Общая характеристика этапов решения задачи на персональном компьютере

Человечество живет в мире алгоритмов и программ. Например, все без исключения физиологические процессы — это огромная, тщательно отлаженная и сложно устроенная «библиотека» программ поведения организма. Основы алгоритмизации и программирования фундаментальны и имеют общий характер, приближающий их к основным законам математики, речи или письма.

В процессе решения задачи с применением компьютера пользователь самостоятельно или с помощью специалистов проходит ряд этапов, которые показаны на рис. 1.

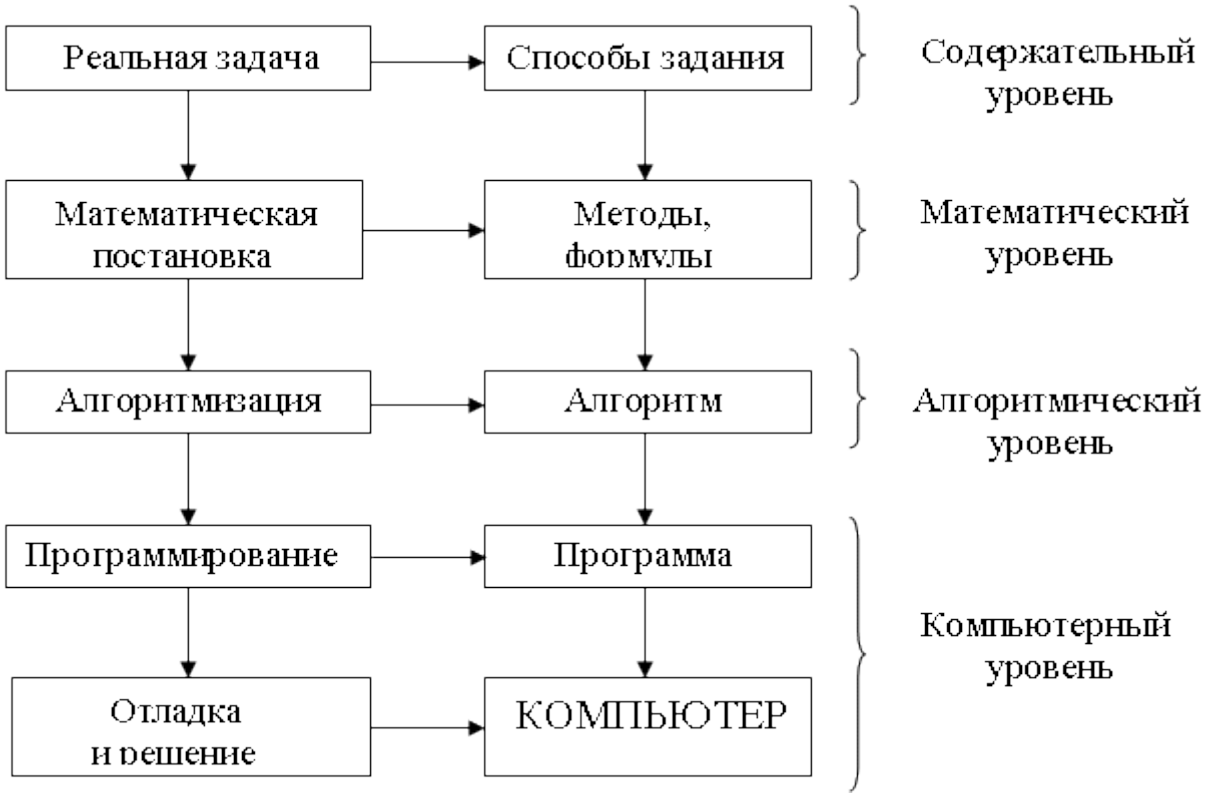


Рис. 1. Этапы решения задачи на ПК.

На всех этапах подготовки задачи к решению и при ее решении на компьютере необходимо учитывать особенности работы вычислительной техники:

- огромное быстродействие компьютера и абсолютная исполнительность;
- ограничения на объемы памяти, которые могут отводиться для записи чисел, переменных и т.п.;
- отсутствие у компьютера интуиции и чувства «здравого смысла»;
- способность компьютера решать только ту задачу, программу решения которой ему подготовил человек.

Рассмотрим последовательность прохождения этапов (рис. 1) на примере решения простой задачи.

На первом этапе формулируются условия задачи (концептуальная модель), например, в словесной форме: функция f(x) должна получить значение, равное единице, если переменная x больше нуля, и ноль, если переменная x принимает другие значения.

Для уяснения «содержания задачи» можно представить ее в графическом виде — отложить на числовой оси точку 0 и отметить значения переменной x, при которых функция f(x) принимает 0 или 1 (рис. 2).

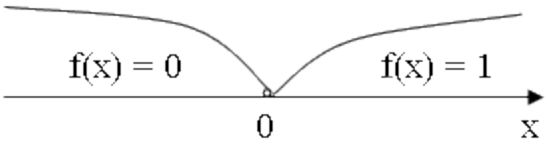


Рис. 2. Графическая интерпретация условий задачи.

На втором этапе производится математическая постановка задачи (математическая модель):

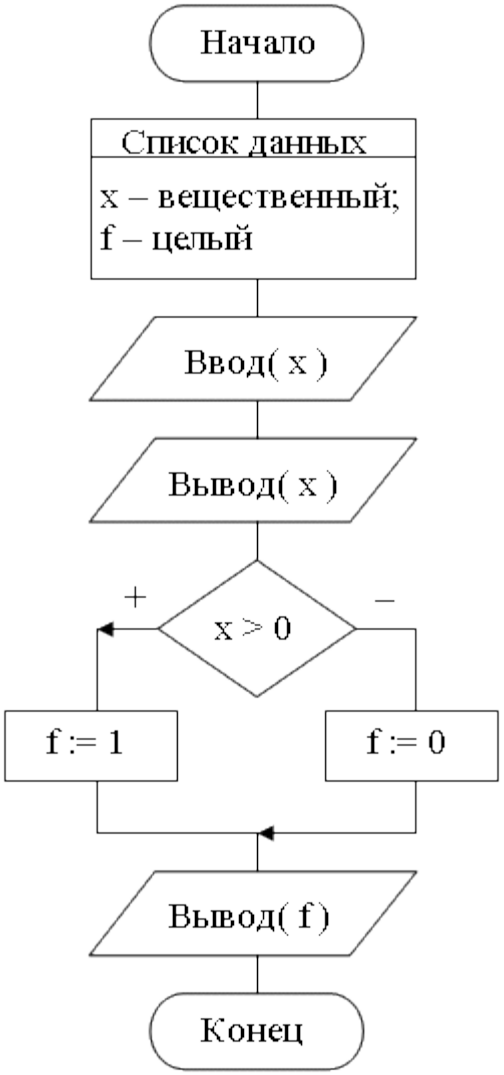
- определяются исходные (вводимые) данные и их типы. В нашем случае к исходным данным относится переменная x, которая может принимать целые и вещественные (содержащие дробную часть) значения. В качестве типа для переменной x выбираем вещественный, поскольку данный тип включает в себя и целые значения тоже;
- решение задачи описывается в виде аналитических зависимостей. Для нашей задачи

$$f(x) = \begin{cases} 1, & \text{при } x > 0; \\ 0, & \text{при } x \le 0; \end{cases}$$

- определяются конечные (выводимые) данные и их типы. В нашем случае конечными данными (результатом решения) является значение функции f(x) целого типа.

На третьем этапе осуществляется разработка алгоритма. Алгоритмизация выступает как связующее звено между «домашними» этапами решения задачи и непосредственно общением человека с компьютером. Алгоритм решения нашей задачи показан на рис. 3.

На четвертом этапе решения задачи алгоритм переводится в программу, записанную на языке высокого уровня. Ниже приводятся программы на языках программирования Turbo Pascal, C++ и QBasic, которые реализуют данный (рис. 3) алгоритм.



Pascal:

```
Program zadacha;
        Var x:real; f:integer;
Begin
        Read(x); WriteLn('x=',x);
        If x>0 Then f:=1
                Else f:=0;
        WriteLn('f=',f);
End.
```

C++:

```
#include <iostream.h>
void main(void)
{
        float x; int f; cin>>x;
        cout<<"x="<<x<<endl;
        f=(x>0)?1:0;
        cout<<"f="<<f<<endl;
}
```

QBasic:

```
REM zadacha
DEFSNG X: DEFINT F
INPUT X: PRINT "X=",X
IF X>0 THEN F=1 ELSE F=0
PRINT "F=",F
END
```

Рис. 3. Алгоритм решения задачи.

На пятом этапе программа вводится в память компьютера, осуществляется ее отладка и решение.

В процентном отношении время, затрачиваемое пользователем на выполнение каждого из этапов при решении задачи с помощью компьютера (на выполнение всех этапов — положим 100%), распределяется примерно так, как показано на диаграмме (рис. 4):

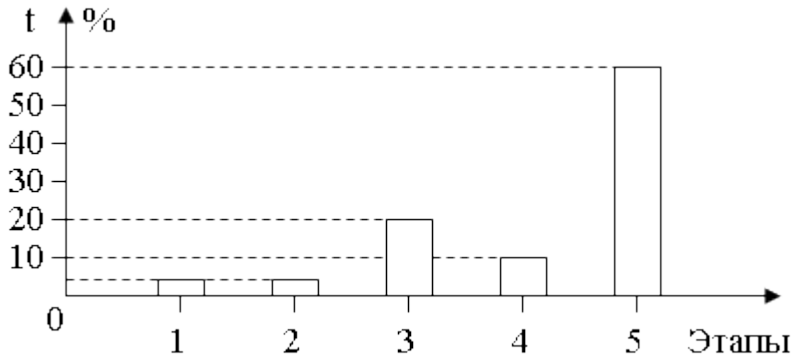


Рис. 4. Распределение времени на решение задачи.

Из диаграммы видно, что больше всего времени, как правило, требуется на выполнение этапа отладки и решения. Это связано с тем, что здесь устраняются ошибки, допущенные пользователем на всех предыдущих этапах решения задачи. Если это ошибки синтаксиса (набор формальных правил написания программы на языке) или семантики (смысловая интерпретация написанного), — они достаточно легко устранимы, особенно, первые. Гораздо хуже наличие алгоритмических ошибок, выявить которые

значительно труднее, а для их устранения требуется осуществлять многократное тестирование программы для различных наборов входных данных, так что иногда проще разработать новый алгоритм и написать новую программу, чем исправить существующую.

Свойства алгоритмов и способы их задания

В соответствии с ГОСТ 19781-74 «Машины вычислительные. Программное обеспечение. Термины и определения», **алгоритм** — это точное предписание, определяющее вычислительный процесс, ведущий от варьируемых начальных данных к искомому результату. Иными словами, алгоритм — это совокупность предписаний, команд, выполнение которых исполнителем приводит к решению данной задачи.

Слово «алгоритм» происходит, как предполагают, от имени среднеазиатского (узбекского) математика IX века Аль Хорезми (Абу Абдалла Мухаммед ибн Муса ал Хорезми ал Меджуси) — «Algorithmi» в латинской транскрипции, впервые сформулировавшего правила (процедуру) выполнения четырех арифметических действий в десятичной системе счисления над простыми числами.

Пока вычисления были несложными, особой необходимости в алгоритмах не было. Когда появилась потребность в решении задач, не имеющих явно выраженного формулами аналитического решения и требующих применения многократных пошаговых процедур, тогда и появилась теория алгоритмов.

Качество алгоритма определяется его свойствами (характеристиками). К основным свойствам алгоритма относятся:

- **Массовость.** Предполагается, что алгоритм может быть пригоден для решения всех задач данного типа. Например, алгоритм для решения системы линейных алгебраических уравнений должен быть применим к системе, состоящей из произвольного числа уравнений.
- **Результативность.** Это свойство означает, что алгоритм должен приводить к получению результата за конечное число шагов, т.е. быть обязательно конечным.
- **Определенность.** Предписания, входящие в алгоритм, должны быть точными и понятными исполнителю. Эта характеристика обеспечивает однозначность результата вычислительного процесса при заданных исходных данных.
- **Дискретность.** Это свойство означает, что описываемый алгоритмом процесс и сам алгоритм могут быть разбиты на отдельные элементарные шаги, возможность выполнения которых на компьютере интерпретируется однозначно. Иными словами, алгоритм должен быть максимально формализован.

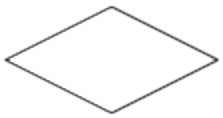
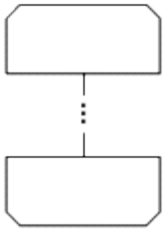
Можно сказать, что алгоритм — это задание, которое нужно исполнить. Утверждения и вопросы могут быть использованы в алгоритмах только как подчиненные предложения в составе предписания.

Наиболее простой и универсальной формой представления алгоритма является словесное описание, которое содержит записанную на естественном языке, точнее, на языке предписаний, четкую пошаговую последовательность таких предписаний. Выполнение алгоритма предполагается в естественной последовательности, то есть в порядке записи. При необходимости изменить порядок выполнения предписаний в явной форме указывается, какое предписание надлежит выполнить следующим.

Наглядностью обладает форма записи — **схема алгоритма**, **графическая схема алгоритма** (ГСА) или **структурная схема**. Такая схема представляет собой графическое представление последовательности команд с использованием специально предусмотренных графических обозначений и представляющая собой документ, который точно описывает порядок выполнения алгоритма. Каждое предписание имеет определенный вид в форме геометрических фигур, а последовательность выполнения операций отображается с помощью линий, соединяющих эти фигуры. Условные обозначения, применяемые при составлении схем алгоритмов, и правила выполнения определены в ГОСТ 19.701-90 «Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения».

В табл. 1 приведены наименования, обозначения и определения отображаемых функций для символов, используемых при описании программ и алгоритмов.

Наименование	Обозначение	Функции	Предписания

			естественного языка записи алгоритма
Данные		Символ отображает данные, носитель данных не определен. Используется для обозначения операций ввода данных к память компьютера или вывода данных на устройство вывода	Ввести данные (следует описание типов данных) Вывести данные (в представлении их соответствующим типом)
Линейный процесс		Символ отображает функцию обработки данных любого вида (выполнение определенной операции или группы операций, приводящее к изменению значения, формы или размещения информации). Используется для обозначения операций присваивания в языках программирования	Выполнить команду вычисления, и сохранить результат для данного, указанного слева от знака операции
Предопределенный процесс		Символ отображает предопределенный процесс, состоящий из одной или нескольких операций или шагов, которые определены в другом месте (в подпрограмме, модуле)	Выполнить группу команд, объединенную в блок, который открывается командой начало и закрывается командой конец
Подготовка		Символ отображает модификацию команды или группы команд с целью воздействия на некоторую последующую функцию (установка переключателя, модификация индексного регистра или инициализация программы). Может быть использован для обозначения заголовка цикла	Команда поготовки группы данных к дальнейшему их использованию в командах алгоритма
Проверка условия/ принятие решения		Символ отображает решение или функцию переключательного типа, имеющую один вход и ряд альтернативных выходов, один и только один из которых может быть активизирован после вычисления условий, определенных внутри этого символа. Соответствующие результаты вычисления могут быть записаны по соседству с линиями, отображающими эти пути. Используется для обозначения команды языка программирования — оператора условного перехода или оператора выбора варианта из возможных.	Команда вида: если условие или несколько условий выполняются, то выполняется одна группа команд, а если условие или группа условий не выполняется, то осуществляется переход на другую группу команд
Граница цикла		Символ, состоящий из двух частей, отображает начало и конец цикла. Обе части символа имеют один и тот же идентификатор. Условия для инициализации, приращения, завершения и т.д. помещаются	Это предписание позволяет оформить группу команд, которая повторяется многократно, то есть реализуется на несколько шагов. Повтори выполнение

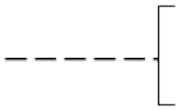
		внутри символа в начале или в конце в зависимости от типа цикла	группы команд (с некоторым шагом), пока не выполнится некоторое условие
Соединитель		Символ отображает выход в часть схемы и вход из другой части этой схемы и используется для обрыва линии и продолжения ее в другом месте. Соответствующие символы-соединители должны содержать одно и то же уникальное обозначение	Используется только для структурных схем
Терминатор		Символ отображает выход во внешнюю среду и вход из внешней среды. Используется для обозначения начала или окончания алгоритма	Предписание, показывающее вход в алгоритм и на какую команду он указывает, или команду, которая прекратит выполнение данного алгоритма
Линия		Символ отображает поток данных или управления. Направления справа налево и снизу вверх обозначаются стрелками. Используется для соединения символов в алгоритме	В предписании соответствует номерам команд, указывающим на очередность их выполнения
Комментарий		Символ используется для добавления описательных комментариев или пояснительных записей с целью объяснений или примечаний. Пунктирные линии в символе комментария связаны с соответствующим символом или могут обводить группу символов. Текст комментариев или примечаний должен быть помещен около ограничивающей фигуры	Предписание, которое поясняет команду (комментарий к команде)

Таблица 1. Описание символов, используемых в ГСА.

Символы могут быть вычерчены в любой ориентации, но предпочтительной является горизонтальная ориентация. Внутрь символа помещают обозначения или описания операций. Направления линий связи слева направо и сверху вниз считаются стандартными, и линии связи изображаются без стрелок, в противоположном случае — со стрелками. В операционные блоки (процесс, ввод-вывод и подпрограмма) входит и из них исходит только одна линия связи. В блок проверки условий (ветвление) входит одна, а выходит не менее двух линий, около которых записываются результаты проверки условий. Из начальной вершины исходит одна линия связи, в конечную вершину также входит одна линия связи. Линии могут соединяться одна с другой, но не могут разветвляться. Символы ГСА могут быть отмечены идентификаторами или порядковыми номерами. Идентификатор представляет собой букву или букву с цифрой и должен располагаться слева над символом.

Внутри соединителей ставятся номера (координаты) блоков, к которым или от которых идут линии связи. Вместо номера (координат) может быть поставлен некоторый (один и тот же в обоих соединениях) идентификатор.

Недостатком графических схем алгоритмов является громоздкость. Для решения специальных задач, например проектирования вычислительных устройств, применяются другие способы задания алгоритмов, такие, как логические (операторные) и матричные схемы. Их достоинствами являются компактность и простота дальнейшей формализации, а недостатком — малая наглядность.

Выполнение алгоритма исполнителем

Практическая реализация всех предусмотренных действий по получению результата для конкретных значений аргументов осуществляется в процессе исполнения алгоритма.

При исполнении алгоритма на компьютере его необходимо записать на языке команд языка программирования, который интерпретируется компьютером в машинные команды. Каждая команда — оператор языка программирования — содержит код операции и данные, над которыми эта операция производится. При этом результаты выполнения команд хранятся в специально отведенных для данных ячейках памяти компьютера. Если какая-то конкретная команда выполняется многократно, то результат выполнения команды может изменяться, при этом в ячейке памяти, отведенной для хранения результата именно этой команды всегда остается последний полученный результат ее выполнения компьютером .

Рассмотрим пример алгоритма, записанного как с помощью предписаний, так и с помощью структурной схемы (рис. 5).

1. Начало;
2. Список типов данных:
   a, b, c — **целый** ;
   x — **вещественный**;
3. **Ввод**(a, b, c);
4. **Вывод**(a, b, c);
5. **Если** (a>b) **То**
6. x**:=**2 a — 3 c;
   Иначе
7. x:=2 b + 3 c;
   Конец-**Если** ;
8. x**:=** x + c;
9. **Вывод**(x);
10. **Конец**.

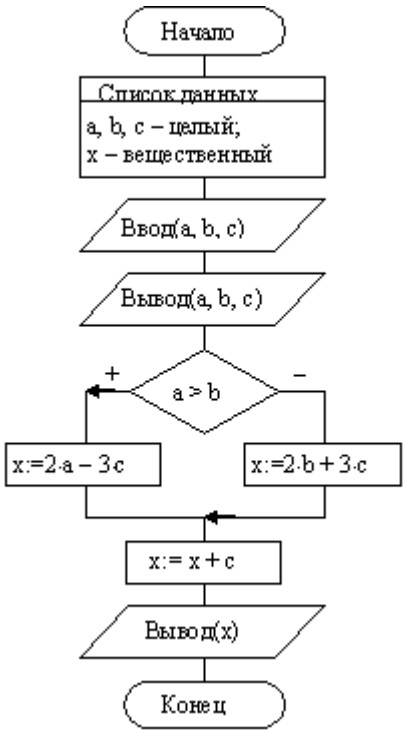


Рис. 5. Пример алгоритма.

На языке предписаний использованы ключевые слова, которые соответствуют графическим формам: это такие слова, как ВВЕСТИ. ВЫВЕСТИ, ЕСЛИ, ТО, ИНАЧЕ, НАЧАЛО, КОНЕЦ, ТИП, «:=» или ВЫЧИСЛИТЬ (выполнить команду с данными, указанную справа от знака «:=» и поместить результат в данное, имя которого указано слева от знака «:=»).

Для общего представления, как будет выполняться этот алгоритм компьютером, составим таблицу значений, заголовок которой имеет следующий вид:

Память	Команда	Вывод

Заполнение таблицы производится последовательно, по мере имитации выполнения алгоритма в компьютере:

- При появлении в записи алгоритма данных, над которым затем будут проводится действия, им даются имена, а при выполнении алгоритма на компьютере — выделяются ячейки памяти. Для этого мы предусмотрели столбец «Память», имитируя выделение в компьютере соответствующей ячейки памяти с таким же именем. Назовем данные, которым дано имя и которые могут изменять свое значение при выполнении команд алгоритма ПЕРЕМЕННЫМИ.
- Первое значение, которое получит переменная А, записывается в столбец «результат» в выделенную для нее строку. Например, если переменная А получит при выполнении команды «Ввести» значение 5, то в строке с дааным, имеющим имя А в столбце «результат» запишем А=5.
- Каждое новое значение переменной записывается в отдельной строке. Например, если переменная А получит новое значение 7, то в строке А запись будет иметь вид: А=7. Следует помнить, что в памяти компьютера новое значение переменной будет записано в туже ячейку памяти с именем А, и предыдущее значение, хранимое в этой ячейке, будет заменено новым значением.
- Столбцы «команда» и «результат» заполняются по мере появления соответствующих предписаний в алгоритме.

Если использовать предложенную форму, то таблица значений для рассматриваемого примера (рис. 5), будет иметь вид, показанный в табл. 2.

Ячейки памяти, участвующие в команде	Команда	Результат
A B C	Ввести 5 Ввести 4 Ввести 3	A=5 B=4 C=3
A, B	Проверить 5>4?	«истина», «верно»
X, A, C	ВЫЧИСЛИТЬ 2*A- 3*C	X=7
X, X, C	ВЫЧИСЛИТЬ X + C X=8	
X	Вывести	X=8

Таблица 2. Пошаговое выполнение алгоритма.

Для поиска ошибок в алгоритмах, а также для анализа алгоритмов и программ, можно использовать специально предусмотренные на компьютере процедуры вывода на экран монитора или бумагу на принтере пошагового выполнения алгоритма с состоянием на каждом шаге ячеек памяти. Анализ такого представления выполнения алгоритма компьютером помогает при поиске ошибок.

Принципы структурной алгоритмизации

Английский математик **Чарлз Бэббидж** (Charles Babbage, 1792-1871) выдвинул идею создания программно-управляемой счетной машины, имеющей арифметическое устройство, устройство управления, ввода и печати.

Первая спроектированная Бэббиджем машина, **Разностная машина**, работала на паровом двигателе. Она высчитывала таблицы логарифмов методом постоянной дифференциации и заносила результаты на металлическую пластину. Работающая модель, которую он создал в 1822 г., была шестицифровым калькулятором, способным производить вычисления и печатать цифровые таблицы.

Одновременно с английским ученым работала леди **Ада Лавлейс** (Ada Byron, Countess of Lovelace, 1815-1852 гг.), дочь известного английского поэта лорда Байрона. Она разработала первые программы для машины, заложила многие идеи и ввела ряд понятий и терминов, сохранившихся до настоящего времени. Именно ее многие считают первым программистом в мире. В ее честь один из процедурных языков программирования, разработанный в США в 1969 г. был так и назван — АДА.

Свое понятие программистской деятельности ввели инженеры, обслуживающие цифровой компьютер «Марк-1» в Гарвардском университете (1944 г.). При написании программ для вычисления

артиллерийских баллистических таблиц впервые стали применять многократно используемые последовательности — подпрограммы. Устранение неисправности, вызванной попаданием мотылька (bug — насекомое) в электрические цепи машины, привело к появлению понятия «отладка» (debugging), которое используется сейчас для обозначения поиска неисправностей в компьютере и его программном обеспечении. Впервые была предложена программа- компоновщик как вспомогательное средство для создания других программ из нескольких различных подпрограмм ... и т.д.

Появление языков программирования высокого уровня дало мощный импульс развитию программного обеспечения. Однако первоначально не было выработано подходов (методологий) к программированию, использование которых позволило бы уменьшить вероятность пропуска ошибок в программах, упрощало бы их понимание, облегчило бы модификацию программ и их сопровождение.

Значительный вклад в теорию программирования внес научный сотрудник фирмы «Барроуз» (Burroughs), голландский ученый Эдсгер Дийкстра. В 1968 г. в работе под названием «Заметки по структурному программированию» он доказал, что большинство программ неоправданно сложны из-за отсутствия в них четкой математической структуры. По мнению Дийкстры, использование трех типов управляющих структур — **простой последовательности, альтернативы** и **повторения** — позволило бы программистам обходиться без операторов безусловного перехода (goto) и тем самым преодолеть сложность и запутанность программ, добиться простоты их модификации. Проверку программ Дийкстра предложил производить математическими методами, а не просто тестированием, которое, по его мнению, может показать лишь наличие ошибок, а не их отсутствие.

На сегодняшний день **структурное программирование** является одной из самых популярных технологий программирования. Эта технология представляет собой процесс пошагового разбиения алгоритма на все более мелкие части с целью получить такие элементы, для которых можно легко написать конкретные предписания (высказывания). Идея структурного программирования является формализованным выражением принципов рационального мышления, сформулированных еще Рене Декартом более 350 лет назад. Под конкретными предписаниями обычно понимают линейные, разветвляющиеся и циклические структуры. В алгоритмах, разработанных таким методом, как правило, меньше ошибок, и верифицировать алгоритм (доказывать его правильность) легче.

В основу структурной алгоритмизации положены следующие требования:

- Программа должна составляться мелкими шагами. Размер шага определяется количеством решений, принимаемых программистом на каждом шаге. Таким образом, сложная задача разбивается на достаточно простые, легко воспринимаемые части.
- Логика программы должна опираться на минимальное число достаточно простых базовых управляющих структур.

По своей сути структурное программирование является воплощением принципов системного подхода к процессу создания и эксплуатации математического обеспечения компьютера.

Другими словами, алгоритм надо писать так, как мы его понимаем и как хотели бы объяснить его другим людям.

Структурная алгоритмизация основывается **на двух принципах**:

- Последовательная детализация алгоритма «сверху вниз».
- Ограниченность базового набора структур для построения алгоритма любого уровня сложности.

Можно перечислить основные свойства и достоинства структурного программирования:

- возможность преодоления барьера сложности программ;
- возможность демонстрации правильности программ на различных этапах решения задач;
- наглядность программ;
- простота модификации (внесения изменений);
- возможность реализации технологии «нисходящего» проектирования программ.

Составление алгоритма с применением структурной алгоритмизации осуществляется следующим образом. На первом шаге решения задачи считаем, что перед нами самый совершенный компьютер,

который уже «все умеет и все знает». Поэтому задача для него записывается на естественном для данной предметной области языке. Например, нам необходимо по результатам испытаний выбрать оптимальные параметры двигательной установки ракеты. Тогда первоначально алгоритм можно записать так:

1.Начало;
2. Список данных: данные: тип; … ; данные: тип;
3. Ввод (исходные данные);
4. Вывод (исходные данные);
5. ОБРАБОТКА РЕЗУЛЬТАТОВ ИСПЫТАНИЙ;
6. ОЦЕНКА НАДЕЖНОСТИ ДВИГАТЕЛЬНОЙ УСТАНОВКИ;
7. ВЫБОР ОПТИМАЛЬНЫХ ПАРАМЕТРОВ;
8. Вывод (результат);
9. ;Конец.

Затем следует определить, какие из использованных нами конструкций «понятны» компьютеру, имеющемуся в нашем распоряжении. Дело в том, что алгоритм может быть реализован в компьютере, если он содержит только элементарные предписания и структуры, входящие в базовый набор (о них речь пойдет ниже). **Элементарными**, т.е. не требующими детализации, мы будем считать следующие предписания или операции: Начало; Конец; Список данных; Ввод; Вывод; вычислительные операции, реализуемые операторами присваивания (:=). Сложные конструкции, которые компьютер не может реализовать сразу, разбиваются на более простые. Например, предписание 5 для рассматриваемого алгоритма может быть записано так:

5.1.Начало 5;
5.2. ...Операции, раскрывающие
смысл обработки результатов
испытаний ... ;
5.N. Конец 5.

Количество шагов при последовательной детализации не ограничивается. Детализация заканчивается, когда конструкции алгоритма будут содержать только элементарные предписания и структуры, входящие в базовый набор.

Обратите внимание на то, что предписания 1 — 4 и 8 — 9, как правило, присутствуют в каждом алгоритме, разрабатываемом методом структурной алгоритмизации.

Базовый набор структур для построения алгоритма

Теория структурного программирования доказывает, что алгоритм любой степени сложности можно построить с помощью основного базового набора структур:

- последовательная (линейная) структура (рис. 6а);
- ветвящаяся структура (рис. 6б);
- циклическая структура (рис. 6в).

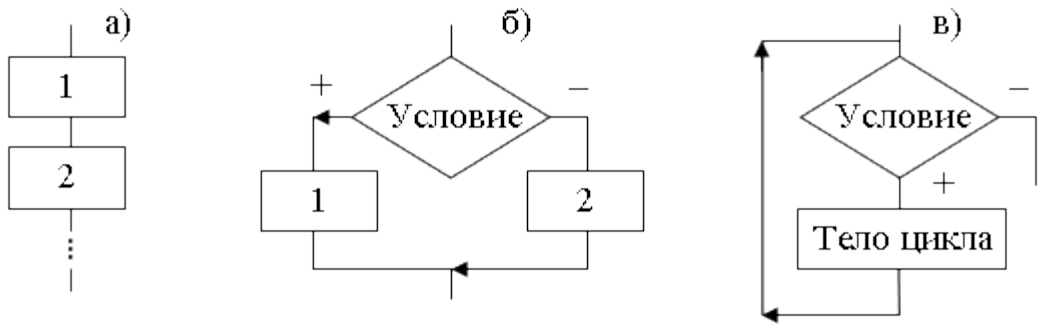
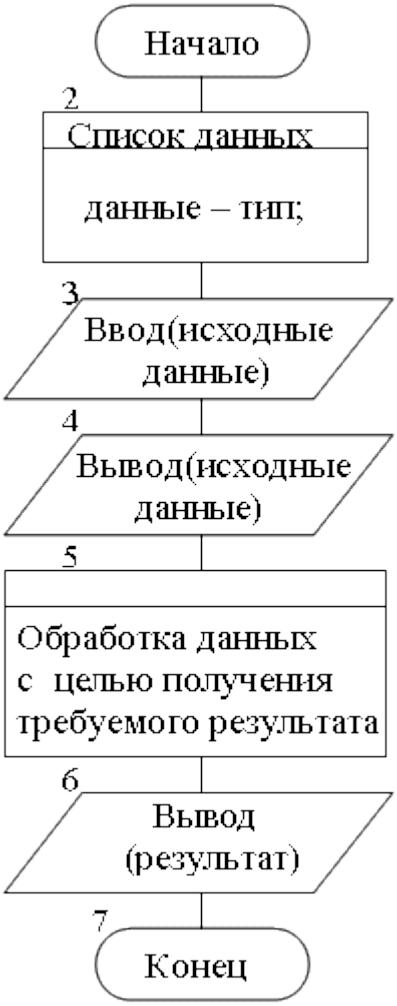


Рис. 6. Базовый набор структур.

Поскольку каждая типовая структура имеет 1 вход и 1 выход, то любой операционный блок может быть, в свою очередь, представлен в виде последовательности любых базовых структур (в общем случае с любой глубиной вложения). Эта возможность и обеспечивает, в конечном счете, построение алгоритмов любой степени сложности только из основных базовых структур.



Линейные и разветвляющиеся структуры

Наиболее простыми для понимания и использования являются линейные структуры. Первоначально с их помощью можно описать любой вычислительный процесс. Предписания, которые обычно содержит такой алгоритм, представлены на рис. 7.

Предписание «Список данных» содержит сведения об именах и типах данных, обрабатываемых в этом алгоритме. Предписание «Ввод(исходные данные)» определяет, какие исходные данные и в каком порядке должны быть введены в компьютер. Предписание «Вывод (исходные данные)» позволяет проконтролировать правильность ввода информации. Предписания 5 и 6 позволяют получить требуемые результаты и выдать их пользователю. Рассматриваемый алгоритм относится к линейным.

Линейной называется алгоритмическая структура (см. рис. 6а), в которой отдельные предписания выполняются в естественном порядке (в порядке записи) независимо от значений исходных данных и промежуточных результатов.

Линейной, например, является последовательность **вычислений** по какой-либо формуле с помощью карманного калькулятора. Более подробно следует рассмотреть запись математических формул в виде специального предписания:

X:=A;

которая читается следующим образом: «Переменной X присвоить значение, равное результату, полученному при вычислении по формуле A». В этом предписании: X — переменная; A — может быть любым, сколь угодно сложной математической формулой. В процессе выполнения этого предписания компьютером A вычисляется, и результат вычисления присваивается содержимому ячейки памяти (помещается в эту ячейку), отведенной для хранения переменной X. При этом переменная X теряет свое предыдущее значение и приобретает новое. Таким образом, символ «:=» употребляется как предписание «присвоить». Как следствие этого, бессмысленное, с точки зрения алгебры, выражение:

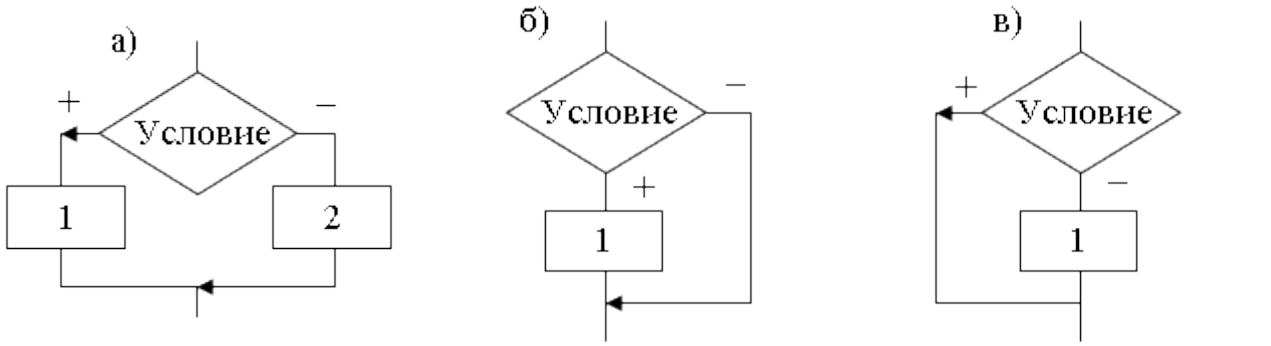
X:= X+5;

является широко распространенной командой в программировании. Такая команда называется оператором присваивания и означает, что к текущему содержимому ячейки с именем X добавляется число 5, после этой операции сложения ячека X теряет свое старое значение и приобретает новое, которое на 5 больше предыдущего. В результате выполнения оператора присваивания всегда обновляется значение ячейки памяти, имя которой указывается в записи оператора слева от знака «:=» после вычисления формулы, которая записана в операторе справа от знака «:=».

Линейные структуры используются на первых этапах детализации задачи. Однако только в редких случаях все предписания такого алгоритма являются элементарными. Так, например, предписание 5 на рис. 7 не является элементарным и требует дальнейшей детализации. Поэтому назначение блока 5 предусматривает сверху свободное место для записи координат блока, в котором будет раскрываться смысл детализируемого участка.

Часто для дальнейшей детализации алгоритма используются ветвящиеся структуры (рис. 8).

Ветвящейся структурой (разветвляющейся) называется алгоритм (фрагмент алгоритма), в котором в зависимости от исходных данных или промежуточных результатов вычисления реализуется по одному из нескольких, заранее предусмотренных (возможных) направлений. Такие направления называются ветвями вычислений.



Если (условие) То
операторы ветви 1;
Иначе
операторы ветви 2;
Конец-Если;

Если (условие) То
операторы ветви;
Конец-Если;

Если (условие) То
Иначе
операторы ветви;
Конец-Если;

Рис. 8. Ветвящаяся структура: а — стандартная схема; б, в — частные случаи ветвления.

Каждая ветвь может быть любой степени сложности, а может вообще не содержать предписаний (рис. 8б — 8в), т.е. быть «вырожденной». Выбор той или иной ветви осуществляется в зависимости от результата проверки условия. В каждом конкретном случае алгоритм реализуется только по одной ветви, а выполнение остальных исключается. В схемах, приведенных на рис. 8, положительный исход проверки условия обозначен знаком «+» (да или true/истина, что соответствует значению в ячейке памяти, хранящей результат операции проверки условия, равном «1»), а отрицательный — знаком «-» (нет или false/ложь, «0»).

При составлении алгоритма в виде псевдокодов линии связи заменяются словами «Идти» или «Перейти» с указанием номера предписания (оператора), которое должно выполняться на следующем шаге алгоритма. Горизонтальная линия, объединяющая ветви «+» и «-», в псевдокодах имеет аналог — «Конец-Если». После фразы «Конец-Если» можно указать номер псевдокода, в котором записано проверяемое условие. Например, после пункта 7 в алгоритме, представленном на рис. 5, указывается: «Конец-Если 5». Использование данной конструкции при записи алгоритма в псевдокодах позволяет легко определить место окончания разветвления (продолжения основного алгоритма).

На практике часто встречаются задачи, когда нужно выбрать не одно из двух, а одно из трех или более предписаний. Такую структуру называют выбором варианта, ее также можно построить из линейных и ветвящихся структур, как показано на рис. 9. В такой структуре сначала вычисляется значение выражения, стоящего в операторе присваивания. В зависимости от значения переменной i затем будет выбран либо 1-й, либо 2-й, либо 3-й, либо 4-й оператор. Число выбираемых операторов в такой структуре не ограничено.

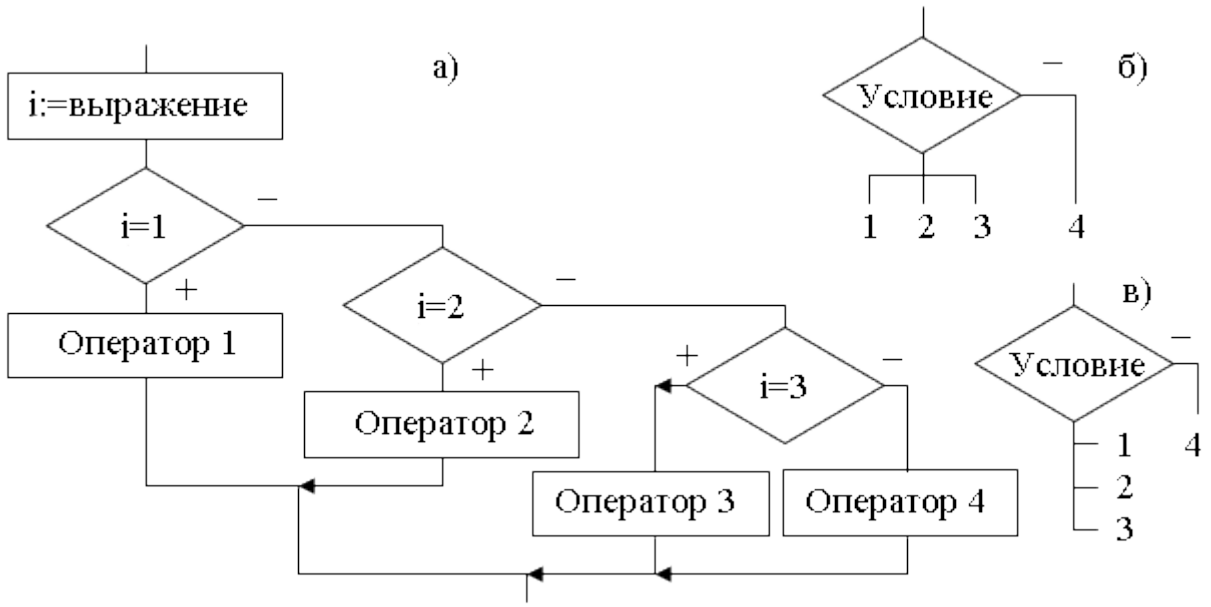


Рис. 9. Выбор варианта: а — структура выбора варианта;
б, в — условное обозначение.

В качестве примера использования ветвлений рассмотрим составление алгоритма для вычисления функции f в зависимости от конкретных значений x, a, b:

$$f=\begin{cases} x+a, & \text{при } x \le 2.5; \\ x-a\cdot b, & \text{при } 2.5 < x < 10; \\ x-b, & \text{при } x \ge 10. \end{cases}$$

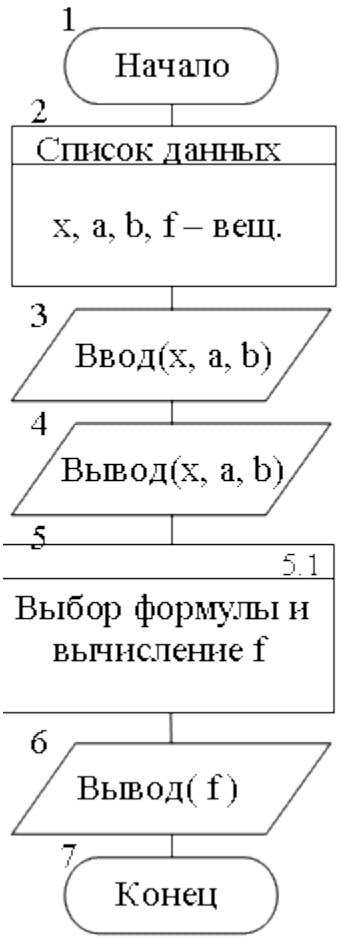


Рис. 10. Первый этап составления алгоритма.

Первое приближение алгоритма будет иметь вид, показанный на рис. 10. Анализ этого алгоритма показывает, что все его блоки, кроме блока 5, являются элементарными. Возможный вариант дальнейшей детализации блока 5 представлен на рис. 11. Все блоки второго этапа детализации являются элементарными, поэтому составление алгоритма практически закончено.

На заключительном этапе производится сборка алгоритма, т.е. содержимое рис. 11 помещается на рис. 10 вместо блока 5.

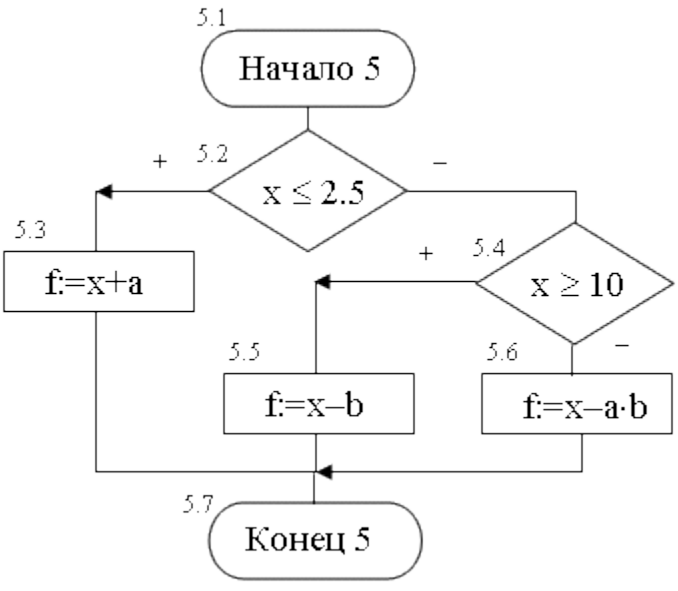


Рис. 11. Второй этап составления алгоритма.

Организация циклических структур алгоритма

Настоящие преимущества компьютера в скорости выполнения команд становятся очевидными лишь при решении тех задач, где возникает необходимость многократного повторения одних и тех же фрагментов алгоритмов.

Циклическими называются структуры алгоритмов, у которых выполнение некоторых операторов (групп операторов) осуществляется многократно с одними и теми же или модифицированными данными.

Циклические алгоритмы находят самое широкое применение в программировании, так как при этом человек составляет программу, описывая в ней циклическую структуру один раз, а компьютер выполняет ее многократно. Циклические структуры часто называют циклами.

В зависимости от способа организации числа повторений циклической структуры различают три типа циклов: цикл с заданным условием продолжения работы, цикл с заданным условием окончания работы и цикл с заданным числом повторений.

Логика работы **цикла с заданным условием продолжения работы (цикл-ПОКА)** описывается схемой, показанной на рис. 10а.

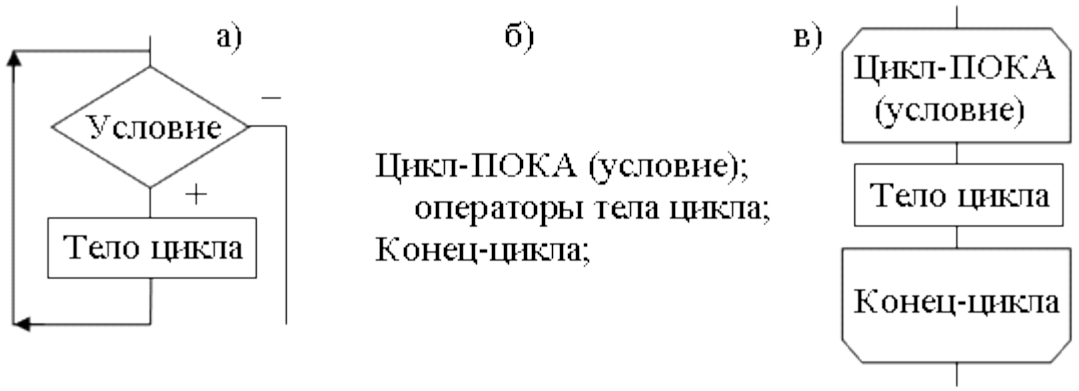


Рис. 12. Структура цикл-ПОКА: а — развернутая схема цикла;
б — запись в псевдокодах; в — компактная схема цикла.

Тело цикла может включать в себя группу операторов любой степени сложности. При выполнении условия продолжения работы выполняется тело цикла, если же условие не выполняется, то работа циклической структуры заканчивается и начинает выполняться следующая структура основного алгоритма.

Структура цикл-ПОКА предусматривает вариант, когда тело цикла не выполняется ни разу. Такое возможно, если условие, стоящее в начале цикла, сразу же не выполняется. Когда на практике возникает необходимость использовать структуру, у которой тело цикла выполняется хотя бы один раз, то в этом случае применяется **цикл с заданным условием окончания работы (цикл-ДО)**.

Структура цикла-ДО приведена на рис. 13. С помощью такой структуры обычно составляют алгоритмы итерационных вычислительных процессов, то есть процессов, в которых для определения последующих значений переменной используется ее предыдущее значение. Итерационный процесс положен, например, в основу метода последовательных приближений.

Выход из конструкции цикл-ДО осуществляется по достижении заданной точности или по какому-либо другому признаку.

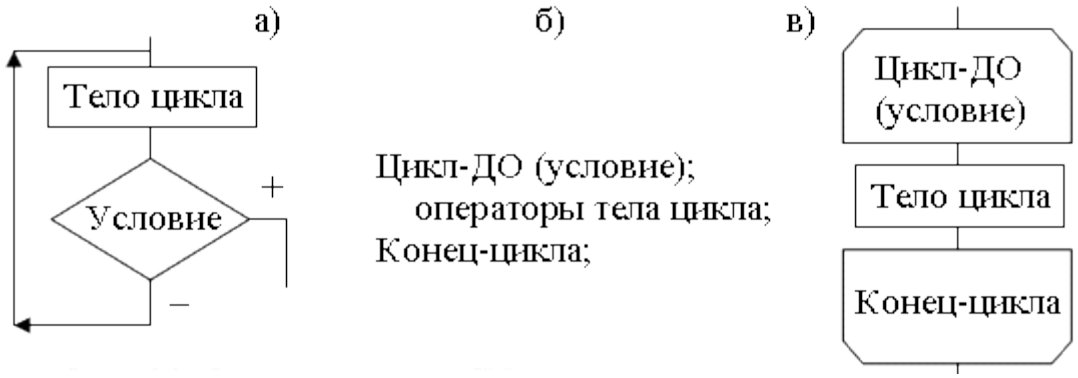


Рис. 13. Структура цикл-ДО:
а — развернутая схема цикла; б — запись в псевдокодах; в — компактная схема цикла.

Если условие проверяется до входа в «тело цикла», то оно называется предусловием, если после — постусловием.

Рассмотренные типы циклических структур имеют один недостаток: при ошибочном задании исходных данных может произойти зацикливание, т.е. возникает неприятная ситуация, когда происходит бесконечное повторение операторов, входящих в тело цикла. В этом случае приходится принудительно завершать работу программы, иногда это связано с потерей не сохраненных данных и самой программы.

В практических инженерных задачах обычно известны начальные значения изменяемых величин, закон изменения и конечное число повторений. Переменная, изменение которой организуется в ходе реализации цикла, называется **параметром цикла**, или управляющей переменной. Алгоритм работы **цикла с заданным числом повторений (иногда его называют циклом с параметром)** приведен на рис. 14.

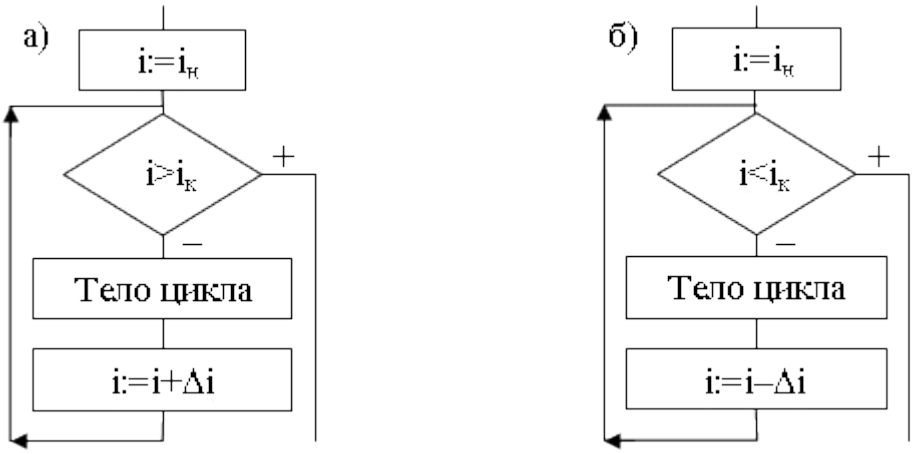


Рис. 14. Развернутая схема цикла с заданным числом повторений:
а — с возрастающим параметром; б — с убывающим параметром.

На рис. 14 обозначено: i — параметр цикла; $i_н$ — начальное значение параметра; $i_к$ — конечное значение параметра; Δi — приращение (шаг). Прочитать этот алгоритм можно следующим образом: «Меняя параметр от начального значения до конечного значения, повторять тело цикла».

Алгоритм, приведенный на рис. 14, принято называть развернутой схемой цикла с заданным числом повторений. Такая схема является удобной для анализа алгоритма и поиска ошибок. Однако при написании алгоритма можно использовать и компактную запись. В псевдокодах она выглядит так:

Цикл по **параметр** от **начальное значение**
до **конечное значение** шаг **приращение**;
операторы тела цикла;
Конец-цикла.

Необходимо особо подчеркнуть, что развернутая и компактная записи после реализации в машине дают один и тот же результат. Компактная запись менее громоздкая за счет того, что в ней не задаются в явном виде связи между отдельными элементами структуры.

Для изображения компактной графической схемы цикла с параметром могут быть использованы символы «Подготовка» или «Граница цикла» (см. табл. 1), как показано на рис. 15.

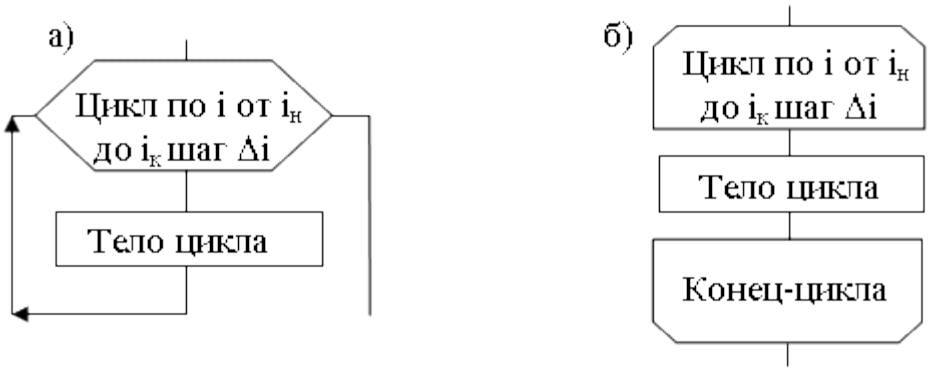


Рис. 15. Компактная запись цикла с параметром:
а — с использованием символа «Подготовка»; б — с использованием символа «Граница цикла».

При работе с технической литературой, а также при отладке алгоритмов и программ необходимо уметь правильно читать циклические операторы. Сложность состоит в том, что при компактной записи, которая чаще всего используется в технической литературе, проверка условия окончания цикла и изменение параметра цикла с заданным числом повторений не указаны явно. В компактной форме могут быть заданы и другие типы циклов, поэтому анализ и проверку исполнения циклических структур рекомендуется проводить по развернутой схеме алгоритма.

В качестве примера рассмотрим анализ циклического алгоритма, записанного в виде псевдокодов:

1. Начало;
2. Список данных: x, f1 — целый;
3. Цикл по x от 1 до 10 шаг 3;
4. f1:=x2;
5. Вывод(x, f1);
6. Конец-цикла 3;
7. Конец.

Результаты анализа рассматриваемого алгоритма приведены в табл. 3.

Память	Условия	Вывод
x=1, 4, 7, 10, 13 f1=1, 16, 49, 100	1>10 нет 4>10 нет 7>10 нет 10>10 нет 13>10 да	1 1 4 16 7 49 10 100

Ячейка памяти	Команда	Результат
X	:=1	1
	1>10	нет
F1	1*1	1
X F1	вывод	1 1
X	:=1+3	4

	4>10	нет
F1	:=4*4	16
X, F1	вывод	4 16
X	:=4+3	7
	7>10	нет
F1	:=7*7	49
X, F1	вывод	7 49
X	:=7+3	10
	10>10	нет
F1	:=10*10	100
X, F1	вывод	10 100
X	:=10+3	13
	13>10	да
	конец	

Таблица 3. Таблицы значений.

Особо отметим, что при x=10 результатом проверки 10>10 является ответ «нет», и тело цикла выполняется последний раз. При x=13 условие окончания цикла выполняется, и управление передается оператору, стоящему в строке 7 алгоритма.

Основы типизации и структуризации данных

Информацию, относящуюся к решаемой задаче, принято подразделять на данные (исходные данные, промежуточные и конечные результаты) и программу (информация, задающая алгоритм решения задачи). В алгоритме данные описываются в предписании: «Список данных:…».

Все данные по своему виду подразделяются на константы и переменные (рис. 16). **Константы** — это данные, которые при выполнении алгоритма (программы) всегда определены и неизменны. Запись константы полностью определяет ее назначение, тип, форму представления и фактическое значение. **Переменные** — это условные обозначения данных, которые в процессе выполнения программы не меняют своего типа, но могут менять свое фактическое значение.

По своему назначению данные делятся на арифметические, символьные и управляющие. Управляющие данные применяются для управления выполнением программы. Значениями арифметических данных являются числа, а символьных — строки символов, заключенные в апострофы, например: 'Конец решения', 'Проверка' и т.п. Арифметические данные по своему типу делятся на вещественные (действительные) и комплексные. В том случае, если в вещественном числе отсутствует дробная часть, тип таких данных определяется как целый. По форме представления данные делятся на данные с фиксированной точкой (например, 78.5) и плавающей точкой (с масштабным множителем, например, $7.85\text{E+}01 = 7.85 \cdot 10^1$).

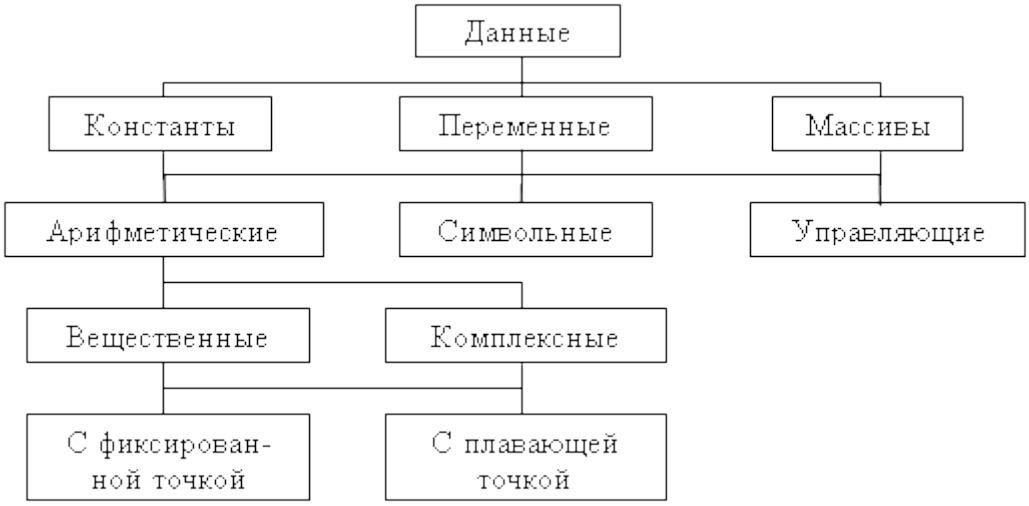


Рис. 16. Классификация данных.

Переменные, применяемые в алгоритмах и программах, могут образовывать упорядоченные структуры. Широкое применение находит упорядочение переменных в виде массивов.

Массивы

Массив — это упорядоченное множество однотипных переменных (элементов массива), объединенных общим именем и отличающихся номерами (индексами). Массивы сходны с такими понятиями в математике, как векторы и матрицы.

Значения индексов указываются в скобках справа от имени массива и однозначно определяют положение переменной в массиве.

Например:

a(1:5) — массив (вектор) из пяти элементов:

a(1), a(2), a(3), a(4), a(5);

b(1:2, 1:3) — массив (матрица) из шести элементов:

b(1,1), b(1,2), b(1,3),

b(2,1), b(2,2), b(2,3).

Каждый массив характеризуется размерностью (числом измерений), границами индексов по каждому измерению и длиной.

Диапазон изменения индексов на каждой позиции определяется парой чисел — минимальным и максимальным значениями индекса, разделенными двоеточием. Если нижняя граница (минимальное значение) равна единице, то ее можно не указывать.

Длина массива равна количеству элементов в массиве.

Размерность массива — это число индексов в списке индексов. Чаще всего на практике используются одно-, двух- и трехмерные массивы, графическая интерпретация которых показана на рис. 17 и 18. Из рисунков видно, что одномерный массив (рис. 17а) можно представить «линией», состоящей из столбцов ячеек памяти. Двухмерный (рис. 17б) массив — это «плоскость», состоящая из строк и столбцов ячеек памяти. Трехмерный (см. рис. 18) массив — это «куб», состоящий из плоскостей, строк и столбцов ячеек памяти. Кстати, одну ячейку памяти можно в общем случае представить одномерным массивом, состоящим из одного столбца, и обращаться к ней как к X(1).

Для работы с массивами, необходимо в специальном предписании «Список данных:…» указать имя и границы изменения индексов массива.

Предписание «Список данных» может включать в себя константы, переменные и массивы. В нем указывается, сколько ячеек памяти следует выделить в машине, чтобы записать все необходимые данные. Например:

…Список данных:

Константы: Pi=3.14159;
Q=1.5678;
Переменные: a, b — целый;
x — вещественный;
c — символьный;
Массивы: y(1:2, 1:3) — вещественный;
…

В разделах «Переменные» и «Массивы» желательно указать тип используемых переменных и массивов.

Работа с массивами сводится к работе с его элементами. Обращение к элементам массива осуществляется с помощью переменной с индексом. Так, x(1,5) — элемент массива x, расположенный на пересечении первой строки и пятого столбца.

Пример. Заданы два двухмерных массива x(1:2, 1:2) и y(1:2, 1:2):

$$x=\begin{pmatrix} 2 & -3 \\ 4 & 8 \end{pmatrix}; \quad y=\begin{pmatrix} 1 & 0 \\ 3 & 2 \end{pmatrix}.$$

Определить значение: H:=x(1, 1) y(2, 1) + x(2, 2).

После выборки указанных элементов и выполнения операции присваивания получим: H=14.

Следует особо отметить, что x(1) не равно x1, т.е. первый элемент массива x не имеет никакого отношения к переменной x1.

Некоторые языки высокого уровня дают возможность обрабатывать массивы целиком; в этом случае для обозначения массива достаточно указать только его имя. При одинаковой размерности, типе и длине массивов можно записать

c:=a+b; c:=a b; c:=a-b,

что означает соответственно суммирование, умножение и вычитание элементов массивов a и b (с одинаковыми индексами) и запись результатов в массив c.

Для того чтобы задать всем элементам массива одинаковые значения, например 5, можно записать: a:=5, где a — массив. В дальнейшем будем пользоваться такой записью, чтобы сократить, упростить написание алгоритма, считая элементарными предписания: ввод и вывод всего массива, присваивание всему массиву каких-либо значений и т.д. На самом деле, за редким исключением, такие операции можно осуществить только при работе с каждым элементом массива в отдельности, используя для этого циклические операторы.

Основным инструментом для работы с массивами являются операторы цикла. Порядок использования циклических операторов для обработки массивов рассмотрим без потери общности на следующих конкретных примерах.

Пример. Составить фрагмент алгоритма, в котором четным элементам массива x(1:100) будут присвоены значения квадратов индексов, а остальным элементам — нули.

Будем формировать массив по следующему правилу: сначала весь массив обнулим, а затем будем присваивать четным элементам нужное значение, т.е. записывать значения в массив с помощью цикла:

```
Список данных:

        k —  целый;

x(1:100) —  вещественный;
            x:=0;
            Цикл по k от 2 до 100 шаг 2;
                    x(k):=k;
            Конец-цикла;
```

В данном фрагменте алгоритма (и далее) под записью вида x:=0 (в том случае, конечно, если x — это массив) будем понимать работу с массивом целиком, т.е. каждый элемент массива x получает значение 0. На самом деле такое присваивание (ввод, вывод и т.д.) можно выполнить, например, в цикле: «… Цикл по k от 1 до 100; x(k):=0; Конец-цикла; …».

Пример. Составить алгоритм вычисления скалярного произведения векторов a и b длиной 10 по формуле:

$$c = \sum_{k=1}^{10} a_k \cdot b_k .$$

Смысл этой задачи сводится к выбору одноименных элементов из массивов, их перемножению с последующим суммированием полученных произведений. Алгоритм имеет следующий вид:

```
1. Начало;
2.  Список данных:
      k —  целый;
      a(1:10), b(1:10), c —  вещественный;
                    3.  Ввод(a, b);
                    4.  Вывод(a, b);
5.  c:=0;
6.   Цикл по k от 1 до 10;
7.      c:=c + a(k) b(k);
8.   Конец-цикла 6;
9.  Вывод(c);
10. Конец.
```

В этом алгоритме для организации накопления значений «c» мы предварительно очистили отведенную для этого ячейку памяти. В каждом новом повторе цикла к значению «c» прибавлялось очередное слагаемое.

Пример. Составить таблицу отклонений экспериментальных данных $V_1, V_2, \ldots V_{20}$ от их среднего значения.

Здесь необходимо сначала вычислить среднее значение, а затем построчно сформировать и отпечатать таблицу значений.

Фрагмент алгоритма решения этой задачи представлен на рис. 19.

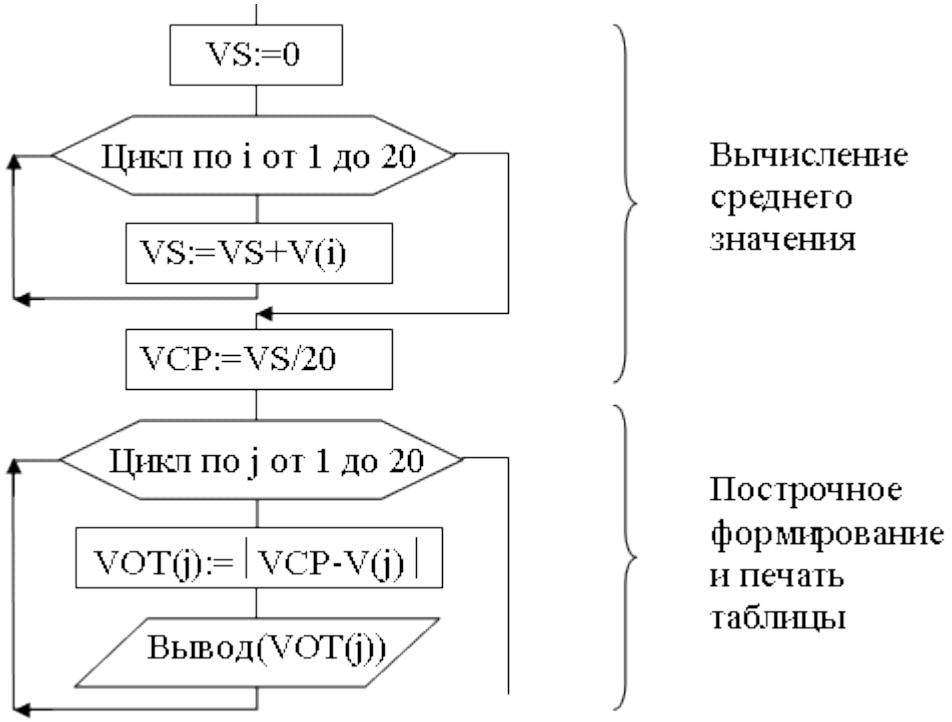


Рис. 19. Фрагмент алгоритма вычисления и печати таблицы отклонений от среднего значения.

Пример. Построить алгоритм для определения суммы элементов матрицы, состоящей из 2 строк и 3 столбцов, при условии, что первый элемент матрицы больше или равен нулю, или произведения элементов матрицы, если первый элемент меньше нуля.

Математическая постановка задачи будет иметь следующий вид.

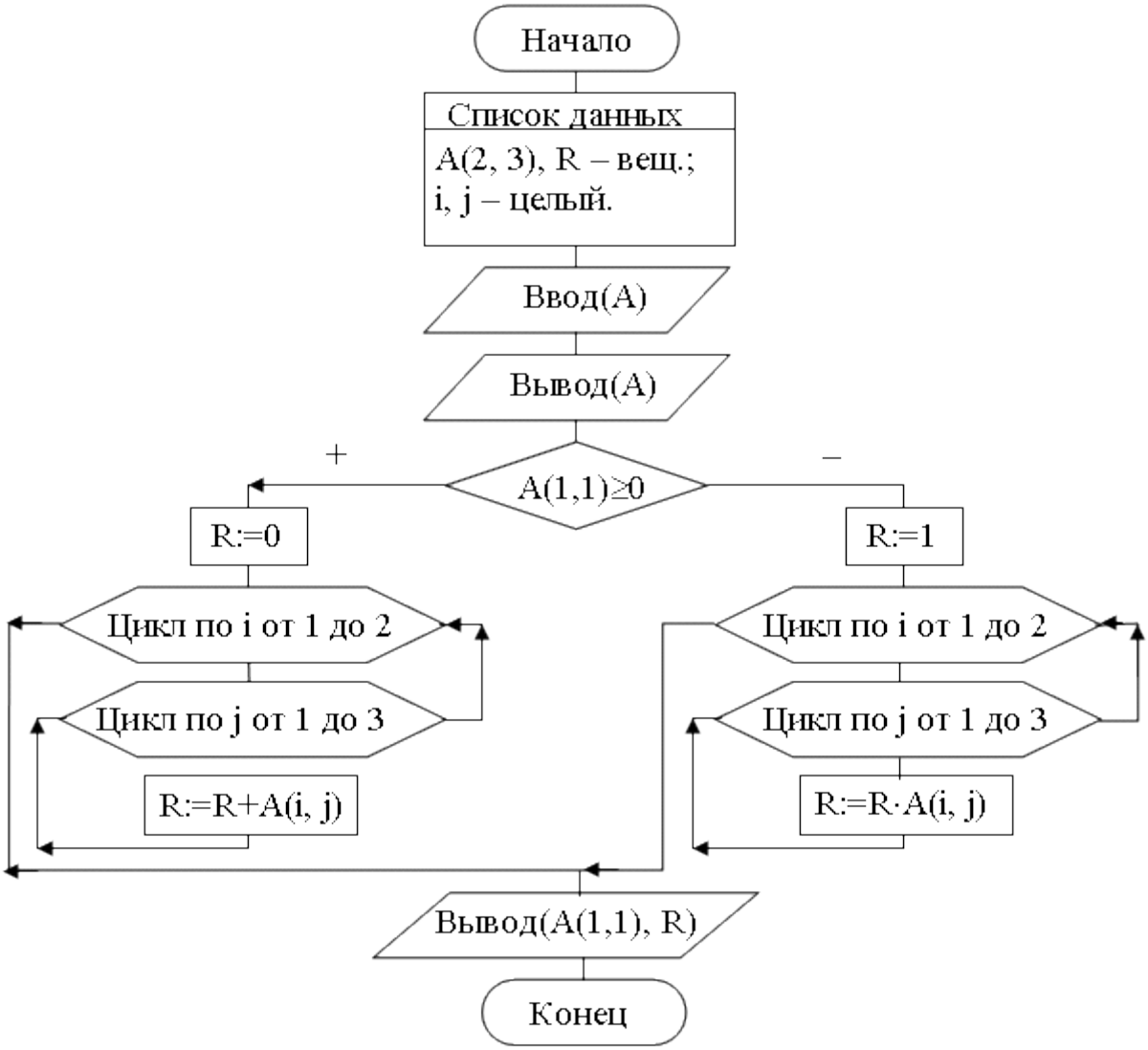
Задана матрица:

$$A=\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{pmatrix}.$$

Необходимо рассчитать величину R по формуле:

$$R=\begin{cases} \sum_{i=1}^{2}\sum_{j=1}^{3} a_{ij}, \text{ если } a_{11}\geq 0; \\ \prod_{i=1}^{2}\prod_{j=1}^{3} a_{ij}, \text{ если } a_{11}< 0. \end{cases}$$

Алгоритм решения этой задачи, заданной в форме ГСА, приведен на рис. 20.



Рассмотрим еще два примера, иллюстрирующих циклическую обработку массивов.

Пример. Составить фрагмент алгоритма, позволяющий вычислить сумму положительных элементов, лежащих под (над) дополнительной диагональю матрицы A(4,4):

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}.$$

В задачах такого рода важно определить закономерность изменения индексов элементов массива: в этой задаче нужно увидеть, что **сумма индексов элементов**, лежащих под дополнительной диагональю, что **больше 5** (над диагональю — меньше 5; на диагонали — равна 5). Фрагмент алгоритма решения задачи в виде псевдокодов выглядит так:

```
…
5.  S:=0;
6.    Цикл по i от 1 до 4;
7.        Цикл по j от 1 до 4;
8.              Если (i+j>5) и (A(i,j)>0) То
```

```
9.                  S:=S+A(i,j);
10.             Конец-Если 8;
11.       Конец-цикла 7;
12.   Конец-цикла 6;
13. Вывод(S);
…
```

Пример. Составить фрагмент алгоритма, позволяющий вычислить сумму отрицательных элементов, лежащих над (под) главной диагональю матрицы A(4,4):

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}.$$

В этой задаче важно увидеть, что **первый индекс** у элементов, лежащих над главной диагональю, **меньше второго** (под диагональю — первый индекс больше второго; на диагонали — индексы равны). Фрагмент алгоритма решения задачи очень похож на предыдущий, изменения потребует лишь пункт 8 (изменится проверяемое логическое условие). Фрагмент алгоритма решения этой задачи можно написать самостоятельно. В заключение рассмотрим еще один алгоритм, обеспечивающий сортировку массива. Реализуем алгоритмически «метод сортировки обменами».

Пример. Дана последовательность чисел $a_1, a_2, \ldots, a_{10}$ (одномерный массив a(10)). Требуется переставить числа в порядке возрастания. Для этого сравниваются два соседних числа a_i и a_{i+1}. Если $a_i > a_{i+1}$, то делается перестановка. Так продолжается до тех пор, пока не будут выполнены все перестановки. Алгоритм решения задачи показан на рис. 21.

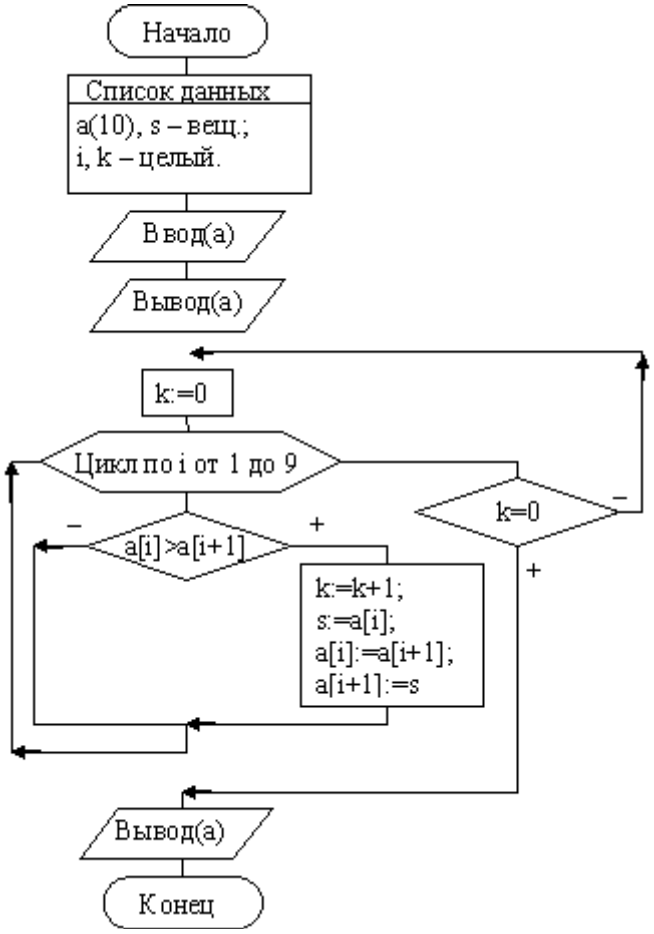


Рис. 21. Алгоритм сортировки массива.

Типовые конструкции процедурных языков программирования

Типовые конструкции процедурных языков программирования рассмотрим на примере языка Паскаль.

Язык Паскаль был разработан в начале 70-х годов XX века профессором Никлаусом Виртом (Швейцария) как инструмент для систематического обучения программированию. Для этого в состав языка включены элементы структурного программирования (последовательные, разветвляющиеся и циклические структуры) и структуры данных (массивы, записи, файлы и т.п.). Это простой язык, однако все, чем он располагает, настолько соответствует исполняемым задачам, что на практике этот язык оказывается продуктивнее, чем его более развитые конкуренты. Широкое распространение язык получил благодаря усилиям французского математика Филипа Кана, разработавшего в 1983 г. компактный, быстродействующий и дешевый компилятор, который он назвал Турбо Паскаль.

В 1992 г. фирма Borland International выпустила в свет очередную версию языка программирования — Turbo Pascal 7.0, отличающуюся от предыдущих улучшенным интерфейсом пользователя и более быстрым компилятором. Естественно, в новую версию вошли все предыдущие наработки: использование мыши, многооконный режим работы, возможность включения программ на языке Ассемблера, возможность создания объектно-ориентированных программ и т.д.

К основным особенностям языка программирования Паскаль следует отнести довольно строгие требования к структуре программы: оператор начала программы, блок (разделы описаний меток, констант, типов, переменных, процедур и функций, операторов). Стандартный Паскаль предполагает запись вышеперечисленных разделов программы в строгой последовательности. Версии языка, появившиеся в сравнительно недавнем прошлом (в том числе и ТП 7.0), в определенной мере свободны от этого ограничения — разделы описания можно менять местами (это не относится к разделу операторов, которым должна заканчиваться программа).

Операторы Паскаль-программы должны заканчиваться точкой с запятой, в одной строке может быть записано несколько операторов, что тоже может быть отнесено к особенностям языка. В Паскале есть все типы циклов: «Пока», «До», «С параметром». Хорошо продумано использование выбирающих

операторов: операторы безусловного и условного переходов, оператор варианта. В Паскале имеется возможность подключения библиотечных модулей, расширяющих, например, графические возможности языка. Вместе с тем отсутствие некоторых, «привычных» функций (например, функции тангенса, функции возведения в степень и т.д.) может на первых порах изучения языка вызывать недоумение, а порой негодование у рядового пользователя. Безусловно, как и любой язык программирования, Паскаль имеет свои достоинства и недостатки.

Алфавит и словарь

Основой любого языка программирования является алфавит — набор допустимых символов, которые можно использовать для записи программ. Язык Паскаль оперирует со следующим набором символов:

- латинские прописные и строчные буквы (A, B, C, …, x, y, z);
- арабские цифры (0, 1, 2, …, 7, 8, 9);
- символ подчеркивания;
- специальные символы и символы-разделители.

Специальные символы и их комбинации чаще всего используются для обозначения операций. Список специальных символов языка Паскаль приведен в табл. 4.

Изображение	Название	Изображение	Название
+	Плюс	=	Равно
-	Минус	>	Больше
*	Звездочка	<	Меньше
/	Наклонная черта	>=	Больше или равно
.	Точка	<=	Меньше или равно
,	Запятая	<>	Не равно
:	Двоеточие	( )	Круглые скобки
;	Точка с запятой	[ ]	Квадратные скобки
'	Апостроф	{ }	Фигурные скобки
^	«Крышка»	: =	Присваивание
@	Знак адреса	. .	Две точки
#	Знак номера	(* *)	Скобка-звездочка
$	Знак доллара	(. .)	Скобка-точка

Таблица 4. Специальные символы.

Символами-разделителями считаются пробелы, комментарии и концы строк. Комментарии могут содержать любые символы, заключенные в ограничивающие скобки. В качестве ограничивающих скобок в этом случае используются фигурные скобки { } или скобки со звездочкой (* *). Например:

{Главная программа} (*Prima 1*)

С помощью перечисленных символов формируются:

- имена;
- ключевые (служебные) слова;
- числа;
- строки символов;
- метки.

Имена в Паскале применяются для обозначения различных конструкций языка: констант, переменных, типов, границ, процедур и функций. Имена обязательно начинаются с латинской буквы или символа подчеркивания «_» ; за ними могут следовать в любой комбинации латинские буквы и цифры. ТП 7.0 не различает прописные и строчные буквы. Длина имени может быть любой, но сравнение имен производится по первым 63 символам. Не допускается использование для написания имен специальных символов и символов-разделителей.

Например: _x, B12, Stack — правильно; Label. 4, Root-3 — неправильно.

Некоторые слова имеют в Паскале заранее определенный смысл (это имена математических функций, типов данных, констант и т.д.). Такие имена называются **зарезервированными**или **ключевыми**. Имена, применяемые пользователем для обозначения конструкций, не должны совпадать по написанию с ключевыми словами, они называются словами пользователя. Ключевые слова — это некоторое множество имен, зарезервированных в языке для написания операторов и других конструкций.

Список зарезервированных слов в ТП 7.0 приведен в табл. 5.

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

Таблица 5. Зарезервированные слова в ТП 7.0.

Имя метки перехода может представлять собой целое число (от 0 до 9999), строку символов или символьно-цифровую конструкцию. Например: 1, 9999, h2, 4t32e, metka1.

Строка символов — это последовательность символов, заключенная в апострофы. Если нужно включить в строку сам апостроф, то он записывается дважды.

Например: '*', ';', 'Begin', 'A', 'B', 'C' .

Числа, обозначающие целые и вещественные значения, записываются в Паскале в десятичной системе счисления. Перед любым числом может стоять знак «+» или «-». В вещественном числе целая часть от дробной отделяется точкой. Например: 3, — 50, 6278244 — целые числа; — 0.6, 15.75 — числа с фиксированной точкой; 5E-8, 9.422E+08, 1E10 — числа с плавающей точкой.

Вещественные числа, содержащие десятичную точку, должны иметь перед ней и после нее, по крайней мере, по одной цифре. Числа с плавающей точкой имеют мантиссу (число с фиксированной точкой) и масштабный множитель (порядок). Например, 4.897E+08 означает, что число 4.897 надо умножить на десять в восьмой степени. **Порядок** — это целое двухразрядное число со знаком. Пробелы внутри чисел недопустимы. Числа, как и имена, нельзя разрывать при переносе текста программы с одной строки на другую. Например, неправильно написаны числа:

0,2 — запятая вместо точки (правильно 0.2);
1 000 — число содержит пробел (правильно 1000);
E3 — нет мантиссы перед E (правильно 1E3);
.08 — нет целой части (правильно 0.08);
120. — нет дробной части (правильно 120.0 или 1.2E2).

Типы данных, используемые в языке Паскаль

Любой тип данных определяет множество значений, которые может принимать та или иная переменная, и множество операций, которые можно к ним применить. Типы данных, используемые в языке ТП 7.0, можно разделить на три группы:

- Простые типы (целочисленный, вещественный, символьный, строковый, логический, интервальный, перечисляемый).
- Структурированные типы (запись, массив, множество, файл).
- Ссылочные типы.

Рассмотрим более подробно простые типы данных.

Целочисленные типы. Переменные целого типа могут принимать только целые значения. Использование десятичной точки в изображении целого числа недопустимо (например: 35.0 — неправильно, 35 — правильно). Если используются операнды целого типа, то операции сложение «+», вычитание «-», умножение «*», Mod и Div (см. прил. 3) дают результат целого типа. Операция mod означает взятие остатка от целочисленного деления (например: 13 Mod 5 = 3; — 5 Mod 2 = — 1). Операция div является операцией целочисленного деления с отбрасыванием остатка (например: 13 Div 5 = 2; — 5 Div 2 = — 2).

Чаще всего для обозначения переменных целого типа пользователи выбирают тип **Integer**, который может вмещать целые числа, лежащие в диапазоне от — 32768 до 32767.

Вещественные типы. Вещественный тип формируется из элементов подмножества вещественных чисел, представленных в форматах с фиксированной или плавающей точкой (например: 3.14, 2.5E+01).

Чаще всего для обозначения переменных вещественного типа пользователи выбирают тип **Real**. Все операции над данными вещественного типа приближенные. Операции, дающие вещественные результаты, приведены в табл. 6.

Операция	Действие	Тип операнда	Тип результата
Арифметические + — * / Div Mod	Сложение Вычитание Умножение Деление Целочисленное деление Остаток от деления	целый или вещественный — — целый —	целый или вещественный — вещественный целый —
Сравнения = <> < > <= >= In	Равенство Неравенство Меньше Больше Меньше или равно Больше или равно Присутствие в множестве	любой — простой — — — 1-й операнд простой, 2-й — множество	логический — — — — — —
Логические			

Not	Отрицание	логический	логический
Or	Дизъюнкция	—	—
And	Конъюнкция	—	—
Xor	Исключающее ИЛИ	—	—
С множествами			
+	Объединение	множество	множество
—	Разность	—	—
*	Пересечение	—	—

Таблица 6. Перечень операций языка Паскаль.

Символьный тип. Значениями символьного типа **Char** являются элементы конечного и упорядоченного множества символов. Символьные данные используются для облегчения общения человека с компьютером. К сожалению, не существует единого стандарта множества символов. Чаще всего используются символы американского стандарта ASCII (American Standard Code for Information Interchange).

Значения типа Char записываются одним символом, заключенным в апострофы (например: '*', 'A', '!'), или задаются непосредственно ASCII- кодом символа (например: #65 — код символа 'A'). В памяти компьютера переменная типа Char занимает один байт памяти.

В рамках этого типа десятичные цифры упорядочены в соответствии с их численными значениями (например: '5' < '6'). Буквы упорядочены в алфавитном порядке (например: 'A' < 'B').

Строковый тип. Тип данных **String** предназначен для хранения последовательности символов (элементов типа Char). Строка символов должна быть заключена в апострофы (например: 'строка символов'). Максимальная длина строки — 255 символов. В машине каждый символ строки занимает байт памяти. Кроме того, один байт отводится дополнительно для хранения информации о длине строки. Таким образом, область памяти, выделяемая под строку символов, всегда на один байт больше той, которую вы запрашиваете при объявлении переменной.

Логический тип. Данные логического типа **Boolean** могут принимать одно из двух значений, обозначаемых стандартными именами True (истина, 1) и False (ложь, 0). Эти величины упорядочены следующим образом: True > False. Чаще всего на практике используются следующие логические операции, дающие логические результаты применительно к логическим операндам:

Not — инверсия (логическое отрицание, операция «НЕ»);
And — конъюнкция (логическое умножение, операция «И»);
Or — дизъюнкция (логическое сложение, операция «ИЛИ»);
Xor — операция «исключающее ИЛИ».

Логические операции дают следующие результаты:

Not False = True; Not True = False;

False And False = False; False Or False = False; False Xor False = False;
False And True = False; False Or True = True; False Xor True = True;
True And False = False; True Or False = True; True Xor False = True;
True And True = True; True Or True = True; True Xor True = False.

К переменным логического типа могут применяться операции сравнения «=» и «<>».

Переменные типа Boolean занимают 1 байт памяти.

Интервальный тип (Тип-диапазон). Интервальный (ограниченный, Subrange) тип описывается так:

имя_типа = min .. max;

где min — левая граница диапазона, max — правая граница диапазона. Диапазон значений задается с помощью любого простого типа данных, за исключением вещественного. Например, запись Period=1975..1998; будет означать, что переменные типа Period могут принимать значения целых чисел, лежащих в интервале от 1975 до 1998. Если попытаться присвоить этим переменным значения, не входящие в заданный интервал, — это может привести к ошибке в программе.

Перечисляемый тип. Перечисляемый тип конструируется пользователем путем перечисления в круглых скобках через запятую всех констант, которые должны принадлежать этому типу данных. Константы типа обязательно задаются именами. Перечисляемый тип описывается так:

имя_типа = (имя1, имя2,…, имяN);

г

де имя1, имя2,…, имяN — имена конкретных констант, включаемых в указанный тип. Эти значения упорядочены, т.е. имя1<имя2…имяN. Например, запись Color(Red, Yellow, Green, Blue); будет означать, что переменные типа Color могут принимать одно из перечисленных значений: Red, Yellow, Green или Blue. К переменным этого типа применимы операции сравнения. Значения переменных перечисляемого типа не могут вводиться оператором ReadLn и выводиться оператором WriteLn./p>

Структура программы на языке Паскаль

Паскаль-программа состоит из двух частей: описания данных и описания действий. Описание данных содержит сведения обо всех данных, обрабатываемых программой. Если программа содержит отдельные процедуры и функции, то их описание также располагается в этой части. Описание действий включает все операторы, с помощью которых осуществляется обработка данных.

Программа, записанная на языке Паскаль, должна иметь заголовок и тело программы. Заголовок программы состоит из зарезервированного слова **Program** и имени программы. В ТП 7.0, в отличие от стандартного Паскаля, эта строка необязательна. После заголовка программы в некоторых случаях записывается оператор **Uses**, используемый для подключения к тексту программы стандартных библиотечных модулей, например: Uses Crt, Graph, Strings, Printer;. Оператор Uses может быть использован в программе только один раз.

Тело программы состоит из шести разделов:

- раздел описания меток (Label);
- раздел описания констант (Const);
- раздел описания типов (Type);
- раздел описания переменных (Var);
- раздел описания процедур и функций;
- раздел операторов.

В стандартном Паскале менять разделы местами нельзя. В ТП 7.0 соблюдение такого порядка следования разделов (за исключением раздела операторов) необязательно, за исключением раздела описания типов, который должен предшествовать ссылкам на его содержимое.

В качестве примера рассмотрим структуру Паскаль-программы (табл. 7), реализующей алгоритм определения суммы или произведения элементов матрицы в зависимости от значения первого элемента.

Заголовок программы и раздел описаний

В заголовке программы указывается имя программы. Общий вид заголовка:

Program имя;

В ТП 7.0 эта строка необязательна, однако рекомендуется в начале программы все-таки указывать имя программы и ее версию — это повысит «удобочитаемость» программы и поможет в дальнейшем упорядочить вашу работу. К написанию имени программы предъявляются такие же требования, как и к

именам констант, переменных и т.д. В примере, приведенном в табл. 7, заголовком программы является строка:

Program MATR;

В следующей строке программы (см. табл. 7) использован оператор USES для подключения к программе библиотеки CRT, в которой находится, например, такая процедура, как ClrScr (очистка экрана).

Раздел описания меток (Label). Любой оператор программы может быть снабжен меткой перехода. Метка представляет собой строку символов (например: Metka1, 1c2) или целое положительное число, содержащее не более четырех цифр (от 0 до 9999). Метки отделяются от операторов двоеточием. Все метки, встречающиеся в программе, должны быть описаны в разделе Label.

Общий вид раздела:

Label метка1, метка2,…;

В примере, приведенном в табл. 9, раздел описания меток имеет вид:

Label 10, 20;

а метки указывают на строки:

```
10:   If A[1,1]>=0 Then Sum(A,n,m,R)
                        Else Pro(A,n,m,R);
20:   WriteLn('A(1,1)=',A[1,1],' R=',R);
```

<table>
<tr><td colspan="3">Структура программы</td><td>Пример</td></tr>
<tr><td colspan="3">Заголовок программы</td><td>Program MATR;</td></tr>
<tr><td colspan="3">Оператор Uses</td><td>Uses Crt;</td></tr>
<tr><td rowspan="6">Тело программы (блок)</td><td rowspan="5">Описание данных</td><td>1. Раздел описания меток (Label)</td><td>Label 10, 20;</td></tr>
<tr><td>2. Раздел описания констант (Const)</td><td>Const n=2; m=3;</td></tr>
<tr><td>3. Раздел описания типов (Type)</td><td>Type Mat=array[1..n,1..m] of real;</td></tr>
<tr><td>4. Раздел описания переменных (Var)</td><td>Var A:Mat;
R:real;
i, j:integer;</td></tr>
<tr><td>5. Раздел описания процедур и функций (Procedure, Function)</td><td>Procedure Sum(A:Mat;n,m:integer;Var R:real);
Var i, j:integer;
Begin R:=0;
For i:=1 To n Do
For j:=1 To m Do
R:=R+A[i,j];
End;
Procedure Pro(A:Mat;n,m:integer;Var R:real);
Var i, j:integer;
Begin R:=1;
For i:=1 To n Do
For j:=1 To m Do
R:=R*A[i,j];
End;</td></tr>
<tr><td>Описание действий</td><td>6. Раздел операторов</td><td>Begin
ClrScr;
For i:=1 To n Do
For j:=1 To m Do Begin
Read(A[i,j]);
WriteLn(A[i,j]); End;</td></tr>
</table>

			10: If A[1,1]>=0 Then Sum(A,n,m,R) Else Pro(A,n,m,R); 20: WriteLn('A(1,1)=',A[1,1],' R=',R); End.

Раздел описания констант (Const). Под константой в языке Паскаль подразумевается число, отделенный символ или строка символов.

Общий вид раздела описания констант:
Const имя1= значение; имя2= значение;…
Здесь имя1 и имя2 — имена констант, описываемых в разделе.

Числовые константы могут быть целыми и вещественными. Вещественные константы могут быть представлены в двух формах: с фиксированной и плавающей точкой. Символьная константа представляет собой символ, заключенный в апострофы. Константы-строки — это последовательность символов, заключенная в апострофы. Длиной строки называется число элементов в ней. Логические (булевы) константы имеют два значения — True и False.

Пример:
```
Const a=23; b= — 15;
      c=4.35; d=7.12E+02;
      e='A'; f='5';
      g='Dubna'; h='123456';
      r=True; s=False;
```
В приведенном примере (см. табл. 7) раздел описания констант имеет вид:
```
Const n=2; m=3;
```

Раздел описания типов (Type). Этот раздел используется, если в программе вводятся типы, отличные от стандартных.

Общий вид раздела описания типов:
Type имя1= тип; имя2= тип;…
где имя1 и имя2 — имена типов.
В примере, рассмотренном в табл. 7, раздел описания типов имеет вид:
```
Type Mat=array[1..n,1..m] of real;
```

Здесь тип Mat определен как вещественный двухмерный массив с индексами интервального (ограниченного) типа.

Раздел описания типов включается в блок, если необходимо предусмотреть несколько имен для стандартных типов или ввести новые имена. Эти возможности делают программу более наглядной и отражающей смысл решаемой задачи.

Раздел описания переменных (Var) имеет следующий вид:
```
Var имя1, имя2,…: тип;
    имя11, имя12,…: тип;…
```

В разделе VAR вводится имя каждой переменной и указывается, к какому типу эта переменная принадлежит. Каждая переменная должна быть описана до ее использования в программе и отнесена к одному и только одному типу.

В примере, рассмотренном в табл. 7, раздел описания переменных имеет вид:
```
Var A:Mat;
    R:real;
    i, j:integer;
```

Здесь A — массив типа Mat; R — вещественная переменная; i и j — переменные целого типа.

Таким образом, любая переменная в программе может быть описана двумя способами: либо описанием самого типа в разделе Var, либо указанием в разделе Var имени типа, которое предварительно описано в разделе Type.

Разделу описания данных в программе на языке Паскаль соответствует конструкция «Список данных» в алгоритме.

Операторы языка Паскаль

Операторы в языке Паскаль предназначены для описания действий. Операторы сосредоточены в специальном разделе (разделе операторов), который располагается в конце блока. Наличие этого раздела в программе обязательно.

Раздел операторов имеет следующий вид:

```
Begin
    оператор1;
    оператор2;
    . . . . . .
    операторN;
End.
```

В этот раздел могут быть включены пустые, простые и сложные операторы. Любой из них может быть помечен меткой. Операторы отделяются друг от друга точкой с запятой. Если оператор стоит **перед словом Else**, то в конце оператора **точка с запятой не ставится.**

Пустой оператор («;») не содержит никаких символов и не обозначает никаких действий.

Простым называется оператор, в который не входят как составные части другие операторы. К простым операторам относятся: оператор присваивания, операторы процедур ввода и вывода данных и оператор перехода.

Сложные операторы — это конструкции, состоящие из других (простых) операторов, заключенных в операторные скобки вида Begin…End;.

Оператор присваивания является основным, фундаментальным оператором. Общий вид оператора:

имя_переменной:=выражение; Например: F:=3*c+2*Sin(x+Pi/2);

Выражение состоит из операндов и операций. Операндом может быть константа, переменная или функция. Операции объединяют операнды и выражения. Язык Паскаль содержит ряд предварительно описанных (встроенных) функций, перечень которых приведен в табл. 8.

Запись функции на Паскале	Обычная математическая запись функции или ее назначение
	Математические функции
Abs(x)	|x|- абсолютное значение величины x
Exp(x)	e^x или exp(x)
Ln(x)	ln x — значение натурального логарифма
Pi	π — число Пи
Sqr(x)	x^2
Sqrt(x)	$\sqrt{x}$, x > 0
Sin(x)	sin x, x — в радианах
Cos(x)	cos x, x — в радианах

Arctan(x)	arctg x, x — в радианах
Trunc(x)	Целая часть числа x, получаемая путем отбрасывания дробной части (например: Trunc(-3.6) = — 3)
Round(x)	Целая часть числа x, получаемая путем округления до ближайшего большего целого по абсолютной величине числа (например: Round(-3.6) = — 4)
Int(x)	Целая часть числа x, получаемая путем округления до ближайшего меньшего целого (например: Int(-3.6) = — 4.0)
Frac(x)	Дробная часть числа x
Random(x)	Случайное число от 0 до x (например: Random(4)+1 — выдает случайное число из интервала от 1 до 4)
Random	Случайное число от 0 до 1
Odd(x)	Выдает значение True, если целое число x нечетно
Chr(x)	**Символьные и строковые функции** Символ, соответствующий ASCII-коду числа x (x — тип Byte)
Ord(x)	ASCII-код, соответствующий символу x (x — тип Char)
Pred(x)	Символ, предшествующий по ASCII-коду символу x (x — тип Char)
Succ(x)	Символ, следующий по ASCII-коду за символом x (x — тип Char)
UpCase(x)	Преобразование символа x из строчных букв латинского алфавита в прописные (x — тип Char)
Length(x)	Определение длины текстовой строки x (x — тип String)
Copy(x,n,m)	Копирование фрагмента, выделяемого из строки x, начиная с позиции n, длиной m символов (x — тип String)
Pos(x,y)	Номер позиции в строке y, с которого начинается текстовый фрагмент x (x, y — тип String). Если фрагмент не найден, функция возвращает нуль.

Таблица 8. Встроенные функции языка ТП 7.0.

Вид выражения однозначно определяет правила его вычисления: действия выполняются слева направо с соблюдением старшинства, указанного в таблице приоритетов (табл. 9).

Очередность выполнения операций	Операции
1-я	@, +, — , Not
2-я	*, / , Div, Mod, And, Shr, Shl
3-я	+ , — , Or, Xor
4-я	In, >, <, =, <>, >=, <=

Таблица 9. Таблица приоритетов.

Операция @ — операция взятия адреса.

Операция Shr (Shl) — циклический сдвиг вправо (влево).

Операция In является операцией отношения и позволяет определять присутствие элемента в множестве, ее результат — true или false. Любое выражение в скобках вычисляется раньше, чем выполняется операция, предшествующая скобкам. Так, в примере F:=3*c+2*Sin(x+Pi/2); сначала вычисляется значение

x+Pi/2, затем определяется значение функции Sin(x+Pi/2), затем выполняется операция умножения 3*c, затем — вторая операция умножения 2*Sin(x+Pi/2) и последней выполняется операция сложения.

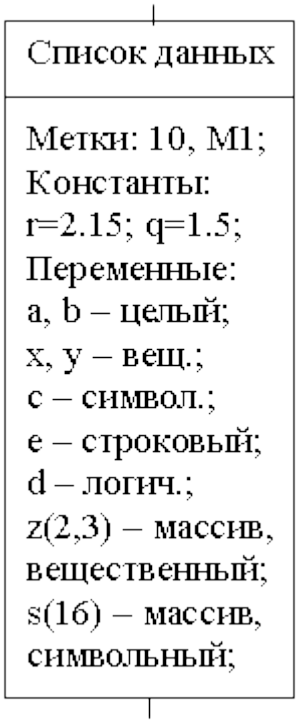
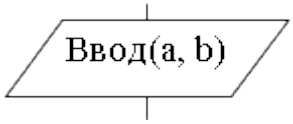
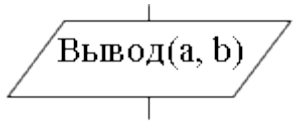
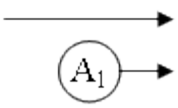
Операция присваивания допускается для переменных всех типов, за исключением типа файл. В операторе присваивания переменная, стоящая слева от знака присваивания, и выражение, стоящее справа от него, должны иметь один и тот же тип. Переменные и выражения, относящиеся к символьному или строковому типам в операции присваивания, должны иметь одинаковое число элементов. Разрешается присваивать переменной типа Real выражение типа Integer.

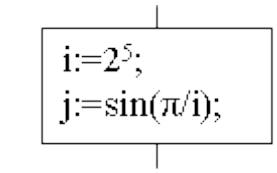
Примеры: y:=Sin(x)+Cos(x); a:=(b+c)*(1.01+d); k:=n>m;

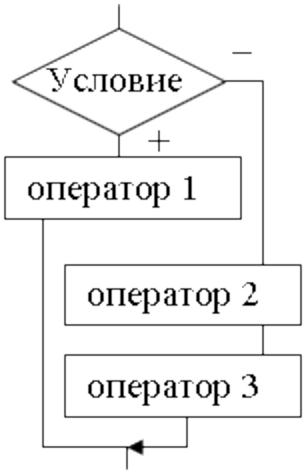
Здесь переменным y и a присваивается значение типа real. Логическая переменная k принимает значение True или False.

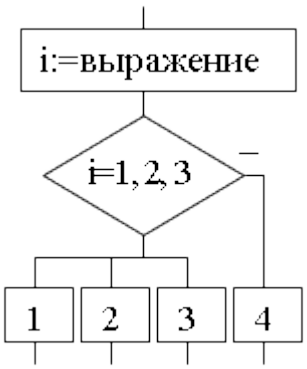
Типовые конструкции языка Паскаль

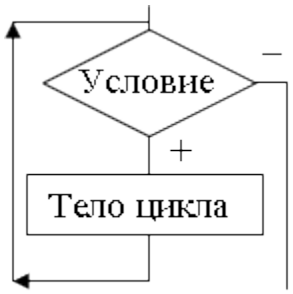
Перевод алгоритмических конструкций в конструкции языка программирования ТП 7.0 показан в табл. 10.

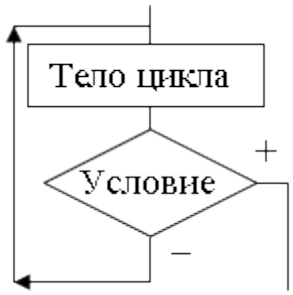
Обозначения на блок-схеме	Запись на языке предписаний	Запись на языке Паскаль
	Начало;	Program имя;
	Список данных: Метки: 10, M1; Константы: r=2.15; q=1.5; Переменные: a, b — целый; x, y — вещ.; c — символ.; e — строковый; d — логич.; z(2,3) — массив, вещественный; s(16) — массив, символьный;	Не переводится Label 10, M1; Const r=2.15; q=1.5; Var a, b: integer; x, y: real; c: char; e: string; d: boolean; z: array[1..2, 1..3] of real; s: array[1..16] of char;
	Ввод(a, b);	Read(a, b);
	Вывод(a, b);	WriteLn('a=', a, 'b=', b);
	Перейти к оператору строке метке	Goto метка;

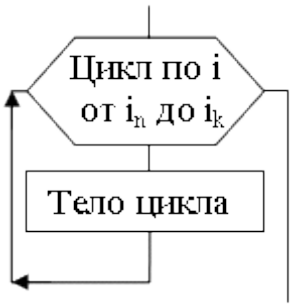
	i:=2^5; j:=sin(π/i);	i:=Exp(5*Ln(2)); j:=Sin(Pi/i);

	Если условие То
	оператор 1
	Иначе
	
	оператор 2;
	оператор 3;
	
	Конец-Если;
		If условие Then
		оператор 1
		Else
		Begin
		оператор 2;
		оператор 3;
		End;
		Не переводится

	i:=выражение;
	Если i=1 То
	оператор 1;
	Если i=2 То
	оператор 2;
	Если i=3 То
	оператор 3;
	Иначе
	оператор 4;
	Конец-Если;
	Конец-Если;
	Конец-Если;
		Case i Of
		1: оператор 1;
		
		2: оператор 2;
		
		3: оператор 3;
		Else
		оператор 4;
		End;

	Цикл-ПОКА
	(условие);
	операторы тела
	цикла;
	Конец-цикла;
		While (условие) Do
		Begin
		операторы тела
		цикла;
		End;
		Не переводится

	Цикл-ДО (условие); операторы тела цикла; Конец-цикла;	Repeat операторы тела цикла; Until (условие); Не переводится

	Цикл по i
	от i_n до i_k;
	операторы тела цикла;
	Конец-цикла;
		Цикл по возрастающему параметру (in < ik):
		For i:=in To ik Do
		Begin
		операторы тела цикла;
		End;
		Не переводится
		
		Цикл по убывающему параметру (in>ik):
		For i:=in DownTo ik Do
		Begin
		операторы тела цикла;
		End;

	Конец;	End.

Таблица 10. Таблица перевода основных алгоритмических конструкций в конструкции языка ТП 7.0.