способа распределения памяти для переменных: статический и динамический. При *статическом распределении памяти* всем объявленным в программе переменным в сегменте данных ПК выделяются фиксированные участки оперативной памяти. Поэтому использование в программе не объявленных переменных не допускается.

При ***динамическом распределении памяти*** имеется возможность создавать новые, не объявленные заранее в полном объеме переменные и размещать их на свободные участки в динамической области оперативной памяти. Это достигается за счет использования так называемых *указателей данных*.

Указатель данных – это элемент данных, представляющий собой ссылку (адрес) на определенную ячейку оперативной памяти, начиная с которой записывается значение переменной. *Такие переменные, которые размещаются в динамической области оперативной памяти с помощью указателя, называются* ***динамическими переменными.***

Указатель может принимать значения, равные всем адресам оперативной памяти, по которым возможна запись данных. Указатель может также иметь стандартное значение Ni1 (пусто), которое говорит о том, что соответствующая динамическая переменная в оперативной памяти отсутствует.

Указатель объявляется с помощью специального символа *каре* (^), за которым записывается идентификатор типа динамической переменной в следующем виде:

Type имя_типа = ^ тип;
Var имя_переменной : имя_типа ;

Пример.

```
Type  T = ^ integer ;
Var    A, B, C : T ;      или      Var   A, B, C : ^ integer ;
```

В данном случае A, B, C являются указателями на динамические переменные типа integer . Для обращения к значениям этих переменных служат идентификаторы A^ , B^ , C^ .

Указатель может быть объявлен также и *явно* следующим образом:

Var *P: pointer* ;

где *P* – имя указателя, а *pointer* - тип указателя.

ТП допускает описание типизированных констант типа pointer (констант ссылочного типа). Начальным значением таких констант может быть только Ni1. *Например*:

```
Type A = array [0 .. 5]  of char ;
       P = ^A;
Const   P1:  P = Ni1;
```

Значения указателей можно сравнивать только с помощью проверок на равенство или неравенство. Допустимо также для них использование оператора присваивания.

Для динамических переменных допустимы все операции, что и над обычными переменными.

Любым действиям с динамической переменной должна предшествовать процедура ее размещения в оперативной памяти, имеющая вид:

New (var P: pointer);

Она создает новую динамическую переменную, присваивая **P** значение адреса ее размещения в оперативной памяти. При этом динамической переменной отводится блок памяти, соответствующий размеру типа, с которым объявлен указатель **P**.

Если в ходе вычислительного процесса динамическая переменная становится ненужной, ее необходимо удалить с помощью такой процедуры:

Dispose (var P: pointer);

При этом память, занятая динамической переменной, освобождается, делая значение ее указателя **P** неопределенным.

Кроме указанных процедур ТП поддерживает и другие стандартные процедуры и функции для работы с указателями и динамическими переменными. *Например:*

GetMem (var P: pointer; Size: word); - создается новая динамическая переменная размером Size байт, устанавливая значение указателя **P** на начало выделенной ей динамической области оперативной памяти. Значение **Size** не может превышать 65 535 байт ;

FreeMem (var P: pointer; Size: word); - уничтожается динамическая переменная с освобождением **Size** байт памяти, а значение **P** становится неопределенным.

В программах, использующих работу с массивами данных, при объявлении (описании) каждого массива необходимо указывать его размерность. Это требование снижает потребительские свойства таких программ, т.е. эти программы являются зависимыми от конкретной размерности используемых в них массивов данных. От такого недостатка можно избавиться, если массивы использовать как *динамические*. Например, следующая программа по транспонированию квадратной матрицы размером N*M использует динамический массив данных :

```
Program Trans;
Type
        A = array [ 1 .. 2, 1 .. 2 ]  of  real;
Var     N, I, O : integer;
        Mas : ^A;
        R: real;
Begin   { $ R- }
    WriteLn ( ‘ Введите размерность матрицы ‘ );
    Read ( N );
       {Размещение динамического массива }
    GetMem ( Mas, SizeOf(real)*N*N) ;
       { Ввод значений массива }
    WriteLn ( ' Введите исходную матрицу ' ) ;
    For I:= 1 to N do
        for J : = 1 to N do
            Read ( Mas^ [ I, J ] ) ;
       { Транспонирование матрицы }
    For I : = 1 to N do
```

```
            If I< N then
               for  J:= I+1 to N do
                    begin
               R: = Mas^ [ I, J ] ;
               Mas^ [ I, J ] : = Mas^ [ J, I ] ;
               Mas^ [ J, I ] : = R
                    end;
           { Вывод результатов }
        For I : = 1 to N do
            begin
            for J : = 1 to N do
                Write ( Mas^ [ I,  J ] ) ;
                WriteLn;
                FreeMem ( MasSizeOf(real)*N*N) ;
             end;
        { $ R+ }
    End.
```

В этой программе размерность матрицы задается при вводе исходных данных (т.е. значением вводимой переменной **N**). А затем с помощью процедуры **GetMem** в оперативной памяти размещается динамический массив **Mas^** . Так как в программе область изменения индексов этого массива выходит за пределы объявленных значений [1..2] , то следует выключить автоматическую проверку диапазона изменения переменных с помощью директивы компилятору **{ $ R- }**. Далее, после ввода значений матрицы, она транспонируется. Причем при транспонировании достаточно просмотреть лишь наддиагональную (или поддиагональную) часть матрицы.

После вывода транспонированной матрицы динамический массив уничтожается с помощью стандартной процедуры **FreeMem .**

Модуль М7 – «Программирование алгоритмов с использованием подпрограмм»

В 1957 г. была описана М.Уилксом концепция подпрограмм, которая получила широкое распространение практически во всех языках программирования. ***Подпрограммой*** *называется именованная логически законченная группа операторов языка, которую можно вызывать по имени многократно из разных мест основной программы.* В Паскале для организации подпрограмм используются процедуры и функции, которые подразделяются на две группы: стандартные (встроенные) и определенные пользователем. Стандартные процедуры и функции (например, рассмотренные раньше арифметические функции) являются частью языка и могут вызываться по имени без предварительного описания в программном блоке, а процедуры и функции пользователя разрабатываются самим программистом в соответствии с синтаксисом языка и предварительное описание их обязательно. Процедуры и функции пользователя по структуре похожи с основной программой.

Процедура пользователя. Описание процедуры включает заголовок и тело процедуры (рис. 7.1). Заголовок состоит из ключевого слова **Procedure,** имени процедуры и необязательного заключенного в круглые скобки списка формальных параметров с указанием их типа. Тело процедуры представляет собой локальный блок, по структуре аналогичный программе.

Procedure имя (формальные параметры);
Разделы описаний
Begin
Раздел операторов
End;

Рис. 7.1. Структура процедуры пользователя

Для обращения к процедуре из программы используется оператор вызова процедуры, который состоит из имени процедуры и списка фактических параметров (без указания их типа), отделенных друг от друга запятыми и заключенных в круглые скобки. При выполнении процедуры формальные параметры заменяются на фактические. Количество, порядок расположения и тип формальных параметров соответствуют количеству, порядку расположения и типу фактических параметров. Имена формальных и соответствующих фактических параметров могут быть разными или одинаковыми. *Входными* фактическими параметрами могут быть константы, переменные и выражения, а *выходными* – только переменные. Значение формального входного параметра может изменяться в самой процедуре, однако это не сказывается на значении фактического входного параметра. Поэтому входной параметр не может быть результатом работы процедуры. Для этого необходимо использовать выходные параметры (параметры переменные).

Если процедура возвращает в программу какие-то значения, то соответствующие переменные в заголовке процедуры должны быть описаны как параметры-переменные с использованием слова **Var,** например:

Procedure W1 (Var X1, X2 : integer; R1, R2 :integer);

Пример. Пусть требуется составить программу вычисления площади **(Sabcd)** выпуклого четырехугольника (рис. 7.2), заданного длинами четырех сторон **ab, bc, cd, da** и диагонали **ac**.

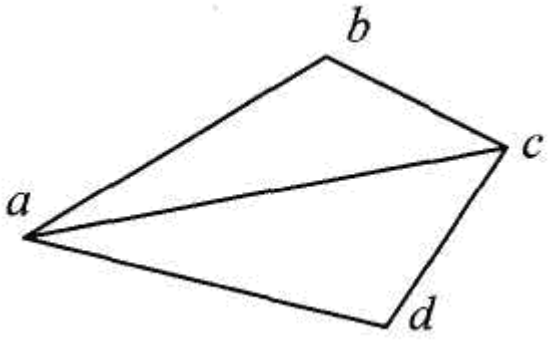


Рис. 7.2

Диагональ делит выпуклый четырехугольник на два треугольника, к которым применима формула Герона по вычислению площади треугольника **S** :

$$S = p(p-a)(p-b)(p-c),$$

где **a, b, c** – длины сторон треугольника;

p – полупериметр треугольника.

Решим поставленную задачу путем вычисления площади **Sabcd** как сумму площадей треугольников **Sabc** и **Sacd** согласно алгоритму рис. 7.3, в котором вычисление площади треугольника по формуле Герона оформим в процедуру с именем **STR** согласно алгоритму рис. 7.4.

Тогда один из вариантов программы на Паскале будет иметь вид:

```
Program Proc ;
  Var AB, BC, CD, DA, AC, Sabc, Sacd, Sabcd : real;
Procedure STR (Var S : real; a, b, c : real);
  Var p : real;
  begin
    P : = ( a + b + c ) / 2 ;
   S: = S q r t ( p * ( p - a ) * ( p - b ) * ( p - c ) )
  end ;

   Begin
ReadLn ( AB, BC, CD, DA, AC);
      STRK ( Sabc, AB, BC, AC ) ;
      STRK ( Sacd, AC, CD, DA ) ;
      Sabcd : = Sabc + Sacd ;
      Write (' Sabcd = ', Sabcd )
  End.
```

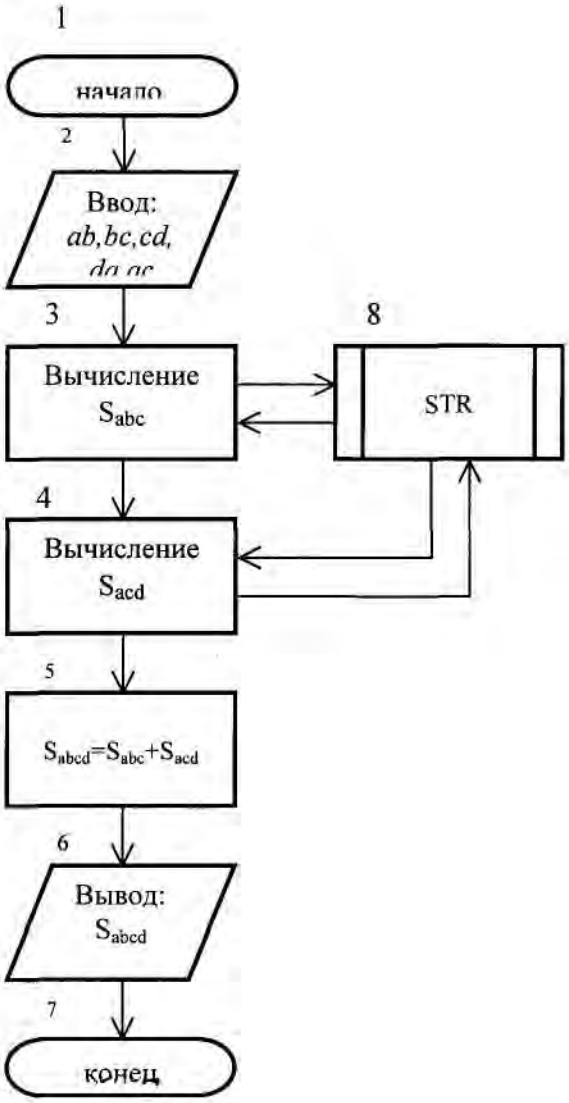


Рис. 7.3

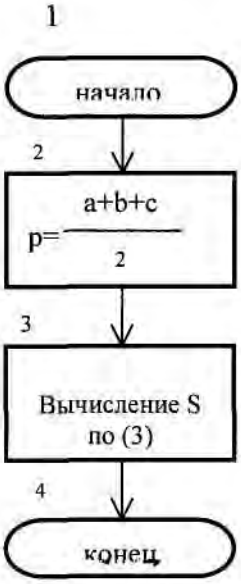


Рис. 7.4

При использовании в качестве параметров процедуры данных сложного типа (массивов и др.) в основной программе необходимо предварительно описать имя типа этих данных, которые потом указываются в списке формальных параметров процедуры. *Например:* пусть необходимо составить процедуру, которая перемножает две квадратные матрицы *A* и *B*, а результат помещает в матрицу *C*. Прежде чем записать заголовок этой процедуры, в основной программе необходимо задать следующее определение:

Type Mas = **array** [1 .. 10, 1 .. 10] **of** real ;

Тогда заголовок процедуры может иметь такой вид:

Procedure Mult (A, B : Mas; var C : Mas);

Через выходные параметры можно не только получать результирующее значение из процедуры, но и передавать начальные значения в процедуру. Так, заголовок процедуры **Mult** можно записать и в таком виде: **Procedure Mult (A, B, C : Mas);**

Функция пользователя. Она состоит (рис. 7.5) из заголовка и тела функции. Заголовок содержит ключевое слово **Function** , имя функции, заключенный в круглые скобки необязательный список формальных параметров с указанием их типа, а после скобок через двоеточие тип возвращаемого функцией значения.

Например:

Function W2 (X, Y, T : integer): real;

Тело функции представляет собой локальный блок (рис. 7.5), по структуре аналогичный программному блоку.

Function имя (формальные параметры): тип результата ;
Разделы описаний
begin
Раздел операторов
end ;

Рис. 7.5. Структура функции пользователя.

В разделе операторов должен находиться, по крайней мере, один оператор, присваивающий имени функции пользователя значение.

Обращение к функции из основной программы *осуществляется по имени* с указанием списка фактических параметров, которые должны соответствовать формальным параметрам и иметь тот же тип. Имя функции, встречающееся в выражениях основной программы, называется ***указателем функции*** или ***обращением к функции***.

Функция имеет следующие отличия от ***процедуры***:

1) она возвращает результат своей работы (значение) в точку вызова;
2) возвращает результат через имя функции;
3) при этом возвращает только один результат;
4) ***имя функции входит в выражение как операнд*** (этим именем она вызывается), а *имя процедуры не может входить в выражение в качестве операнда; процедура вызывается отдельным оператором, может возвращать несколько результатов, при этом возвращает результаты через формальные параметры, а не через имя.*

Пример. Решим задачу предыдущего примера, оформив вычисление площади треугольника по формуле Герона с помощью функции пользователя с именем STR , а не процедуры. Тогда программа на Паскале будет иметь вид:

```
Program Funk ;
 Var AB, BC, CD, DA, AC, Sabc, Sacd, Sabcd : real;
  Function STR ( a, b, c : real): real;
   Var S, p : real;
   begin p : =  (a + b + c ) / 2 ;
         S : = Sqrt ( p * ( p - a ) * ( p - и ) * ( з - с ) ) ж
        STR: = S ;
  end ;
  Begin
 ReadLn (AB, BC, CD, DA, AC);
 Sabc : = STR (AB, BC, AC) ;
 Sacd : = STR (AC, CD, DA);
 Sabcd : = Sabc +  Sacd;
 WriteLn ( ' Sabcd = ', Sabcd )
 End.
```

Модуль М8 – «Методика программирование алгоритмов сложных задач»

В предыдущих модулях рассматривались примеры программирования сравнительно простых задач. Рассмотрим методику написания Паскаль-программы, включая разработку алгоритма, на следующем конкретном примере более *сложной* инженерной задачи. Под понятием «*сложная задача*» будем понимать такую задачу, в которой используются линейные, разветвляющиеся и циклические участки вычислительного процесса; среди данных, участвующих в обработке, имеются массивы данных, а также вычисление некоторой функции требуется организовать по подпрограмме.

Постановка задачи. Пусть требуется разработать алгоритм и по нему написать программу на Паскале для вычисления всех значений функции

$y_k = k \cdot \beta$, если $\beta < b$;

$y_k = \alpha / k$, если $\beta \geq b$,

где $\beta = 2t$; $\alpha = \sin(3\beta)$; $k = 1, 2, 3, ..., m$; b – наименьший по значению элемент одномерного массива данных $x_i = (x_1, x_2, ..., x_n)$; вычисление величины b организовать по подпрограмме.

Один из вариантов исходных данных:

$n = 7$; $x_1 = 1{,}7 \cdot 10^1$; $x_2 = -0{,}23 \cdot 10^2$; $x_3 = 2{,}07$; $x_4 = -1{,}3$;

$x_5 = 23{,}2 \cdot 10^{-1}$; $x_6 = 75{,}3$; $x_7 = -5{,}5 \cdot 10^{-1}$; $t = 0{,}6$; $m = 20$.

Решение.

Так как данная задача математически сформулирована (приведены формулы для вычислений), то сначала разработаем *алгоритмы* основной программы и подпрограммы. Затем запрограммируем *отдельные блоки* алгоритмов (с помощью соответствующих операторов) и *структуры* (с помощью соответствующей последовательности операторов). Наконец запишем *полную Паскаль-программу* решения заданной задачи в полном соответствии с разработанными алгоритмами основной программы и подпрограммы, не нарушая их структур и связей между отдельными блоками алгоритмов.

8.1. Разработка алгоритма решаемой задачи

Анализ условия задачи показывает, что для вычисления всех значений функции y_k необходимо организовать цикл с параметром k, изменяющимся от 1 до m с шагом 1. Внутри этого цикла имеется разветвление на два направления, при этом для проверки условия разветвления предварительно необходимо определить минимальное значение элемента массива x_i, используя циклический процесс перебора и сравнения между собой всех элементов этого массива.

Так как минимальное значение элемента массива x_i необходимо определять среди исходных данных, то это можно сделать сразу после ввода исходных данных.

При организации циклических процессов рекомендуется выносить за пределы цикла те вычисления, которые достаточно выполнить один раз, что уменьшает затраты времени на решение задачи. Поэтому за пределы цикла в нашем примере целесообразно вынести вычисление величин $\beta = 2t$ и $\alpha = \sin(3\beta)$.

Тогда, с учетом проведенного анализа условия задачи, алгоритм ее решения будет иметь вид, представленный на рис. 8.1. В блоке 2 осуществляется как ввод, так и вывод исходных данных с целью контроля за правильностью ввода их в персональный компьютер. В блоке 2 осуществляется

вычисление b (наименьшего по значению элемента одномерного массива x_i исходных данных) по подпрограмме с именем *Bmin* , схема алгоритма которой приведена на рис. 8.2.

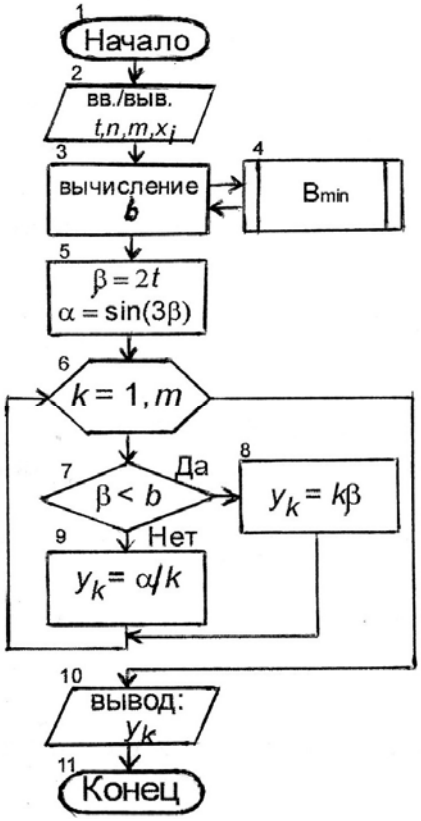


Рис. 8.1

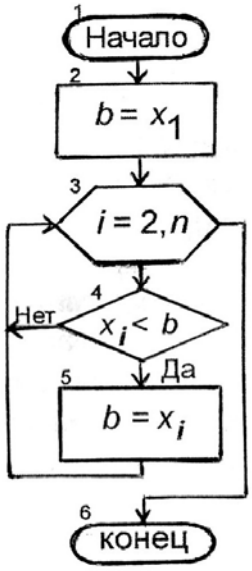


Рис. 8.2

8.2. Программирование отдельных блоков и структур разработанного алгоритма решаемой задачи

Так как подпрограмма по структуре совпадает с основной программой и записывается в ее описательной части, то сначала запишем на Паскале подпрограмму в соответствии с ее алгоритмом на рис. 8.2, например, в виде процедуры пользователя без формальных параметров.

Программирование блока 1. Этот блок включает заголовок подпрограммы, описательные разделы и операторную скобку *begin* раздела операторов. Пусть в нашей подпрограмме (с именем *Bmin*) глобальными переменными будут *b, n, x_i,* а локальной переменной будет *i.* Тогда блок 1 на Паскале можно записать так:

```
Procedure Bmin;
 Var i : integer;
   begin
```

Программирование блока 2 и 5. Эти блоки на Паскале записываются с помощью операторов присваивания:

```
b : = x[1];     b : = x[i];
```

Программирование циклической структуры из блоков 3-5. Блоки 4 и 5, являющиеся телом цикла, представляют собой разветвляющуюся структуру, которую можно записать на Паскале или с помощью оператора **If**, или оператором **Case** следующим образом:

```
If x[i] < b then        или      Case  x[i] < b  of
b : = x[i];                      True :    Ь : = x[i]  end ;
```

Теперь циклическая структура из блоков *3-5* на Паскале будет иметь вид:

```
For i:= 2  to  n  do        или      For  i : = 2  to  n   do
If x[i] < b then                        Case x[i] < x_i of
  b : = x[i];                           True :   b : = x[i]   end ;
```

Программирование блока 6. Этот блок на Паскале записывается операторной скобкой ***End;***

Итак, весь алгоритм (см. рис. 8.2) на Паскале можно записать в виде процедуры пользователя следующим образом:

```
Procedure Bmin;          или      Procedure Bmin;
   Var i : integer;                   Var i : integer;
      begin                              begin
   b : = x[1];                        b : = x[1];
For i:= 2  to  n  do                For  i : = 2  to  n   do
If x[i] < b then                      Case x[i] < b of
   b: = x[i];                           True : b : = x[i]    end;
    end ;                                 end ;
```

А теперь составим Паскаль-программу по алгоритму рис.8.1.

Программирование блока 1. Этот блок включает заголовок программы (пусть имя программы будет Primer), описательные разделы и операторную скобку *Begin* раздела операторов:

```
Program Primer; { заголовок программы }
Var             {раздел описания переменных}
   b, t, beta, alfa : real ;
   k, n, m : integer ;
   x, y: array [1..99] of real ;
Procedure Bmin; { раздел описания процедуры}
Var i : integer ;
begin
b : = x[1];
For  i : = 2 to n do
If  x[i] < b then
b : = x[i] ;
Begin { операторная скобка }
```

Программирование блока 2. Сначала зададимся формой ввода численных значений переменных:

1) переменные будем вводить в таком порядке: *n, m, t, x_i* ;

2) численные значения этих переменных будем набирать на клавиатуре: *n, m, t* в одной строке через пробел « _ » ; x_i – по одному элементу в строке, т.е.:

```
7 _ 20 _ 0.6
1.7 E + 1
- 0. 23 E + 2
2.07
- 1.3
23.2 E - I
```

75.3
-5.*5* E - I

Теперь запишем соответствующие операторы ввода:

```
ReadLn (n, m, t);
For k: = 1 to n do
   ReadLn  (x[k]) ;
```

Зададимся следующей формой вывода значений исходных данных на экран дисплея:

```
_ _ИСХОДНЫЕ_ _ДАННЫЕ
п =7_ _m = 2О_ _ t = _б.0000000000Е-01
x [ _1 ] = _ 1.7000000000E+01
x [ _2 ] =  - 2.3000000000E+01
x [ _3 ] = _ 2.0700000000E+00
x [ _4 ] = _ 1.3000000000E+00
x [ _5 ] = -  2.3200000000E+00
x [ _6 ] =  _7.5300000000E+01
x [ _7 ] = - 5.5000000000E -01 .
```

Теперь запишем соответствующие операторы вывода:

```
WriteLn ( '_ _ИСХОДНЫЕ_ _ ДАННЫЕ ');
WriteLn ('n = ',  n ,  '_ _m =',  m ,  '_ _ t = ',  t);
For k : = 1 to n do
WriteLn (  x [ ' ,  k : 2 ,  ' ] = ',  x [ k ] ) ;
```

Программирование блока 3. Этот блок записывается оператором обращения к процедуре:

Bmin;

Программирование блоков 5, 8, 9. Эти блоки на Паскале записываются с помощью операторов присваивания:

```
Beta : =  2 * t ;
Alfa : = sin (3 * beta);
y [ k ] : =  k * beta ; { блок 8 }
y [ k ] : = alfa / k ;    { блок 9 }
```

Программирование циклической структуры, состоящей* из *блоков 6-9. Тело цикла составляют блоки 7-9, которые запишем с помощью оператора If . Тогда циклическая структура на Паскале будет иметь вид:

```
For k  : =  1 to m do
  If beta  < и then  y [ k ] : =  k * beta
    else y [ k ] : =  alfa / k ;
```

Программирование блока 10. Зададимся следующей формой вывода результатов вычисления на экран дисплея:

```
_ _ Результаты _ _вычислений
y [ _1 ] = ± Ц . ЦЦЕ±ЦЦ;
y [ _2 ] = ± Ц . ЦЦЕ±ЦЦ;
  ……
y [2 0 ] = ± Ц . ЦЦЕ±ЦЦ,
```

где символ Ц означает одну из десятичных цифр.

Теперь блок 10 на Паскале можно записать так:

```
WriteLn ( '_ _ Результаты _ _вычислений ' ) ;
For k : = 1 to m do
        WriteLn ( ' y [ ',  k : 2 ,  ' ] = ',  y [ k ] : 9 ) ;
        WriteLn ( ' Нажмите клавишу Enter ' ) ;
        ReadLn ;
```

Здесь оператор ***ReadLn;*** используется для просмотра выводимых ПК на дисплей результатов вычислений до тех пор, пока вы не нажмете клавишу ввода ***Enter*** (подсказка с помощью оператора ***WriteLn (‘ Нажмите клавишу Enter ‘);***).

Программирование блока 11. Этот блок на Паскале записывается закрывающей операторной скобкой ***End .***

8.3. Полная Паскаль-программа решаемой сложной задачи

Итак, полная Паскаль-программа, составленная в соответствии с алгоритмом рис. 8.1, будет иметь следующий вид:

```
    { Блок 1 }

  Program Primer;
  Var b, t, beta, alfa : real;
      к, n, m : integer;
      x, y: array [1 .. 99] of real;
  Procedure Bmin;
  Var i : integer;
    Begin
  b : = x [1];
  For  i : = 2 to n do
    If x[i] < b then
      b : = x [i];
     end ;
   Begin

   { Блок 2 }
   ReadLn (n, m, t);
   For k : = 1 to n  do
     ReadLn (x [k]);
   WriteLn (‘_ _ИСХОДНЫЕ_ _ ДАННЫЕ ’);
   WriteLn (‘ n =’,  n ,  ’_ _m =’,  m ,  ’_ _t = ',  t);
    For k : = 1 to n do
      WriteLn (‘x [’,  k : 2,  ’] = ‘,  x [k]);
      { Блок 3 }
        Bmin;
      { Блок 5 }
   Beta : =  2 *  t ;
   Alfa : = sin (3 * beta);
{Структура из блоков  6 - 9 }

    For k  : =  1 to m do
         If beta  < и then  y [ k ] : =  k * beta
                    else y [ k ] : =  alfa / k ;
       { Блок 10 }
    WriteLn ( ‘_ _ Результаты _ _ вычислений ‘ ) ;
    For k: = 1 to m do
      WriteLn (‘y[’,  k : 2,  ’] = ‘,  y[k] : 9);
          WriteLn ( ‘ Нажмите клавишу Enter ‘ ) ;
            ReadLn ;
{ Блок 11 }
  End.
```

ЛИТЕРАТУРА

1. Вальвачев, А.Н. Программирование на языке Паскаль для персональных ЭВМ ЕС / А.Н Вальвачев, В.С. Крисевич. – Минск: Выш. шк., 1989. – 223 с.

2. ГОСТ 19.701-90. ЕСПД. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения. – М.: Госстандарт, 1990. – 28 с.

3. Дембовский, Л.М. Основы алгоритмизации и программирования. Лабораторные работы (практикум) для студентов всех форм обучения специальности 1-40 01 01 «Программное обеспечение информационных технологий» / Л.М. Дембовский. – Минск: БНТУ, 2004. – 98 с.

4. Офицеров, Д.В., Старых, В.А. Программирование в интегрированной среде Турбо-Паскаль / Д.В. Офицеров, В.А. Старых. – Минск: Беларусь, 1992. – 240 с.

5. Паскаль для персональных компьютеров / Ю.С. Бородич [и др.]. – Минск: Выш. шк., 1991. – 365 с.