ГЛАВА ПЯТАЯ

АЛГОРИТМИЗАЦИЯ И ПРОГРАММИРОВАНИЕ

§ 5.1. АЛГОРИТМИЗАЦИЯ

Каждый из нас постоянно решает множество задач: как быстрее добраться на работу, как лучше спланировать дела текущего дня и многие другие. Решение каждой задачи всегда делится на простые действия, составляющие алгоритм.

Алгоритм — это любая последовательность действий, приводящая к решению поставленной задачи.

Слово «алгоритм» появилось в Средние века, когда европейцы познакомились со способами выполнения арифметических действий в десятичной системе счисления, описанными узбекским математиком Мухаммедом бен Муса аль-Хорезми («аль-Хорезми» — человек из города Хорезми; в настоящее время город Хива в Хорезмской области Узбекистана). Слово «алгоритм» есть результат европейского произношения слов «аль-Хорезми».

Алгоритм характеризуется следующими свойствами: дискретностью, массовостью, определенностью, результативностью.

Дискретность — это свойство, означающее следующее: каждый алгоритм состоит из отдельных законченных действий, т. е. «делится на шаги».

Массовость — применимость алгоритма ко всем задачам рассматриваемого типа при любых исходных данных.

Определенность — свойство алгоритма, заключающееся в строгом определении содержания и порядка выполнения отдельных шагов.

Результативность — свойство, состоящее в том, что любой алгоритм должен находить решение за конечное число шагов.

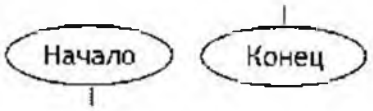
Существует несколько способов описания алгоритмов: словесное описание, блок-схема, алгоритмический язык и программа.

Словесное описание представляет структуру алгоритма на естественном языке. Например, любой прибор бытовой техники (утюг, электропила, дрель и т. п.) имеет инструкцию по эксплуатации, т. е. словесное описание алгоритма, в соответствии с которым данный прибор должен использоваться.

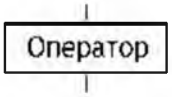
Запись алгоритма осуществляется в произвольной форме на естественном языке, например русском. Этот способ описания не имеет широкого распространения, так как строго не формализуем, допускает неоднозначность толкования при описании некоторых действий, страдает многословностью.

Блок-схема — описание структуры алгоритма с помощью геометрических фигур с линиями-связями, показывающими порядок выполнения отдельных инструкций. Этот способ имеет ряд преимуществ. Благодаря наглядности он обеспечивает «читаемость» алгоритма и явно отображает порядок выполнения отдельных команд. В блок-схеме каждой формальной конструкции соответствует определенная геометрическая фигура или связанная линиями совокупность фигур. К основным геометрическим фигурам, используемым для построения блок-схем, относятся следующие.

Блоки, характеризующие *начало* и *конец* алгоритма:

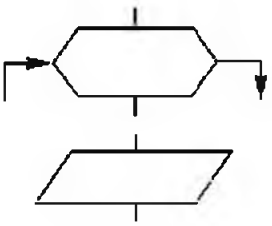


Блок, отображающий *процесс* (*оператор*), предназначенный для описания отдельных действий:



Блок, содержащий *проверку условия*, или *условный блок*:

Блок, описывающий *цикл с параметром*:



Блок *ввода/вывода* с произвольного носителя информации:

Описание алгоритма в словесной форме или в виде блок-схемы допускает некоторый произвол при изображении команд. Вместе с тем оно позволяет человеку легко понять суть дела и исполнить алгоритм.

Алгоритмический язык, именуемый как *псевдокод*, — это запись алгоритмов, во многом напоминающая запись алгоритма на естественном языке и языке программирования. При описании алгоритма на псевдокоде используются следующие конструкции:

нц — начало цикла; кц — конец цикла; для — цикл с параметром; если — условие; то — результат выполнения условия; иначе — результат невыполнения условия; всё — конец условия; пока — условие цикла.

Рассмотрим примеры блок-схем трех основных видов алгоритмов: линейного, разветвляющегося и циклического.

Линейным называется алгоритм, в котором все этапы решения задачи выполняются строго последовательно.

Блок-схема линейного алгоритма нахождения периметра прямоугольного треугольника P при известных длинах его катетов a, b изображена на рис. 5.1.

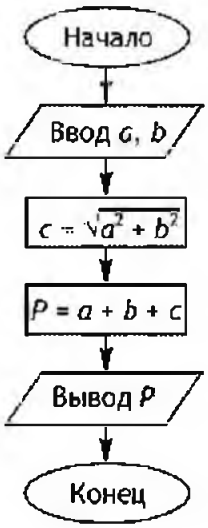


Рис. 5.1

Разветвляющийся алгоритм — это такой алгоритм, в котором выбирается один из нескольких возможных путей вычислительного процесса. Каждый подобный путь называется ветвью алгоритма. Признаком разветвляющегося алгоритма является наличие условия.

Различают неполное (*если–то*) и полное (*если–то–иначе*) виды ветвления.

Неполное ветвление предполагает наличие оператора только на одной ветви (*то*; *Да*; *Истина*), на другой ветви оператор отсутствует и управление сразу переходит к точке слияния (рис. 5.2*а*).

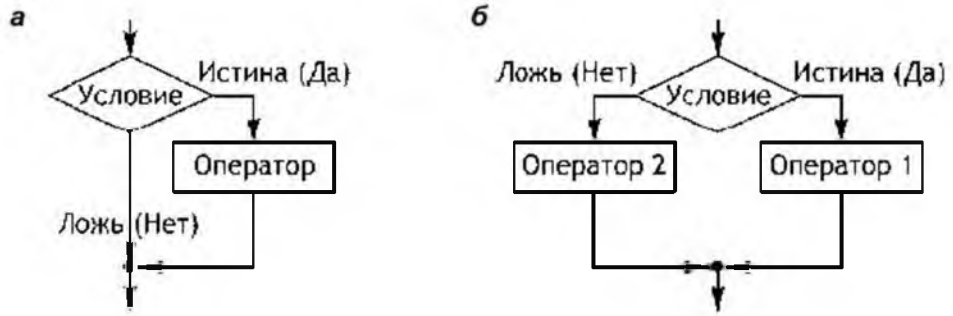


Рис. 5.2

Полное ветвление позволяет организовывать две ветви в алгоритме (*то* или *иначе*; *Да* или *Нет*; *Истина* или *Ложь*), каждая из которых ведет к общей точке их слияния (рис. 5.2*б*).

В двух блок-схемах алгоритмов определения наименьшего числа *min* среди трех чисел *a*, *b*, *c* реализовано неполное (рис. 5.3*а*) и полное (рис. 5.3*б*) ветвление.

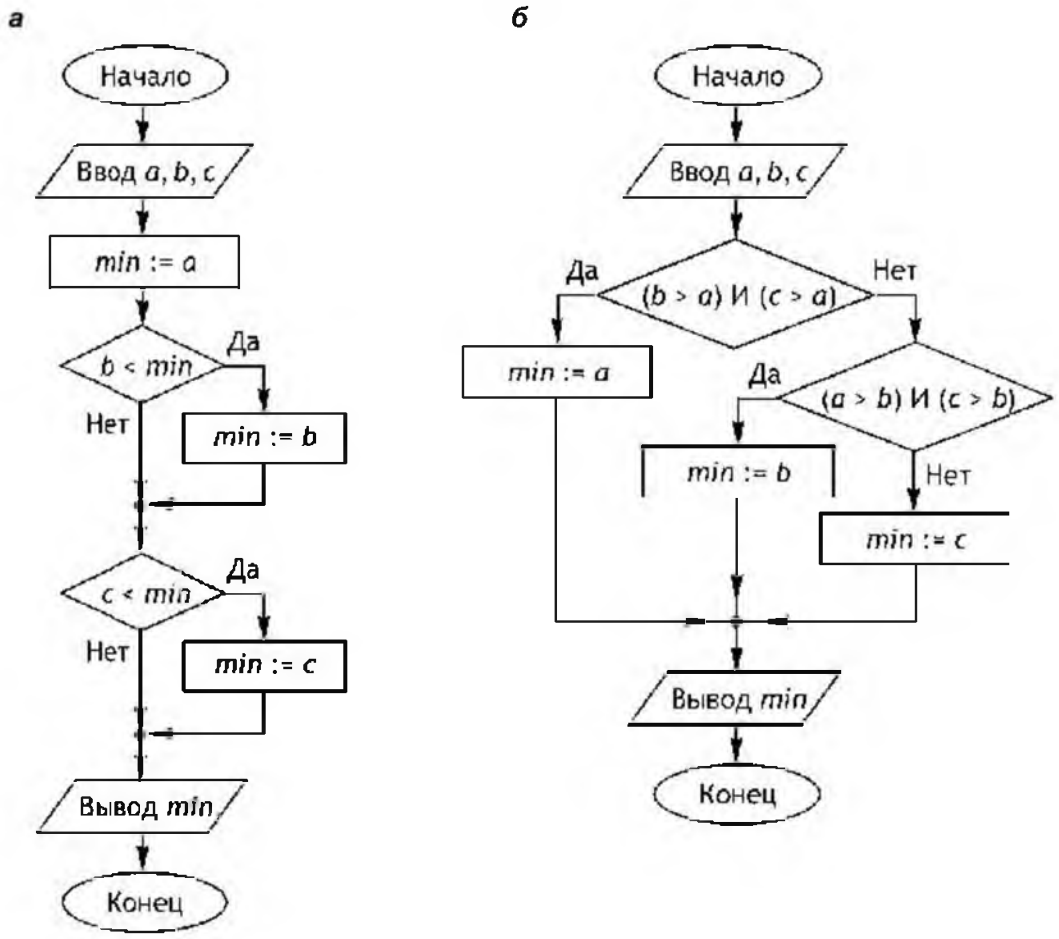


Рис. 5.3

а

```
Ввод a,b,c
min:=a
если b<min
то min:=b
всё
если c<min
то min:=c
всё
Вывод min
```

б

```
Ввод a,b,c
если (b>a)и(c>a)
то min:=a
  иначе
  если (a>b)и(c>b)
  то min:=b
  иначе
  min:=c
  всё
всё
Вывод min
```

Рис. 5.4

Им соответствуют программы на псевдокоде, показанные на рис. 5.4.

Основная идея алгоритма поиска наименьшего значения *min* среди трех чисел *a*, *b*, *c* на рис. 5.3*а* сводится к следующему: за наименьшее *min* принимается одно из чисел, например *a*, и осуществляется попарное сравнение всех чисел, сопровождающееся исключением наибольшего из каждой пары и завершающееся нахождением наименьшего числа.

Выбор минимального значения *min* в алгоритме на рис. 5.3*б* осуществляется, исходя из соотношения

$$min = \begin{cases} a, & \text{если } b > a \text{ и } c > a; \\ b, & \text{если } a > b \text{ и } c > b; \\ c, & \text{в противном случае.} \end{cases} \qquad (5.1)$$

Часто при выборе одного из операторов приходится проверять значение некоторого выражения на принадлежность к определенному интервалу или набору данных. Для этого существует алгоритмическая структура *Выбор*, пример которой приведен на рис. 5.5. Здесь в зависимости от целочисленного значения времени T выводится одно из сообщений: «Доброе утро» ($5 \le T \le 11$), «Добрый день» ($12 \le T \le 16$), «Добрый вечер» ($17 \le T \le 21$) и «Доброй ночи» (в оставшееся время — от 22 до 4).

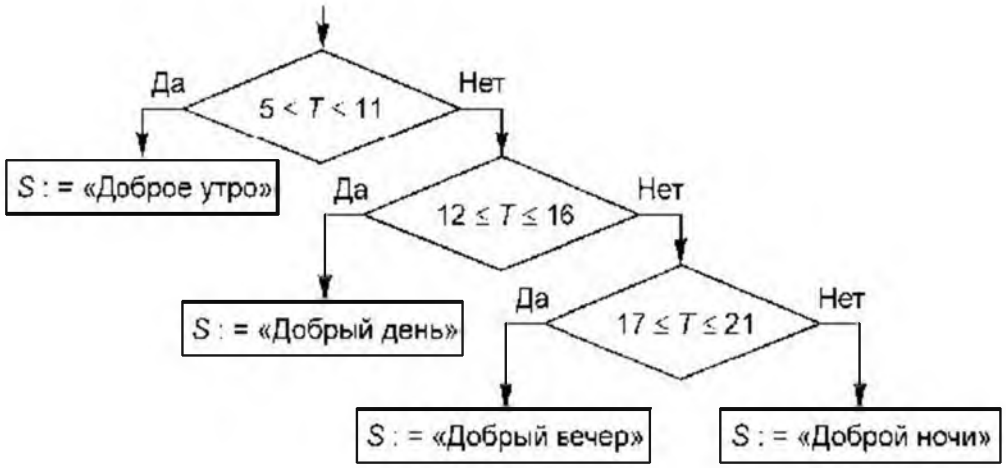


Рис. 5.5

Циклическим, или просто *циклом*, называют такой алгоритм, в котором получение результата обеспечивается многократным выполнением одних и тех же операций. Группа повторяющихся операций называется *телом цикла*.

Широкое применение получили три типа циклов: цикл с параметром, цикл с предусловием и цикл с постусловием, блок-схемы которых приведены на рис. 5.6.

Параметр i цикла *с параметром* на рис. 5.6*а* изменяется от n до k с шагом h следующим образом:

на первом шаге $i = n$;

на втором шаге $i = n + h$;

на третьем шаге $i = n + 2h$;

.....

на последнем шаге $i = k$.

Рис. 5.6

Цикл с параметром используется в тех случаях, когда известна величина k, т. е. количество элементов или шагов цикла.

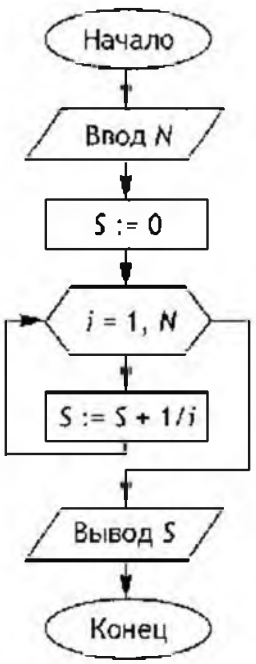
Количество шагов цикла *с предусловием* (см. рис. 5.6*б*) заранее не определено. В нем сначала проверяется выполнение условия. Если оно *Истинно* (*Да*), то исполняется тело цикла, после чего вновь проверяется условие. Указанные действия проверяются до тех пор, пока условие не примет значение *Ложно* (*Нет*).

Цикл *с постусловием* (см. рис. 5.6*в*) отличается от цикла с предусловием расположением условия и тем, что тело цикла всегда будет выполнено хотя бы один раз. Тело этого цикла будет выполняться, пока условие *Ложно* (*Нет*).

Цикл с параметром можно применить в задаче вычисления суммы S элементов $1, 1/2, 1/3, \ldots, 1/N$ до заданного числа N. Алгоритм решения данной задачи представлен блок-схемой на рис. 5.7*а* и программой на псевдокоде (рис. 5.7*б*). На рис. 5.7*а* не указана величина шага, поскольку она равна 1.

Если видоизменить предыдущую задачу, а именно определить количество элементов i суммы $S = 1 + 1/2 + 1/3 + \ldots + 1/i$, величина которой должна быть не меньше заданной S_2, то потребуются циклы с предусловием или по-

а



б

```
Ввод N
S:=0
нц для i от 1 до N
  S:=S+1/i
кц i
Вывод S
```

Рис. 5.7

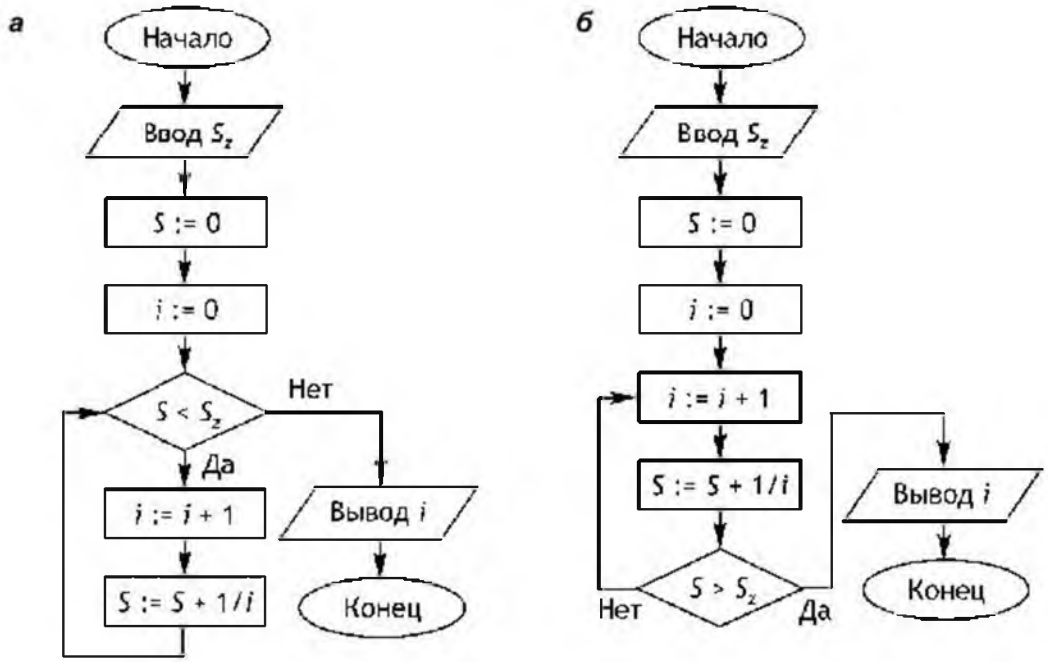


Рис. 5.8

стусловием, содержащиеся в блок-схемах на рис. 5.8*а* и рис. 5.8*б* соответственно.

Ниже приводятся программы на псевдокоде, реализующие решение задачи и содержащие циклы с предусловием (рис. 5.9*а*) и постусловием (рис. 5.9*б*).

Для повышения производительности и качества работы каждый язык программирования имеет структурированный тип данных — *массив*.

Массивом называется упорядоченная совокупность однотипных величин, имеющих общее имя, элементы которой различаются порядковыми номерами, именуемыми индексами.

а

```
Ввод Sz
S:=0
i:=0
нц пока S<Sz
  i:=i+1
  S:=S+1/i
кц
Вывод i
```

б

```
Ввод Sz
S:=0
i:=0
нц
  i:=i+1
  S:=S+1/i
кц до S>Sz
Вывод i
```

Рис. 5.9

В качестве иллюстрации можно представить множество пронумерованных ящиков (совокупность — «Ящик № 1», «Ящик № 2», «Ящик № 3» и т. д.; «Ящик» — общее имя всех ее элементов). Доступ к конкретному ящику (элементу массива) осуществляется после выбора ящика по его номеру (индексу). Элементы массива в памяти компьютера хранятся по соседству. Массивы различаются количеством индексов, определяющих их элементы.

Одномерный массив (множество ящиков в один ряд) предполагает, что каждый его элемент имеет только один индекс. Количество элементов массива называют *размерностью*. При определении одномерного массива его размерность записывается в круглых скобках, рядом с его именем. Например, если сказано: «задан массив $a(5)$», это означает, что даны элементы a_1, a_2, ..., a_5. Рассмотрим алгоритмы обработки элементов одномерных массивов. Начнем с алгоритма ввода элементов массива $a(5)$ в порядке возрастания их индексов (рис. 5.10*а*).

В *двумерном массиве* $a(n, m)$ (множество ящиков, расположенных по горизонтали и по вертикали) каждый элемент a_{ij} имеет два индекса: первый индекс i определяет номер строки, в которой находится элемент (координата по горизонтали), а второй j — номер столбца (координата по вертикали):

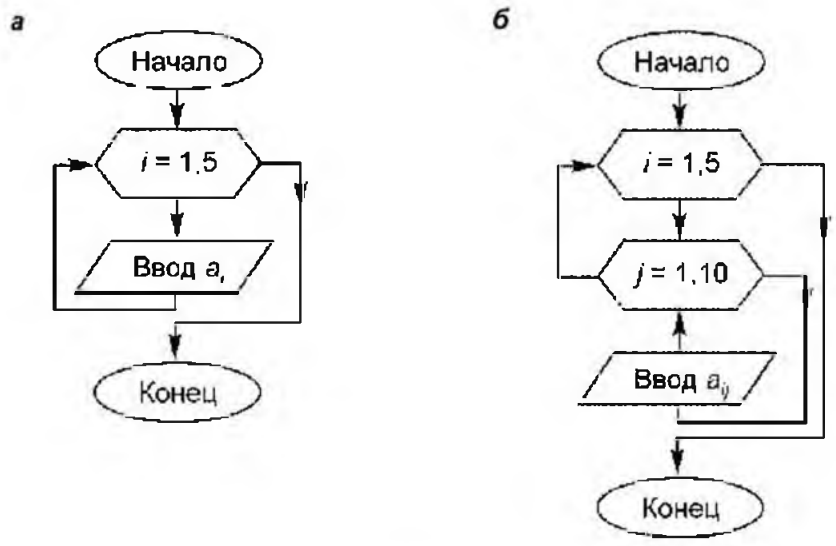


Рис. 5.10

$$a(n,m)=\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1j} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2j} & \dots & a_{2m} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ a_{i1} & a_{i2} & \dots & a_{ij} & \dots & a_{im} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nj} & \dots & a_{nm} \end{pmatrix}.$$

Двумерный массив характеризуется двумя размерностями n и m, определяющими число строк и столбцов соответственно.

Ввод элементов двумерного массива осуществляется построчно, в свою очередь ввод каждой строки производится поэлементно, тем самым определяется циклическая конструкция, реализующая вложение циклов. Внешний цикл определяет номер вводимой строки (i), внутренний — номер элемента по столбцу (j). На рис. 5.10*б* представлен алгоритм ввода матрицы $a(5, 10)$.

На рис. 5.11 приводятся программы на псевдокоде, осуществляющие ввод элементов одномерного $a(5)$ (рис. 5.11*а*) и двумерного $a(5, 10)$ (рис. 5.11*б*) массивов.

Завершим изучение процедур обработки элементов массивов алгоритмами вычисления сумм положительных Sp и отрицательных Sn элементов a_i, $i = 1, 2, \dots, 5$, (см. рис. 5.12*а*) одномерного массива $a(5)$ и элементов a_{ij}, $i = 1, 2, \dots, 5$, $j = 1, 2, \dots, 10$, двумерного массива $a(5, 10)$ (рис. 5.12*б*), а также соответствующими программами на псевдокоде, изображенными на рис. 5.13.

а

```
нц для i от 1 до 5
  Ввод a_i
кц
```

б

```
нц для i от 1 до 5
  нц для j от 1 до 10
    Ввод a_ij
  кц j
кц i
```

Рис. 5.11

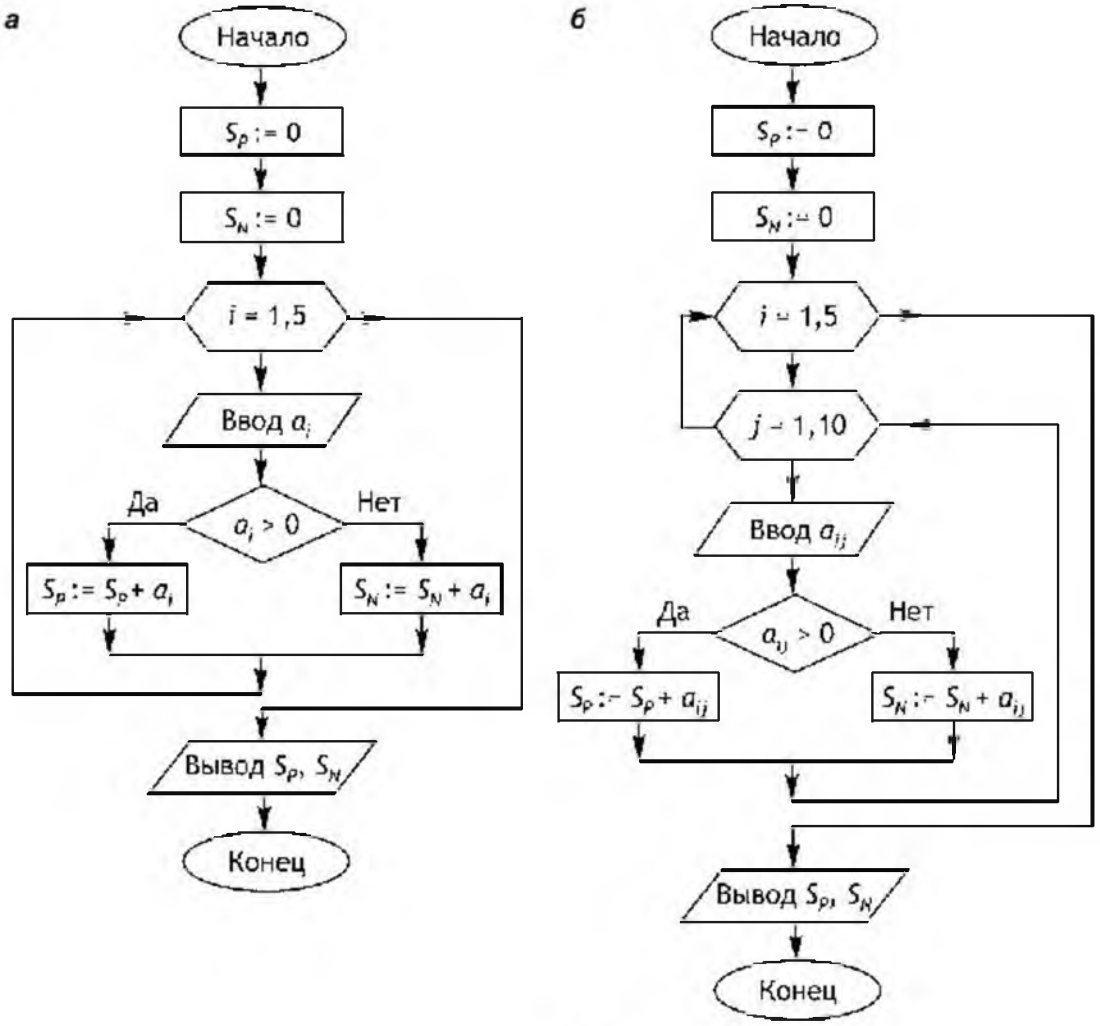


Рис. 5.12

а

```
Sр:=0
Sn:=0
нц для i от 1 до 5
   Ввод ai
   если ai>0
   то Sр:=Sр+ai
   иначе Sn:=Sn+ai
   всё
кц i
Вывод Sр, Sn
```

б

```
Sр:=0
Sn:=0
нц для i от 1 до 5
   нц для j от 1 до 10
      Ввод aij
      если aij>0
      то Sр:=Sр+aij
      иначе Sn:=Sn+aij
      всё
   кц j
кц i
Вывод Sр, Sn
```

Рис. 5.13

Сочетание циклов с параметром и постусловием можно продемонстрировать на примере сортировки элементов одномерного массива $a_1, a_2, \ldots, a_5$ методом «пузырька».

При прохождении массива $a_1, a_2, \ldots, a_5$ от начала до конца сравниваются пары соседних чисел a_i и a_{i+1}, $i = 1, 2, 3, 4$. Если очередная пара удовлетворяет условию $a_i > a_{i+1}$, т. е. нарушается порядок возрастания, то элементы a_i и a_{i+1} меняются местами. Действия повторяются до тех пор, пока очередной проход не вызовет ни одной перестановки. В результате все элементы будут располагаться в порядке возрастания.

На рис. 5.14*а* приводится программа на псевдокоде, а на рис. 5.14*б* — блок-схема алгоритма сортировки методом «пузырька».

а

```
нц для i от 1 до 5
  Ввод a_i
кц i
нц
  Ind=1
  нц для i от 1 до 4
    если a_i>a_i+1
    то b:=a_i
    a_i:=a_i+1
    a_i+1:=b
    Ind=0
    всё
  кц i
кц пока Ind=0
нц для i от 1 до 5
  Вывод a_i
кц i
```

б

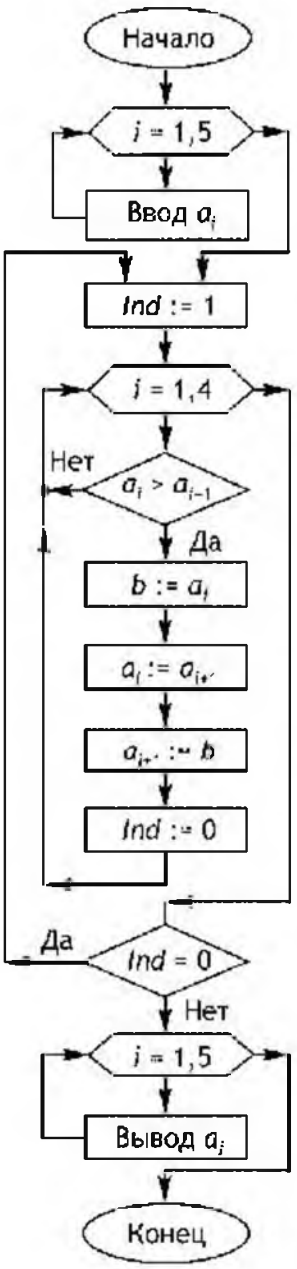


Рис. 5.14

Здесь используется параметр Ind, равный 1, если массив отсортирован, и равный 0, если произошла перестановка соседних элементов. Операция перестановки элементов a_i и a_{i+1} с помощью промежуточного элемента b содержит 3 оператора — $b := a_i$; $a_i := a_{i+1}$; $a_{i+1} := b$.

На практике исполнителями алгоритмов являются компьютеры. Поэтому алгоритм, предназначенный для исполнения на компьютере, должен быть записан на «понятном» ему языке, такой формализованный язык называют *языком программирования*.

§ 5.2. ЭВОЛЮЦИЯ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ

Языки программирования являются искусственными языками со строго определенными синтаксисом и семантикой.

Синтаксис — это набор правил, которые определяют основные внутренние структуры и последовательности символов, допустимых в языке программирования.

Семантика — это значения языковых единиц (слов и предложений).

Составление программ для ЭВМ первого поколения велось на *машинном языке*, который представляет собой свод правил кодирования действий ЭВМ с помощью двоичных чисел.

Более высоким уровнем по сравнению с машинными языками являются *машинно-ориентированные* языки символического кодирования. Основной принцип при создании языков символического кодирования состоит в замене машинных кодов на их буквенные обозначения. Такой машинно-ориентированный язык получил название языка *Ассемблера* (расширение .asm). ЭВМ «понимает» только машинный язык, только команды, операнды и адреса, записанные с помощью двоичных чисел. Поэтому для преобразования программы, написанной на языке Ассемблера, в машинные коды необходим *транслятор* (переводчик) — специальная программа, которая имеет созвучное название: ассемблер. Недостатком машинно-ориентированных языков является их зависимость от типа машины.

На следующем уровне развития языков находятся *процедурно-ориентированные* языки. В отличие от машинно-ориентированных языков, синтаксис и семантика этих языков не зависят от типа конкретной ЭВМ (конкретного процессора). Привязку составленной программы к конкретному типу ЭВМ осуществляет транслятор (программа-переводчик). Запись программы на *процедурно-ориентированном языке* достаточно близка к общепринятой математической записи, компактна и удобна для восприятия. Рассмотрим, как выглядит операция суммирования двух чисел, например 7 и 5, запрограммированная на различных языках.

1) на машинном языке:

Адрес	Команда
1101 0001	0011 1111
1101 0010	0000 0101
1101 0011	0000 0110
1101 0100	0000 0111
1101 0101	1000 0000

2) на машинно-ориентированном языке (микропроцессорный комплект 580-й серии): MVI A, 5 MVI B, 7 ADD B;

3) на процедурно-ориентированном языке: A = 7 + 5.

Из приведенных примеров видно, что наиболее проста для понимания последняя запись.

Одним из первых *процедурно-ориентированных* языков стал язык *Фортран* (FORTRAN, FORmula TRANslation — преобразование формул), созданный в начале 1950-х гг. в США фирмой IBM. Он не только просуществовал до наших дней, но продолжает развиваться и удерживает одно из первых мест в мире по распространенности. Среди причин такого долголетия можно отметить простую структуру Фортрана. Он используется в научных и инженерно-технических вычислениях.

Фортран положен в основу диалогового языка *Бейсик* (BASIC — Beginners All-Purpose Symbolic Instruction Code). BASIC переводится так: многоцелевой язык символических

команд для начинающих. Вначале он предназначался в основном для обучения программированию. Современные версии языка BASIC (расширение .bas) позволяют решать задачи на профессиональном уровне.

В 1960-х гг. появился *Алгол-60* (ALGOL, ALGOritmic Language — алгоритмический язык) — это более совершенный язык, чем Фортран. Он обладает большей гибкостью и надежностью программ.

В 1971 г. Н. Виртом для обучения программированию был разработан язык *Паскаль* (PASCAL), который является преемником Алгола-60. Он имеет конструкции, аналогичные существующим в Алголе-60, однако более лаконичен. В Паскале проводятся идеи *структурного программирования*. Благодаря хорошей структурированности программ, написанных на языке Паскаль, над разработкой сложных проектов могут одновременно работать несколько программистов. Программы, написанные на Паскале, имеют расширение .pas.

Языки *Пролог* и *Лисп* были созданы для решения задач искусственного интеллекта. Эти языки позволяют обрабатывать текстовую (символьную) информацию, решать логические и математические задачи.

Язык *Пролог* (PROLOG, PROgraming in LOGic — программирование в логике), созданный в 1973 г. французским ученым А. Кольмероэ, является непроцедурным языком *логического программирования*. Программа на языке Пролог (расширение .pro), опирающемся на теорию исчисления предикатов, строится из последовательности фактов и правил, затем формируется цель, которую Пролог пытается доказать (опровергнуть) с помощью механизма обратного вывода. Он выбран основным языком при разработке ЭВМ пятого поколения, которые будут обладать искусственным интеллектом.

Язык функционального программирования *Лисп* (LISP, LISt Processing — обработка списков), разработанный в 1959 г. Д. Маккарти, ориентирован на работу со структурой данных в форме списка. Лисп позволяет эффективно обрабатывать большие объемы текстовой информации и обладает единообразием программных структур и структур данных: все они записываются в виде списков.

Перспективным направлением дальнейшего развития технологии программирования явилось создание *объектно-ориентированных* языков. *Объекты* представляют собой многократно используемые программные модули. Структурно объекты состоят из двух частей: методов и переменных. Методы представляют собой набор процедур и функций, определяющих алгоритм работы объекта. *Переменные* могут содержать как простые *данные* (числа, массивы, текст и т. д.), так и информацию сложной структуры (графика, звуки и т. д.). Однотипные объекты объединяются в *классы*.

Объектно-ориентированное программирование (ООП) характеризуется тремя признаками: инкапсуляцией, наследованием и полиморфизмом.

С помощью *инкапсуляции* данные одного объекта могут быть защищены от других объектов. С помощью механизма *наследования* дочерний класс способен унаследовать от своего родительского класса все его методы и данные, причем потомок может унаследовать способности и от нескольких родителей. *Полиморфизм* — это присвоение единого имени процедуре, которая передается по иерархии объектов, с выполнением этой процедуры способом, соответствующим каждому объекту в иерархии.

Первый объектно-ориентированный язык программирования *Simula 67* был разработан в конце 1960-х гг. для решения задач моделирования. В настоящее время широко используются такие объектно-ориентированные языки, как *C++*, *Delphi*, *Java*, *Visual Basic*. Язык *Visual Basic* интегрирован в Microsoft Office: СУБД Access, электронные таблицы Excel, текстовый редактор Word.

Язык гипертекстовой разметки HTML (HyperText Markup Language) был предложен Тимом Бернерсом-Ли в 1989 г. в качестве одного из компонентов технологии разработки распределенной гипертекстовой системы World Wide Web (WWW).

Язык HTML (расширения .htm, .html) позволяет описывать структуру электронного документа с полиграфическим уровнем оформления. В основу гипертекстовой разметки была положена тэговая модель описания документа,

позволяющая представить документ в виде совокупности элементов, каждый из которых окружен тэгами. *Тэги* — это скобки, между которыми записан текст программы. Таким образом, гипертекстовая база данных в концепции WWW — это набор текстовых файлов, размеченных на языке HTML.

Для работы с Web-страницами были также разработаны интерпретируемые языки сценариев Perl, Java Script и VB Script.

Язык Perl (Practical Extraction and Report Language), созданный Л. Уоллом, переводится как язык для практического извлечения данных и составления отчетов. С помощью Perl, например, можно создавать скрипт, который открывает один или несколько файлов, обрабатывает информацию и записывает результаты. Аналогичные действия могут выполнить программы на языках JavaScript и VB Script, созданных на основе языков Java и Visual Basic соответственно.

§ 5.3. ПРОГРАММИРОВАНИЕ НА BASIC

BASIC — это один из самых простых языков программирования, успешно используемый для целей обучения и решения научно-технических и экономических задач. Предлагаемая версия языка применяется во всех приложениях MS и Open Office. В данном разделе рассматриваются основные элементы языка (переменные, константы, операции, операторы управления и циклы) и простые приемы традиционного программирования на BASIC.

5.3.1. ПЕРЕМЕННЫЕ И КОНСТАНТЫ

Переменная — это именованная область памяти, отведенная для хранения данных. Тип данных задает определенную форму или размер содержимого переменной. Оператор, определяющий с помощью ключевых слов **Dim** и **As** тип данных или переменной, имеет следующий синтаксис:

```
Dim имя переменной As тип данных
```

Имя переменной можно выбрать произвольное, соблюдая следующие правила:

- имя переменной должно начинаться с буквы;
- максимальная длина имени — 255 символов;
- имена могут содержать только буквы, цифры и символ подчеркивания;
- имя не может быть зарезервированным в BASIC словом (например, **Type**, **Print** и т. д.).

BASIC поддерживает следующие основные типы данных.

Данные типа **Boolean** могут принимать значения **True** или **False**, которым соответствуют числа 1 или 0.

Данные типа **Integer**, **Long** содержат целые числа:

- **Integer** от −32 768 до +32 767;
- **Long** от −2 147 483 648 до +2 147 483 647.

Данные типа **Single**, **Double** и **Currency** содержат числа с плавающей запятой. При этом переменная **Single** оперирует с числами до $3{,}4 \cdot 10^{32}$, а **Double** — с числами до $1{,}8 \cdot 10^{308}$. Данные типа **Currency** предназначены для выполнения финансовых расчетов и содержат 15 знаков до запятой и 4 — после.

Данные типа **String** служат для хранения строк, содержащих до $2 \cdot 10^{32}$ символов. Чтобы BASIC отличал строку от имени переменной, строка заключается в кавычки:

```
Dim Переменная As String
Переменная = "Hello"
```

Тип данных **Variant** либо объявляется, либо устанавливается по умолчанию в зависимости от содержимого переменной.

Если ее содержимое целое число, то она принимает тип **Integer**, если целое число с десятичной дробью — **Double**; если текст, то **String**. Во время выполнения программы переменная типа **Variant** может менять свой тип, что приводит к двум проблемам.

Во-первых, при чтении кода не видно, какой тип имеет переменная в данный момент, что сильно затрудняет обнаружение логических ошибок программирования.

Во-вторых, данные этого типа из-за частых внутренних преобразований занимают гораздо больше памяти, чем аналогичные данные, объявленные с указанием типа.

Для хранения большого объема данных одного типа применяются одномерные и многомерные массивы. Приведем пример объявления одномерного массива:

```
Dim Emp(10) As String
```

для хранения 11 элементов, и объявления двумерного массива:

```
Dim Loc(20,25) As Integer
```

описывающего матрицу с 21×26 элементами.

Константы — это элементы, не меняющие свои значения во время выполнения программы. Приведем пример описания константы:

```
Const NV As Integer = 1
```

Описание состоит из ключевого слова **Const**, за которым следует имя константы **NV**, тип **Integer** и значение 1, присваиваемое константе.

5.3.2. ОПЕРАТОРЫ И ОПЕРАЦИИ

Строка с кодом в исходном тексте программы на BASIC называется *оператором*. Оператор — это неделимое предложение, выполняющее какое-либо действие. Оператор может включать выражение и знак присвоения (=). В выражении переменные могут быть связаны математическими операциями сложения (+), вычитания (−), умножения (*), деления (/), возведения в степень (^).

Например, вычисление площади круга **S** с диаметром **D** может быть записано выражением

```
(3.14 * D ^ 2) / 4
```

Чтобы получить значение площади **S**, используют операцию присвоения (=):

```
S = (3.14 * D ^ 2) / 4
```

Теперь рассмотрим два оператора, выполняющих деление целых чисел.

Оператор **a\b** (в Паскале **div(a,b)**) возвращает целую часть числа от деления целого числа **a** на целое число **b**. Например, **7\3** или **div(7,3)** равно 2.

Оператор **a mod b** (в Паскале **mod(a,b)**) возвращает остаток от деления целого числа **a** на целое число **b**. Например, **7 mod 3** или **mod(7,3)** равно 1. Если одно число делится на другое без остатка, оператор **mod** возвращает значение 0. Например, **8 mod 4** равно 0.

В отличие от математических операций, результатом выполнения которых может быть любое числовое значение, операция отношения может иметь только два значения — **True** (Истина) и **False** (Ложь), которые могут принимать переменные типа **Boolean**.

В языке BASIC применяются следующие обозначения для операций отношения: = (равно); <> (не равно); > (больше); < (меньше); => (больше или равно); <= (меньше или равно).

Переменные, выражения или константы, связанные операциями отношения, будем называть *условиями*.

Например, для условия 25 <> 30 результатом будет **True** (Истина); для условий 25 < 30 — **True** (Истина); 25 > 30 — **False** (Ложь).

Логические операции применяются в логических выражениях. Если существует несколько вариантов выбора в операциях отношения или в условиях, то они связываются между собой логическими операциями.

Рассмотрим примеры логических операций (И — конъюнкция (**And**); ИЛИ — дизъюнкция (**Or**); НЕ — отрицание (**Not**)):

- (Условие 1) **And** (Условие 2) — результатом будет **True**, если оба условия имеют значения **True**, в противном случае — **False**;
- (Условие 1) **Or** (Условие 2) — результатом будет **False**, если оба условия имеют значения **False**, в противном случае — **True**;
- **Not** (Условие) — меняет значение условия на обратное, т. е. **False** на **True**, а **True** на **False**.

Теперь приведем операторы ввода:

```
a=InputBox("a=")
```

и вывода:

```
MsgBox "a=" & a
```

значений переменной **a**, снабженных подсказкой "**a=**".

```
Sub Fig5_1
Dim a As Single
Dim b As Single
Dim c As Single
Dim P As Single
a = InputBox("a=")
b = InputBox("b=")
c = Sqr(a^2+b^2)
P = a+b+c
MsgBox "P=" & P
End Sub
```

Рис. 5.15

Завершим подраздел созданием программы расчета периметра прямоугольного треугольника **P** с известными катетами **a** и **b** (рис. 5.15) на основании блок-схемы на рис. 5.1. Программный модуль в BASIC, как правило, оформляется в виде процедуры, которая начинается с ключевого слова **Sub** (процедура) и имени (в данном случае **Fig 5_1**) и завершается ключевыми словами **End Sub**. Имя процедуры может содержать скобки, например **Fig 5_1(a, b)**, что означает присутствие фактических параметров **a** и **b**, используемых в программе, именуемой телом процедуры.

Здесь в строках 2–5 описываются переменные, т. е. указывается их тип **Single**. Строки 6, 7 — операторы ввода значений переменных (катетов **a**, **b**). В строке 8 вычисляется гипотенуза **c**, а в строке 9 — периметр треугольника **P**. Строка 10 — оператор вывода значения периметра **P**.

5.3.3. УСЛОВНЫЕ ОПЕРАТОРЫ

В языке BASIC имеются различные операторы ветвления, называемые условными операторами и позволяющие сделать выбор: какие действия должны выполняться в зависимости от выполнения логических условий. Кратко опишем и приведем примеры применения основных операторов BASIC.

Условные операторы. Начнем с условных операторов, имеющих *однострочный* синтаксис и осуществляющих неполное и полное ветвление в зависимости от результатов анализа *одного* условия:

```
If условие Then
[Оператор]
End If
```

```
If условие Then
 [Оператор1]
Else
 [Оператор2]
End If
```

Если **условие** после **If** истинно, т. е. результат равен **True** (истина), выполняется **Оператор** при неполном ветвлении или **Оператор1** при полном ветвлении. В противном случае, когда результат равен **False** (ложь), при полном ветвлении выполняется **Оператор2**, а при неполном — не выполняется ничего.

На основании блок-схем на рис. 5.3 составим программы нахождения минимального **min** среди трех чисел **a**, **b**, **c** (рис. 5.16), использующие условные операторы.

а

```
Sub Fig5_3a
Dim a As Single
Dim b As Single
Dim c As Single
Dim min As Single
a = InputBox("a=")
b = InputBox("b=")
c = InputBox("c=")
min = a
If b<min Then
min = b
End If
If c<min Then
min = c
End If
MsgBox "min=" & min
End Sub
```

б

```
Sub Fig5_3b
Dim a As Single
Dim b As Single
Dim c As Single
Dim min As Single
a = InputBox("a=")
b = InputBox("b=")
c = InputBox("c=")
If (b>a)And(c>a) Then
  min = a
  ElseIf (a>b)And(c>b) Then
  min = b
  Else
  min = c
End If
MsgBox "min=" & min
End Sub
```

Рис. 5.16

В программе на рис. 5.16*а* имеются два условных оператора с однострочным синтаксисом, осуществляющие неполное ветвление.

В большинстве реальных случаев приходится выполнять несколько операторов на основании результатов анализа нескольких условий, применяя условный оператор с *многострочным* синтаксисом:

```
If условие1 Then
 [Оператор1]
Elseif условие2 Then
 [Оператор2]
```

```
Else
  [Оператор3]
End If
```

Такой условный оператор содержит программа на рис. 5.16*б*. Здесь простые условия **b>a** и **c>a**, **a>b** и **c>b**, объединенные логической операцией **And**, позволяют определить минимальные числа **a** и **b** согласно соотношению (5.1).

Еще одним условным оператором ветвления является **Select Case**, позволяющий выполнить одну или несколько групп операторов в зависимости от значения условия. Синтаксис оператора **Select Case** имеет следующий вид:

```
Sub Fig5_5
Dim T As Integer
Dim S As String
T=InputBox("T=")
Select Case T
  Case 5 To 11
   S="Доброе утро"
  Case 12 To 16
   S="Добрый день"
  Case 17 To 21
   S="Добрый вечер"
  Case Else
   S="Доброй ночи"
End Select
MsgBox S
End Sub
```

```
Select Case выражение
  Case значение 1
   [Оператор 1]
  Case значение 2
   [Оператор 2]
.............
  Case значение n
   [Оператор n]
  Case Else
   [Оператор n+1]
End Select
```

Рис. 5.17

Параметр **выражение** — любое числовое или строковое выражение. Значение выражения сравнивается с параметрами **значение 1**, **значение 2**, ..., **значение n**. Если хотя бы одно из сравнений удачное, то выполняется соответствующий **Оператор 1**, **Оператор 2**, ..., **Оператор n**, в противном случае **Оператор n+1**. В качестве параметров **значение** блока **Case** могут выступать одна (**1**) или несколько разделенных запятой (**2 , 3 , 4**) числовых величин, пределы изменения (**5 To 8**) или условия (**Is >= 9**). Применяя

оператор **Select Case**, можно составить программу вывода сообщения о времени суток (рис. 5.17) на основании блок-схемы алгоритма, приведенного на рис. 5.5. Здесь значениями параметра **T** блока **Case** являются пределы изменения или интервалы времени **5 To 11**, **12 To 16** и **17 To 21**.

5.3.4. ЦИКЛЫ

Циклы предназначены для многократного выполнения одного или нескольких операторов. В языке BASIC имеются две конструкции: цикл **For**... **Next**, дающий возможность устанавливать число проходов, и циклы **While**... **Wend** и **Do**... **Loop**, завершающиеся при выполнении заданного условия.

Цикл **For**... **Next** является самой старой и самой простой конструкцией:

```
For Счетчик=Начало To Конец [Step шаг]
 [Операторы]
Next [Счетчик]
```

Перед выполнением цикла значение **Счетчик** устанавливается в **Начало**. При каждом проходе переменная **Счетчик** увеличивается на 1 или на величину **шаг** ≠ 1 до тех пор, пока переменная **Счетчик** не станет больше, чем значение **Конец**. На рис. 5.18 приводится программа вычисления суммы **S** чисел 1, 1/2, 1/3, ..., 1/*N* до некоторого заданного числа *N* согласно блок-схеме алгоритма на рис. 5.7. В этом случае известно заранее число элементов суммы, поэтому используется оператор **For**... **Next**.

Если число элементов цикла неизвестно, применяются другие разновидности циклов: **While**... **Wend** и **Do**... **Loop**.

Цикл **While**... **Wend** является циклом с предусловием и имеет следующий синтаксис:

```
While условие
 [Операторы]
Wend
```

```
Sub Fig5_7
Dim N As Integer
Dim S As Single
Dim i As Integer
N = InputBox("N=")
S = 0
For i = 1 To N
S = S+1/i
Next i
MsgBox "S=" & S
End Sub
```

Рис. 5.18

В зависимости от позиции условия различают два варианта цикла с постусловием **Do**... **Loop**, содержащего выражение **While** или **Until**:

```
Do
  [Операторы]
Loop [While|Until] условие
```

Слово **While** вызывает очередной проход цикла, если **условие** истинно, а **Until** — если **условие** ложно. На рис. 5.19 приведены программы определения количества i чисел $1, 1/2, 1/3, \ldots, 1/i$, при котором их сумма **S** должна стать не меньше заданной **Sz**. Программа на рис. 5.19*а* построена на основании оператора цикла с предусловием **While**... **Wend** и блок-схемы на рис. 5.8*а*, а программа на рис. 5.19*б* — на основании оператора цикла с постусловием **Do**... **Loop Until** и блок-схемы на рис. 5.8*б*.

Программные реализации алгоритмов ввода элементов одномерного (рис. 5.10*а*) и двумерного (рис. 5.10*б*) массивов приводятся на рис. 5.20. Развитие этих программ связано с вычислением сумм положительных **Sp** и отрицательных **Sn** элементов одномерного (рис. 5.21*а*) и двумерного (рис. 5.21*б*) массивов на основании соответствующих блок-схем алгоритмов на рис. 5.12.

а

```
Sub Fig5_8a
Dim Sz As Single
Dim S As Single
Dim i As Integer
Sz = InputBox("Sz=")
S = 0
i = 0
While S < Sz
  i = i + 1
  S = S + 1/i
Wend
MsgBox "i=" & i
End Sub
```

б

```
Sub Fig5_8b
Dim Sz As Single
Dim S As Single
Dim i As Integer
Sz = InputBox("Sz=")
S = 0
i = 0
Do
  i = i + 1
  S = S + 1/i
Loop Until S > Sz
MsgBox "i=" & i
End Sub
```

Рис. 5.19

а

```
Sub Fig5_10a
Dim a(5) As Single
Dim i As Integer
For i = 1 To 5
 a(i) = InputBox("a=")
Next i
End Sub
```

б

```
Sub Fig5_10b
Dim a(5,10) As Single
Dim i As Integer
Dim j As Integer
For i = 1 To 5
 For j = 1 To 10
  a(i,j) = InputBox("a=")
Next j
Next i
End Sub
```

Рис. 5.20

а

```
Sub Fig5_12a
Dim a(5) As Single
Dim i As Integer
Dim Sp As Single
Dim Sn As Single
Sp = 0
Sn = 0
For i = 1 To 5
 a(i) = InputBox("a=")
  If a(i) > 0 Then
  Sp = Sp+a(i)
  Else
  Sn = Sn+a(i)
  End If
Next i
MsgBox "Sp=" & Sp
MsgBox "Sn=" & Sn
End Sub
```

б

```
Sub Fig5_12b
Dim a(5,10) As Single
Dim i As Integer
Dim j As Integer
Dim Sp As Single
Dim Sn As Single
Sp = 0
Sn = 0
For i = 1 To 5
 For j = 1 To 10
  a(i,j) = InputBox("a=")
   If a(i,j)>0 Then
   Sp = Sp+a(i,j)
   Else
   Sn = Sn+a(i,j)
   End If
 Next j
Next i
MsgBox "Sp=" & Sp
MsgBox "Sn=" & Sn
End Sub
```

Рис. 5.21

В заключительной части материала о циклах рассмотрим программную реализацию сортировки одномерного массива a_1, a_2, ..., a_5 методом «пузырька» (см. рис. 5.22) на основании ранее созданных программы на псевдокоде (см. рис. 5.14*а*) и блок-схемы (рис. 5.14*б*).

Помимо циклов с параметрами **For**... **Next**, осуществляющих ввод данных, перебор элементов массива и вывод

```
Sub Fig5_14
Dim a(5) As Integer
Dim Ind As Integer
Dim b As Integer
Dim i As Integer
For i = 1 To 5
 a(i) = InputBox("a=")
Next i
Do
 Ind = 1
 For i = 1 To 4
  If a(i)>a(i+1) Then
  b = a(i)
  a(i) = a(i+1)
  a(i+1) = b
  Ind = 0
  End If
 Next i
Loop While Ind=0
For i=1 To 5
 MsgBox "a(i)=" & a(i)
Next i
End Sub
```

Рис. 5.22

данных, программа содержит цикл с постусловием **Do... Loop While Ind** = 0, в теле которого и выполняется сортировка методом «пузырька» массива a_1, a_2, ..., a_5 по возрастанию.

Во всех программах на рис. 5.20–5.22 используется цикл **For... Next** потому, что известно количество вводимых и выводимых элементов одномерных и двумерных массивов. Метод «пузырька» является самым простым, но не самым эффективным.

Гораздо большим быстродействием обладает сортировка методами включения, выбора, разделения и др.

5.3.5. ОПЕРАЦИИ С СИМВОЛЬНЫМИ ПЕРЕМЕННЫМИ

Рассмотрим некоторые широко используемые операции с символьными данными (строками).

Операция сцепления (*конкатенации*) «+» служит для соединения нескольких строк в одну строку. Например, конкатенация строк «Папа» + « ел» + « суп» дает строку «Папа ел суп».

Операции отношения (=, <, >, <=, >=, <>) производят сравнение двух строк, имеющее в результате логическое значение *Истина* или *Ложь*.

Две строки сравниваются слева направо до первого несовпадающего символа. Строка считается большей, если в ней первый несовпадающий символ имеет больший номер в таблице символьной кодировки. Если строки имеют разную длину, но в общей части символы совпадают, считается, что более короткая строка меньше, чем более длинная.

Строки равны, если они совпадают по длине и содержат одни и те же символы.

Ряд операций с символьными данными выполняют функции, часть из которых приведена в табл. 5.1.

Таблица 5.1

Функция	Выполняемая операция
`Mid(x,i,k)`	Вырезает (заменяет) в символьной переменной `x` `k` символов, начиная с `i`-го
`Str(x)`	Переводит числовое значение переменной `x` в символьное значение
`Val(x)`	Переводит символьное значение переменной `x` в числовое значение
`Left(x,n)`	Возвращает `n` символов, стоящих с левого края символьной переменной `x`
`Right(x,n)`	Возвращает `n` символов, стоящих с правого края символьной переменной `x`
`LTrim(x)`	Устраняет начальные пробелы из строки `x`

На рис. 5.23 приводится программа, содержащая операции с символьными данными, в которой из целого положительного двузначного числа **ab** получается двузначное число **ba** путем перестановки первой **a** и второй **b** цифры.

```
Sub Symbols
Dim a, ab, b, ba As Integer
Dim s As String
'Ввод числа ab
ab = InputBox("Ввод ab")
'Перевод числа ab в строку s
s = Str(ab)
'Удаление пробелов
s = LTrim(s)
`Перестановка символов a,b
a = Left(s, 1)
b = Right(s, 1)
Mid(s, 2, 1) = a
Mid(s, 1, 1) = b
'Перевод строки s в число ba
ba = Val(s)
MsgBox "ba =" & ba
End Sub
```

Рис. 5.23

Программа имеет линейную структуру и состоит из операций перевода числа в строку, удаления пробелов, перестановки символов и перевода строки в число, выполняемых с помощью функций из табл. 5.1. Ввиду отсутствия блок-схемы для лучшего понимания программа снабжена комментариями — поясняющими надписями, начинающимися с апострофа (').

§ 5.4. ОСНОВНЫЕ ПОНЯТИЯ ОБЪЕКТНО-ОРИЕНТИРОВАННОГО ВИЗУАЛЬНОГО ПРОГРАММИРОВАНИЯ

Программы на языках объектно-ориентированного программирования Visual Basic и Visual Basic for Applications (VBA) строятся из объектов подобно тому, как из различных деталей создаются аппараты и установки. Программные модули объектов в большом количестве входят в системы программирования на языках Visual Basic и VBA.

Системы объектно-ориентированного программирования дают возможность визуализировать процесс создания графического интерфейса разрабатываемого приложения, что дает возможность создавать объекты и задавать значения их свойств с помощью диалоговых окон.

5.4.1. КЛАССЫ ОБЪЕКТОВ, ЭКЗЕМПЛЯРЫ КЛАССА И СЕМЕЙСТВА ОБЪЕКТОВ

Основной единицей в объектно-ориентированном программировании является программный объект, который объединяет в себе как описывающие его данные (свойства), так и средства обработки этих данных (методы).

Если говорить образно, объекты — это «существительные», свойства объекта — «прилагательные», а методы объекта — «глаголы». Программные объекты обладают *свойствами*, используют *методы* и реагируют на *события*.

Классы объектов являются «шаблонами», определяющими наборы свойств, методов и событий. По этим шаблонам создаются объекты.

Каждый из классов обладает специфическим набором свойств, методов и событий. Например, в приложении Word существует класс объектов «документ» (**Document**), который обладает:

- свойствами: имя (**Name**), полное имя (**FullName**) и т. д.;
- методами: открыть документ (**Open**), напечатать документ (**Printout**), сохранить документ (**Save**) и т. д.;
- событиями: создание документа (**Document_New()**), закрытие документа (**Document_Close()**) и т. д.

Экземпляр класса — это объект, созданный по «шаблону» класса объектов, который *наследует* весь набор свойств, методов и событий данного класса. Каждый экземпляр класса имеет уникальное для данного класса имя, которое указывается в скобках после названия класса, например:

```
Document("Отчёт.doc")
```

Различные экземпляры класса обладают одинаковым набором свойств, однако значения свойств у них могут отличаться. Так, в приложении Word могут быть открыты несколько документов, экземпляров класса **Document**, которые имеют различные имена, хранятся в различных каталогах и т. д.

Семейство объектов представляет собой объект, содержащий несколько объектов — экземпляров одного класса. Например, все открытые в текущий момент в приложении Word документы образуют семейство, которое обозначается следующим образом:

```
Documents()
```

Обращение к объекту, входящему в семейство, производится по его имени или индексу. Например, обращение к документу производится по его имени:

```
Documents("Отчёт.doc")
```

Все символы, входящие в выделенный фрагмент документа (объект **Selection**), входят в семейство **Characters()**. Обращение к символу производится по его индексу, например: **Characters(7)**.

5.4.2. ОБЪЕКТЫ: СВОЙСТВА, МЕТОДЫ, СОБЫТИЯ

Свойства объектов (**Properties**). Каждый объект обладает определенным набором свойств, первоначальные значения которых можно установить с использованием диалогового окна системы программирования.

Значения свойств объектов можно изменять в программном коде. Для присвоения свойству объекта нового значения в левой части строки программного кода необходимо указать имя объекта и затем название свойства, разделив их между собой точкой. В правой части строки (после знака равенства) необходимо записать конкретное значение свойства:

```
Объект.Свойство = Значение Свойства
```

Например, установим в выделенном фрагменте текста (объект **Selection**) для первого символа (объект **Characters(1)**) начертание *полужирный* (свойство **Bold**). Свойство **Bold** может быть установлено (значение **True** свойства) или не установлено (значение **False** свойства). В данном случае свойству **Bold** присваиваем значение **True**:

```
Selection.Characters(1).Bold = True
```

Методы объектов (**Methods**). Для того чтобы объект выполнил какую-либо операцию, необходимо применить метод. Многие методы имеют аргументы, которые позволяют задать параметры выполняемых действий. Для присваивания аргументам конкретных значений используется знак :=. Чтобы определить, для какого объекта вызывается метод, перед именем метода указывается имя объекта, отделенное точкой:

```
Объект.Метод арг1 := значение
```

Так, сохранение на диске открытого в приложении Word документа реализуется методом **Save** без аргументов:

```
Documents("Отчёт.doc").Save
```

Операция открытия в приложении Word документа **Отчёт.doc** должна содержать не только название метода

Open, но и указание пути к открываемому файлу (аргументу **FileName** метода **Open** необходимо присвоить конкретное значение):

```
Documents(1).Open
FileName :="C:\Документы\Отчёт.doc"
```

События (**Events**) представляют собой действия, на которые реагирует объект. Событие может создаваться пользователем (например, щелчок мышью или нажатие клавиши) или возникать при воздействии других программных объектов. В качестве реакции на события вызывается определенная процедура, которая может изменять значения свойств объекта, вызывать его методы и т. д.

Например, объект **Document** (Документ) реагирует на события **Open** (Открытие), **New** (Создание) и **Close** (Закрытие), а объект **Selection** (Выделенный фрагмент документа) реагирует на события **Cut** (Вырезание), **Copy** (Копирование), **Paste** (Вставка), **Delete** (Удаление) и т. д.