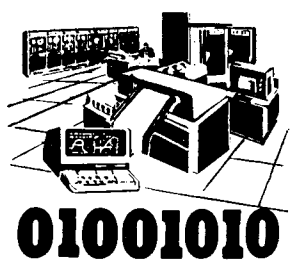
ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА

Под редакцией В. Н. Ларионова

Рекомендовано Комитетом по профессиональному образованию
Министерства образования Российской Федерации
в качестве учебного пособия
для студентов инженерно-педагогических специальностей вузов



01001010

Москва
«Высшая школа» 1992

ББК 32.97
И74
УДК 681.31

Рецензенты: канд. техн. наук В. О. Хорошилов — зав. кафедрой информатики и экономики Московского филиала Всесоюзного института повышения квалификации руководящих работников и специалистов профтехобразования; И. Н. Дмитриевская — преподаватель СПТУ № 148 г. Москвы.

В. В. Вьюхин, С. В. Кудымов, В. Г. Накрохин, В. Н. Ларионов, В. Л. Мучник, М. И. Школьник

Информатика и вычислительная техника: Учеб. пособие для студ. вузов инж.-педагогич. спец./*В. В. Вьюхин, С. В. Кудымов, В. Г. Накрохин* и др.; Под ред. *В. Н. Ларионова*. — М.: Высш. шк., 1992. — 287 с.: ил.

И74

ISBN 5-06-001793-1

В книге приведены основные сведения о компьютерах, рассмотрена технология решения задач различных классов, описано типовое базовое и программное обеспечение персональных компьютеров. Пособие ориентировано на использование комплекта учебной вычислительной техники «Корвет».

И $\frac{2404000000(4309000000)-048}{001(01)-92}$ 170—91 (Инф.-письмо) ББК 32.97
6Ф7

ISBN 5-06-001793-1

ПРЕДИСЛОВИЕ

Данное учебное пособие ориентировано на студентов инженерно-педагогических специальностей — будущих специалистов по подготовке квалифицированных рабочих. Оно соответствует рекомендованной программе курса «Информатика и вычислительная техника». Кроме того, пособие может быть использовано слушателями системы повышения квалификации работников профтехобразования и студентами педагогических институтов.

В пособие включен значительный объем материала по программированию, хотя авторы не считают это главной частью информатики для инженеров-педагогов. Опыт показывает недостаточные (или, по крайней мере, неравномерные) знания основ информатики у поступающих в институты выпускников средних учебных заведений. Традиционное изложение материала по алгоритмизации и программированию должно, по мнению авторов, способствовать достижению приемлемого уровня.

В качестве изучаемого языка выбрана MSX-версия БЕЙСИКа, хотя этот язык не соответствует сложившимся классическим представлениям. Преимуществом языка является его доступность для изучения на базе распространенных учебных комплектов персональных компьютеров «Ямаха», «Корвет» и УКНЦ, которыми оснащены не только вузы, но и те учебные заведения, где в перспективе могут работать выпускники инженерно-педагогических специальностей.

Недостатки языка БЕЙСИК авторы попытались компенсировать включением материала по абстрактным типам данных, использованием общего понятия о структурах управления (алгоритмических структурах), разъяснением смысла параметров подпрограмм и включением справочного приложения по языку ПАСКАЛЬ. Таким образом, читатель может получить общее представление о языках программирования.

В пособии достаточно подробно излагаются вопросы отладки и тестирования программ. Значительное

внимание уделяется базовому и прикладному программному обеспечению. Приводятся примеры способов общения пользователя с компьютером (с файловой системой, локальной сетью, системой управления базами данных).

В учебное пособие включен материал по разработке педагогических программных средств и их применению в учебном процессе. Хотя в теоретическом плане этот материал еще не устоялся, но для будущих инженеров-педагогов он является ключевым. Специфическим для инженерно-педагогических специальностей является также более развернутое изложение сведений о компьютерах и представлении информации в их устройствах. В дальнейшем эти сведения должны стать базой для изучения промышленного учебного оборудования, оснащенного микропроцессорными средствами (например, станков с числовым программным управлением, роботов, тренажеров).

При подборе примеров авторы старались увязать их с реальными задачами либо проводили подробный разбор примеров в главах, посвященных разработке педагогических программных средств, машинной графике и технологии решения задач с использованием компьютера. Последняя глава построена на одном содержательном примере, связанном с задачей линейного программирования, незнакомой первокурсникам, однако это не является препятствием для понимания, хотя требует известного напряжения при чтении. Такой пример позволяет проиллюстрировать все этапы решения.

В пособие включены иллюстрации, позволяющие избежать многословных объяснений и дающие читателю наглядное представление об изучаемых объектах. В конце глав (кроме трех последних) приведены упражнения. Некоторые из них позволяют проконтролировать знания, а другие расширяют и уточняют отдельные важные понятия. Решение предлагаемых задач принесет определенную пользу. Кроме иллюстраций и упражнений в пособие включены тексты программ. Авторы надеются, что у читателей есть возможность доступа к компьютеру и может возникнуть желание проэкспериментировать с приведенными программами.

Материал между авторами распределяется следу-

ющим образом: гл. 4 и § 5.5 и 5.6 написаны В. В. Вьюхиным; гл. 5, 6 и § 9.3 — С. В. Кудымовым; введение, гл. 3 и § 1.1 — В. Н. Ларионовым; гл. 2, § 8.1 и 9.4 — В. Л. Мучником; гл. 1, 8, § 9.1 и 9.2 — В. Г. Накрохиным; гл. 7, 10 и приложения — М. И. Школьником.

Авторы выражают благодарность рецензентам: зав. кафедрой информатики и экономики Московского филиала ВИПК руководящих работников и специалистов профтехобразования канд. техн. наук В. О. Хорошилову и преподавателю СПТУ № 148 г. Москвы И. Н. Дмитриевской, которые внесли существенные предложения и замечания, способствующие улучшению рукописи, а также мл. научн. сотруднику Г. А. Митряевой, выполнившей все иллюстрации с помощью графического редактора.

Замечания и пожелания по улучшению содержания книги просим направлять по адресу: 101430, Москва, ГСП-4, Неглинная ул., д. 29/14, изд-во «Высшая школа».

Авторы

ВВЕДЕНИЕ

Любая деятельность человека связана с обработкой различной информации. При этом наибольший успех имеет тот, кто может качественно обработать достаточно большой объем информации за приемлемое время и эффективно использовать информацию в своей деятельности.

Естественно, что проблема создания различных средств и методов оперирования с информацией всегда привлекала внимание общества. Но качественный скачок произошел в конце 40-х годов в результате изобретения электронных вычислительных машин — компьютеров. Сейчас в некоторых развитых странах в сфере компьютерной обработки информации занято около половины трудоспособного населения, а вложенные сюда средства дают самую высокую прибыль.

Приведем несколько простейших примеров применения компьютерных информационных технологий.

Пример производственного характера — управление движением транспортного робота (штабелера) на автоматизированном складе. В этом случае компьютер хранит информацию о содержимом каждого складского контейнера, а также о местонахождении робота. Получив от соответствующих датчиков сигналы о необходимости складской операции, компьютер по определенному алгоритму вычисляет оптимальную траекторию движения робота и выдает соответствующие сигналы на его привод.

Известным примером применения компьютерной обработки информации является система «Сирена» для резервирования мест на авиарейсы. Она обеспечивает хранение большого объема данных о свободных и занятых местах на различные рейсы, быстрый поиск необходимой пользователю информации о свободных местах, обмен информацией между компьютером и значительным количеством удаленных от него устройств ввода-вывода, установленных на рабочих местах кассиров-операторов.

В информатике как области деятельности можно выделить несколько направлений:

научное (фундаментальные исследования процессов получения, передачи, представления, хранения и обработки информации, а также разработка технических и программных средств для оперирования с информацией);

промышленное (массовое производство компьютеров и других технических средств информатики);

социальное (изучение и решение социальных вопросов информатизации общества, в том числе компьютеризации учебного процесса).

Несмотря на то что становление информатики как науки еще не завершилось, учебная дисциплина «Информатика и вычислительная техника» постепенно входит в число базовых дисциплин для инженерно-педагогического образования.

Незавершенность процесса становления и связанное с этим разнообразие взглядов на суть дисциплины в известной степени иллюстрируются динамизмом терминологии.

Общепринятое сейчас название «информатика» представляет собой русскую транскрипцию французского термина Informatique, образованного из слов information (информация) и automatique (автоматика). В США и Англии для названия дисциплины часто используется сочетание слов Computer Science («наука о вычислительных машинах»). Датские специалисты как альтернативный предлагают термин Datalogi — «наука о данных», т. е. об информации.

Для студентов новизна информатики может быть одним из мотивов ее изучения (хотя именно из-за новизны приобретение знаний и их систематизация требуют значительных усилий). Но, конечно, имеются и более серьезные причины, по которым изучению информатики следует уделить достаточное внимание. Перечислим их:

ЗНАНИЕ ИНФОРМАТИКИ СПОСОБСТВУЕТ ПОВЫШЕНИЮ ЭФФЕКТИВНОСТИ БУДУЩЕЙ ПРОФЕССИОНАЛЬНОЙ ДЕЯТЕЛЬНОСТИ

Квалифицированное применение компьютеров позволяет активизировать, интенсифицировать и индивидуализировать обучение, а также улучшить организацию учебного процесса

ЗНАНИЕ ИНФОРМАТИКИ СПОСОБСТВУЕТ

Например, умение программировать или грамотно применять готовые программы сокращает время на

ПОВЫШЕНИЮ ЭФФЕКТИВНОСТИ, УЧЕБНОЙ ДЕЯТЕЛЬНОСТИ СТУДЕНТА В ВУЗЕ

расчеты при выполнении лабораторных работ, в курсовом и дипломном проектировании

ЗНАНИЕ ИНФОРМАТИКИ СПОСОБСТВУЕТ СОЗДАНИЮ БЛАГОПРИЯТНОГО МНЕНИЯ О СПЕЦИАЛИСТЕ

Умение использовать компьютерные средства обработки информации становится одним из условий успешной профессиональной карьеры, а реализация творческого потенциала в области применения компьютеров в обучении, как правило, вызывает положительную оценку руководителей, коллег и учащихся.

ЗНАНИЕ ИНФОРМАТИКИ СПОСОБСТВУЕТ ПОЛОЖИТЕЛЬНОМУ ВОСПРИЯТИЮ ПРОЦЕССА ИНФОРМАТИЗАЦИИ ОБЩЕСТВА

Проникновение новых информационных технологий во все сферы деятельности человека приводит к необходимости приспосабливаться к жизни в компьютеризованной среде (в частности, повышать свою квалификацию в области использования компьютеров). Владение базовыми понятиями информатики дает основание для уверенности в успехе этого

Продуктивное изучение информатики подразумевает достаточно активную *самостоятельную работу студента за компьютером*, которая состоит в исследовании небольших готовых программ, разработке и программировании алгоритмов учебного характера, освоении типового программного обеспечения (систем текстообработки, графических пакетов, систем управления базами данных и т. п.). В идеальном случае работа за компьютером должна привести к реализации какого-либо собственного проекта обработки информации.

Глава 1
ОСНОВНЫЕ СВЕДЕНИЯ О КОМПЬЮТЕРАХ

В первой главе рассмотрены принципы автоматизированной обработки информации, способы кодирования различных типов информации, структурная схема и взаимодействие основных устройств компьютера. Здесь также в краткой форме описана вычислительная техника, ориентированная на учебный процесс.

1.1. КОМПЬЮТЕР — УНИВЕРСАЛЬНОЕ СРЕДСТВО ОБРАБОТКИ ИНФОРМАЦИИ

Универсальность компьютера как средства обработки информации проявляется двояко: по отношению к информации и по отношению к методам обработки.

Универсальность компьютера по отношению к обрабатываемой информации. Эта универсальность связана с возможностью ее двоичного кодирования. Другими словами, с помощью компьютера можно обрабатывать все, что удается представить в виде последовательности нулей и единиц. Несмотря на такую «простоту» кода, в котором участвуют всего две цифры, с его помощью можно представить информацию различных типов — числовую, символьную, графическую и т. д. Некоторые способы кодирования показаны на рис. 1.1.

Использование двоичного кода позволяет конструировать компьютер из большого количества однотипных элементов, каждый из которых имеет всего два устойчивых состояния. Одно из них соответствует цифре 0, другое — цифре 1.

С двоичным кодированием связано и определение количества информации. Минимальной единицей количества информации считается один разряд двоично-

го кода, который может быть равен 0 или 1. Название такой единицы — **бит** (от англ. binary digit — двоичная цифра). Более крупными единицами являются **байт**, равный восьми битам, **килобайт**, равный

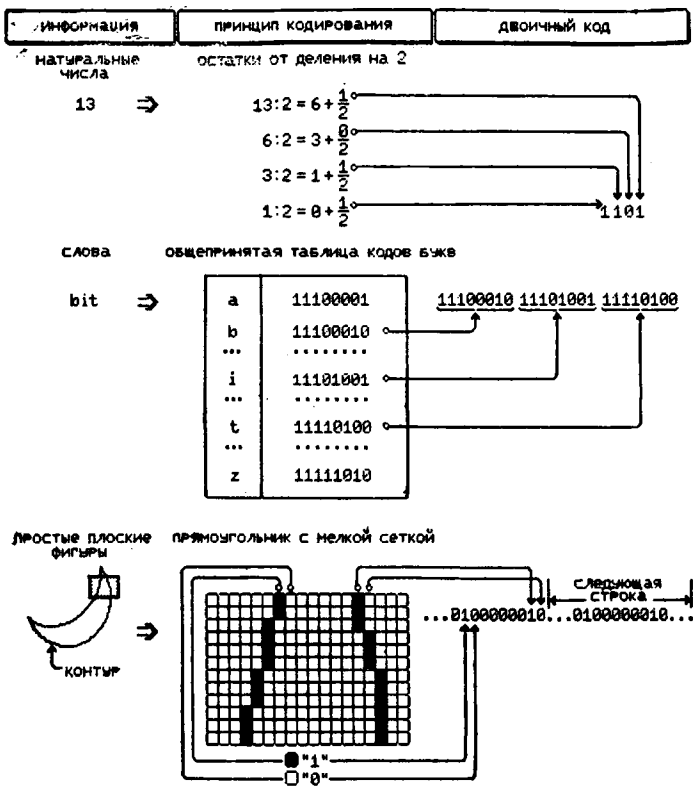


Рис. 1.1. Примеры представления различных типов информации в виде двоичного кода

1024 байт (обозначается К), **мегабайт**, равный 1024 К байт (обозначается М), и, наконец, **гигабайт**, равный 1024 М (обозначается Г).

● Часто битом называют и само физическое устройство, которое служит для представления одного двоичного разряда.

Универсальность компьютера по отношению к методам обработки информации. Эту универсальность

можно пояснить, используя понятие исполнителя. В общем случае под *исполнителем* понимается устройство (например, компьютер), которое может формально выполнять определенный набор операций, не «вникая» в их смысл. В данном случае речь идет об операциях преобразования информации. В связи с двоичным кодированием информации ее преобразование компьютером как исполнителем в конечном счете сводится к операциям с двоичными разрядами. Технически это значит, что компьютер должен по некоторым правилам изменять состояние тех устройств, которые «хранят» обрабатываемую информацию.

Рассмотрим пример исполнителя с условным названием **«Математик»**, который характеризуется таким набором операций: начало работы, восприятие (чтение, ввод) натурального числа, добавление единицы к натуральному числу, воспроизведение (печать, вывод) натурального числа, окончание работы.

С помощью этого исполнителя можно решить, например, задачу определения следующего натурального числа той же четности. Для этого нужно выполнить такую последовательность операций: начало работы, восприятие числа, добавление единицы к (воспринятому) числу, добавление единицы к (полученному) числу, воспроизведение (результирующего) числа, окончание работы.

При этом исполнитель может не иметь представления о том, что такое «следующее натуральное число той же четности». Несмотря на это, *пользователь*, т. е. тот, кто использует этот исполнитель, получит нужный результат.

Компьютер не может выступать в качестве исполнителя «Математик», так как характерные для «Математика» операции не выражены через операции с отдельными разрядами двоичного кода обрабатываемой информации. Перевести команды исполнителя «Математик» в компьютерные команды можно примерно так, как описано ниже. Для операции «добавления» единицы к натуральному числу нужно последовательно осуществить следующие шаги (рис. 1.2):

[1] — установить устройство обработки информации на самый правый (младший) разряд кода числа и перейти к выполнению шага [2];

[2] — «прочитать» значение разряда (0 или 1) и перейти к шагу [3];

[3] — если значение разряда равно 1, то перейти к шагу [4], иначе перейти к шагу [6];

[4] — «записать» в качестве значения разряда 0 и перейти к шагу [5];

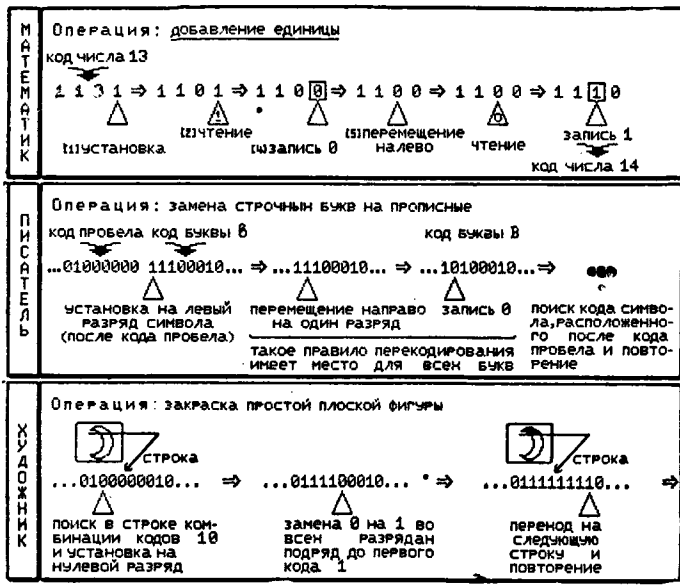


Рис. 1.2. Примеры операций различных исполнителей и их компьютерных реализаций

[5] — переместить устройство обработки информации на один разряд влево и перейти к шагу [2];

[6] — «записать» в качестве значения разряда 1 и перейти к шагу [7];

[7] — закончить выполнение последовательности операций (и перейти к следующей последовательности).

Аналогично представив остальные операции (восприятие и выдачу информации, начало и окончание работы), получим компьютерную реализацию исполнителя «Математик», т. е. его описание с помощью про-

стейших операций (или команд), выполняемых над отдельными разрядами двоичного кода информации.

На рис. 1.2 наряду с операцией «добавления» единицы приведены операции замены строчных букв на прописные (исполнитель «Писатель») и закраски плоской геометрической фигуры (исполнитель «Художник»). Эти операции можно выполнить с помощью таких команд:

«чтение» значения разряда и переход;

«запись» 0 (или 1) в разряд и переход;

«условный» переход;

перемещение обрабатывающего устройства на один разряд налево (направо).

Вообще имеется гипотеза, согласно которой операции любого исполнителя по обработке информации в двоичном коде можно представить в виде последовательности перечисленных команд, т. е. любой исполнитель может быть реализован с помощью компьютера. Именно это и означает универсальность компьютера по отношению к методам обработки информации.

В заключение отметим, что реально компьютер воспринимает от пользователя не элементарные команды, а информацию о более «крупных» операциях (в виде сложения или вычитания чисел), а затем автоматически превращает ее в последовательность элементарных команд (это превращение опять же осуществляется специальной последовательностью элементарных команд). Такая технология замедляет обработку, но значительно облегчает «общение» пользователя с компьютером.

1.2. ОБЩАЯ СТРУКТУРНАЯ СХЕМА КОМПЬЮТЕРА

Компьютер как универсальный исполнитель для обработки различной информации представляет собой совокупность элементов, каждый из которых может находиться в одном из двух устойчивых состояний. Техническая реализация этих элементов может быть различной и здесь не рассматривается. Но структурно их объединяют в разнообразные устройства, выполняющие определенные функции. Функции таких устройств и их взаимодействие представляет описываемая ниже структурная схема компьютера (рис. 1.3).

Оперативное запоминающее устройство (ОЗУ). Одним из главных устройств компьютера является ОЗУ. В нем хранится исходная, промежуточная и окончательная информация, полученная в результате обработки. При этом отдельные элементы ОЗУ (биты) организованы в группы по восемь штук (байты) и соответственно информация пересылается между ОЗУ и другими устройствами целыми байтами (а не отдельными битами). Такая технология пересылки

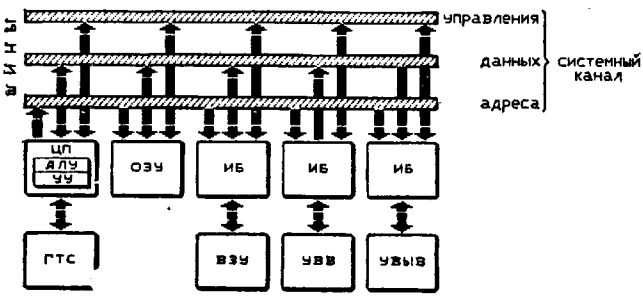


Рис. 1.3. Обобщенная структурная схема компьютера:
АЛУ — арифметико-логическое устройство; УУ — устройство управления

обусловливает и те конкретные способы кодирования различных типов информации, которые рассматривались ранее (например, код символа — восемь двоичных разрядов) и будут рассматриваться далее.

В ОЗУ принято хранить в виде двоичного кода и информацию о последовательности операций, которые необходимо осуществить для получения по исходным данным требуемого результата. Информация такого рода называется **программой.**

Эти операции не являются элементарными командами (см. выше), а представляют собой условные обозначения определенных последовательностей элементарных команд (например, прочитать один или несколько байтов из ОЗУ, записать один или несколько байтов в ОЗУ, сложить два числа и т. п.).

Центральный процессор (ЦП). Он является самым сложным устройством компьютера. Центральный процессор выполняет определенный набор команд и управляет взаимодействием всех устройств компьютера.

Перед выполнением каждой команды он считывает ее из ОЗУ и распознает, а затем заменяет необходимыми элементарными командами и выполняет их. Конечно, перед выполнением команды ЦП считывает необходимую информацию из ОЗУ в специально предназначенные устройства, а после выполнения заносит в них результат и в случае необходимости записывает его в ОЗУ.

Устройства ввода (УВв). Они необходимы для того, чтобы пользователь имел возможность закодировать и разместить в ОЗУ исходную информацию, а также программу для ее обработки. Их может быть несколько различных типов, но такое устройство, как клавиатура, имеется всегда.

Устройства вывода (УВыв). Эти устройства предназначены для декодирования и вывода информации, полученной в результате обработки по программе, на специальный экран (дисплей), на пишущую машинку и т. д.

Внешние запоминающие устройства (ВЗУ). Они обеспечивают неограниченное долгое хранение любой информации (данных, программ) и используются в тех случаях, когда информация не помещается в ОЗУ или прерывается работа пользователя с компьютером. В последнем случае перед окончанием работы информация из ОЗУ переписывается в ВЗУ, а перед возобновлением работы из ВЗУ переписывается обратно в ОЗУ.

Системный канал. Он является последним элементом общей структурной смены и служит для передачи закодированной информации между устройствами в виде электрических импульсов одного из двух уровней — низкого или высокого. Высокий уровень соответствует передаче 1, а низкий — 0.

В состав системного канала входит несколько *шин*, которые физически представляют собой многожильные кабели. По одной из шин (шине данных) передаются обрабатываемая информация и результат; по другой шине (шине адреса) — информация о том, где размещены или должны быть размещены в ОЗУ данные или команды процессора. Наконец, по третьей шине (шине управления) передаются последовательности управляющих импульсов. Тактовая частота этих импульсов, в значительной степени определяю-

щая быстродействие компьютера, задается *генератором тактовых сигналов* (ГТС).

Перечисленные выше устройства подключаются к системному каналу с помощью *интерфейсных блоков* (ИБ) (или просто интерфейсов), которые согласуют скорости циркуляции информации в связываемых каналом устройствах и осуществляют преобразование информации (например, перекодировку) к тому виду, в котором ее воспринимает соответствующее устройство.

1.3. ПРЕДСТАВЛЕНИЕ ЧИСЛОВОЙ И СИМВОЛЬНОЙ ИНФОРМАЦИИ В КОМПЬЮТЕРЕ

Рассмотрим такой принцип кодирования чисел, когда код определяется используемой для кодирования системой счисления, т. е. способом записи и наименования чисел. Общепринято кодирование с применением *позиционных систем счисления.*

В позиционных системах счисления одна и та же цифра имеет различное значение в зависимости от того, какое место (позицию) она занимает в последовательности цифр, изображающих число. Так, например, в общепринятой десятичной системе счисления в последовательности цифр 777,77 одна и та же цифра 7 представляет сотни, десятки, единицы, десятые и сотые доли единицы:

$$777{,}77 = 7\cdot 100 + 7\cdot 10 + 7\cdot 1 + 7\,\frac{1}{10} +$$

$$+\,7\,\frac{1}{100} = 7\cdot 10^2 + 7\cdot 10^1 + 7\cdot 10^0 + 7\cdot 10^{-1} + 7\cdot 10^{-2}.$$

Здесь число 10 является основанием системы счисления, определяемое количеством употребляемых в ней цифр (0, 1, 2, ..., 9). Легко заметить, что в записи числа значение каждой цифры больше значения соседней цифры справа в число раз, равное основанию системы. Это является общим правилом для чисел, представляемых в позиционных системах счисления. Позиции цифр в записи числа называются *разрядами числа.* Если в общем случае основание позиционной системы счисления обозначить через q, то любое n-разрядное число можно представить в развернутом виде следующим образом:

$$a_{n-1}\,a_{n-2}\ldots a_0\,a_{-1}\,a_{-2}\ldots = a_{n-1}\,q^{n-1} + a_{n-2}\,q^{n-2} + \\ + \ldots + a_0\,q^0 + a_{-1}\,q^{-1} + a^{-2}\,q^{-2} + \ldots,$$

где в качестве коэффициентов a_{n-1}, a_{n-2}, ..., a_0, a_{-1}, a_{-2} могут стоять любые из q цифр, используемых в системе счисления: для двоичной системы ($q=2$) — 0, 1; для восьмеричной системы ($q=8$) — 0, 1, 2, 3, 4, 5, 6, 7; для шестнадцатеричной системы ($q=16$) —0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F (A соответствует 10, B — 11, C — 12, D — 13, E — 14, F — 15). Ниже приведены записи десятичного числа $175=1\cdot10^2+7\times\times10^1+5\cdot10^0$ соответственно в шестнадцатеричной, восьмеричной и двоичной системах счисления, а также развернутое изображение в десятичной системе и обычная десятичная запись:

$$1)\ \mathrm{AF}_{16} = 10\cdot16^1 + 15\cdot16^0 = 175_{10};$$

$$2)\ 257_8 = 2\cdot8^2 + 5\cdot8^1 + 7\cdot8^0 = 175_{10};$$

$$3)\ 10101111_2 = 1\cdot2^7 + 0\cdot2^6 + 1\cdot2^5 + 0\cdot2^4 + 1\cdot2^3 + \\ + 1\cdot2^2 + 1\cdot2^1 + 1\cdot2^0 = 175_{10}.$$

Из третьего примера видно, что запись числа в двоичной системе счисления совпадает с той формой представления информации в компьютере, которая приведена на рис. 1.1. Это совпадение не случайно. Переход от записи чисел в двоичной системе к восьмеричной или шестнадцатеричной и обратно производится по очень простым правилам (рис. 1.4 и 1.5).

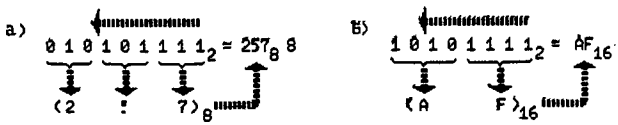


Рис. 1.4. Перевод двоичного числа в восьмеричное (*а*) и шестнадцатеричное (*б*) числа

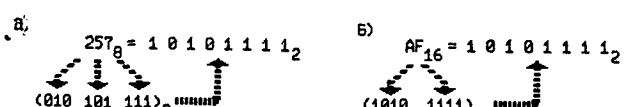


Рис. 1.5. Перевод восьмеричного (*а*) и шестнадцатеричного (*б*) чисел в двоичные числа

Для перехода от двоичной к восьмеричной (или шестнадцатеричной) системе необходимо, двигаясь справа налево, разбить двоичное число групп по три (четыре) разряда соответственно, дополняя при необходимости левую крайнюю группу нулями. Затем каждую группу из трех (четырех) разрядов заменить соответствующей восьмеричной (шестнадцатеричной) цифрой.

Для перевода восьмеричного (шестнадцатеричного) числа в двоичное достаточно заменить каждую цифру указанного числа соответствующим трехразрядным или четырехразрядным двоичным числом. При этом ненужные нули слева отбрасываются.

Запись чисел и кодов в восьмеричной или шестнадцатеричной системах содержит меньше цифр. Это преимущество позволяет облегчить программисту написание программ в кодах машинных команд.

С целью упрощения реализации арифметических операций в компьютере для представления двоичного числа используются прямой, обратный и дополнительный двоичные коды.

Прямой код используется для представления целого двоичного числа в виде

$$X_{пр} = \alpha\, a_{n-1} a_{n-2} \cdots a_1 a_0 ,$$

где α — знаковый разряд (0 — для положительного числа, 1 — для отрицательного числа); a_i — двоичные разряды числа (0 или 1).

Числа X=—11011001; Y=110111001 в прямом коде имеют вид

$$X_{пр} = 111011001;\ Y_{пр} = 011011001.$$

Обратный код используется для представления отрицательных чисел и образуется путем постановки в знаковый разряд единицы и замены во всех других разрядах числа единиц нулями, а нулей единицами. Например, число X=—11011001 в обратном коде будет представлено в виде $X_{обр}$=100100110. Обратный код, если его рассматривать как число, является дополнением модуля исходного числа без знака, помещенного в разрядную сетку:

$$X + X_{обр} = 11011001 + 100100110 = 111111111.$$

Дополнительный код числа получают из обратного путем привлечения единицы к младшему разряду. Например, для числа X=—10100101

$$X_{обр} = 101011010;\ X_{доп} = 101011011.$$

Обратный и дополнительный коды положительного числа совпадают с представлением самого числа.

При сложении и вычитании чисел они обычно представляют-

ся в зависимости от типа арифметико-логического устройства (АЛУ) в обратном или дополнительном коде.

Если в запоминающем устройстве числа представлены в другом коде, то предварительно перед действием алгебраического сложения в АЛУ осуществляется их перевод в необходимый код. После выполнения данной операции результат, если он записывается в оперативную память, снова переводится в код, в котором представлены числа, хранящиеся в запоминающем устройстве.

Сложение чисел в обратных кодах выполняется следующим образом: обратные коды вместе со знаковыми разрядами складываются как двоичные числа, и в случае возникновения единицы переноса из знакового разряда следует прибавить ее к младшему разряду числа. При этом сумма всегда получается в обратном коде. Рассмотрим примеры:

1. Сумма $Z = X + Y$, где $X = 1001101$; $Y = 0011101$.

$X_{обр} = 01001101$

$Y_{обр} = 00011101$

$Z_{обр} = 01101010$

$Z_{пр} = 01101010$

$$\begin{array}{r} 01001101 \\ +00011101 \\ \hline 01101010 \end{array}$$

2. Сумма $Z = X + Y$, где $X = -1011101$; $Y = 0011101$.

$X_{обр} = 10100010$

$Y_{обр} = 00011101$

$Z_{обр} = 10111111$

$Z_{пр} = 11000000$

$$\begin{array}{r} 10100010 \\ +00011101 \\ \hline 10111111 \end{array}$$

3. Сумма $Z = X + Y$, где $X = 1011101$; $Y = -1100101$.

$X_{обр} = 01011101$

$Y_{обр} = 10011010$

$Z_{обр} = 11110111$

$Z_{пр} = 10001000$

$$\begin{array}{r} 11011101 \\ +10011010 \\ \hline 11110111 \end{array}$$

4. Разность $Z=X-Y$, где $X=1101110$; $Y=01001100$.

Операция вычитания заменяется сложением:

$$Z = X_{обр} + (-Y)_{обр} = 01101110 + 11010011.$$

Тогда разность равна:

$$\begin{array}{r} 01101110 \\ +\ 11010011 \\ \hline 101000001 \\ \hline 01000010 \end{array} \quad \begin{array}{l} \\ \\ \\ \\ \end{array} \begin{array}{l} \\ \\ \\ 1 \end{array}$$

$Z_{обр} = 01000010$

$Z_{пр} = 01000010$

Символьная (алфавитно-цифровая) информация в компьютере представляется посредством восьмиразрядных двоичных кодов. Полное число кодовых комбинаций нулей и единиц определяется длиной (разрядностью) кода и составляет $2^8=256$. Каждому символу (цифре, букве, знаку) ставится в соответствие единственный код из числа кодовых комбинаций. С помощью восьмиразрядного кода можно закодировать строчные и прописные буквы латинского алфавита, буквы русского алфавита, цифры, знаки препинания, знаки математических операций и некоторые специальные символы.

Существует много различных систем кодирования на базе восьмиразрядного двоичного кода. В качестве примера в табл. 1.1 приведены кодовые комбинации некоторых символов ASCII (American Standart Code

Таблица 1.1

Символы	Кодовые комбинации									
	Двоичные разряды кода								Шестнадцатеричные разряды кода	
	8	7	6	5	4	3	2	1	2	1
0	0	0	1	1	0	0	0	0	3	0
1	0	0	1	1	0	0	0	1	3	1
2	0	0	1	1	0	0	1	0	3	2
9	0	0	1	1	1	0	0	1	3	9
A	0	1	0	0	0	0	0	1	4	1
B	0	1	0	0	0	0	1	0	4	2
C	0	1	0	0	0	0	1	1	4	3
Z	0	1	0	1	1	0	1	0	5	A
=	0	0	1	1	1	1	0	1	3	D
.	0	0	1	0	1	1	1	0	2	E
?	0	0	1	1	1	1	1	1	3	F
+	0	0	1	0	1	0	1	1	2	B

for Information Interchange — Американский стандартный код для обмена информацией).

Тексты, составленные из этих символов и введенные в компьютер, будут храниться в его памяти в виде последовательности кодовых комбинаций. Например, тексту A=2.1+BC в памяти компьютера будет соответствовать запись: 01000001 00111101 00110010 00101110 00110001 00101011 01000010 01000011.

1.4. ЗАПОМИНАЮЩИЕ УСТРОЙСТВА

В различных функциональных блоках компьютера имеются запоминающие устройства (ЗУ). В этих устройствах в течение некоторого времени либо постоянно хранится обрабатываемая информация, либо информация, с помощью которой осуществляется обработка. Совокупность ЗУ представляет собой *память* компьютера.

Основными операциями в памяти являются занесение информации в память (запись) и выборка информации из памяти (считывание). Обе эти операции называются *обращением к памяти.*

Емкость и быстродействие — важнейшие характеристики отдельных устройств памяти.

Емкость каждого устройства памяти определяется максимальным количеством информации, которое может в ней храниться. Как правило, емкость измеряется в байтах, килобайтах, мегабайтах или гигабайтах.

Быстродействие памяти определяется временем, затрачиваемым на поиск места в памяти, где хранится данная информация, и на ее воспроизведение из памяти (время обращения при считывании), или временем, затрачиваемым на поиск места в памяти, предназначенного для хранения данной информации, и на ее запись в память (время обращения при записи). Обычно эти времена мало отличаются друг от друга.

Как уже отмечалось, для хранения одного разряда двоичного кода (одного бита информации) используются специальные электронные элементы, которые могут находиться в одном из двух устойчивых состояний. Одно из этих состояний интерпретируется как единица (1), другое — как ноль (0). Состояние элемента можно распознать по уровню выходного электрического сигнала. Под воздействием входного сигна-

ла, который в зависимости от его уровня также интерпретируется как 0 или 1, он может переходить из одного устойчивого состояния в другое. Тем самым запоминается соответствующая цифра кода информации, которая передается этим сигналом.

Для хранения многоразрядного двоичного кода используется отдельный набор указанных элементов, который называют *регистром*. Регистры распределены по всей аппаратной части компьютера, выполняя роль ЗУ для временного хранения от одного до нескольких байтов информации.

В состав основной памяти компьютера входит ОЗУ, которое служит для временного хранения больших объемов информации (данных и программ).

Для хранения каждого разряда двоичного кода в ОЗУ, как и в регистрах, используются элементы с двумя устойчивыми состояниями. Обычно эти элементы объединены в группы по восемь и каждая такая группа может хранить один байт информации (иногда сами группы называют байтами).

Объем ОЗУ в современных компьютерах составляет от нескольких десятков килобайт до нескольких мегабайт, а быстродействие исчисляется десятыми долями микросекунды.

Для определения места считывания и записи информации при обращении к ОЗУ каждому байту присвоен определенный номер, называемый *адресом* этого байта. Таким образом, считывание и запись информации здесь осуществляются целыми байтами.

Адреса байтов представляют собой двоичные коды целых неотрицательных чисел. Они используются в различных командах процессора и при выполнении команд временно хранятся в специальных регистрах. При этом, если регистры для хранения адреса имеют n разрядов, то общее количество адресов (т. е. байтов ОЗУ) составляет 2^n. Так, при $n=16$ максимальная емкость памяти составляет 2^{16} байт=64 К; при этом адреса лежат в пределах от 00000_{10} до 65535_{10} или от 0000_{16} до $FFFF_{16}$ (в шестнадцатеричной системе чаще всего и изображаются коды адресов).

Заметим, что имеется возможность объединять отдельные байты в группы по 2, 4, 8 байтов или более. Такие группы называют *ячейками*, а адресом каждой ячейки считается младший адрес входящих в нее бай-

тов. Обращение к ячейкам ОЗУ осуществляется также побайтно. Процедуры такого побайтного обращения управляются процессором, который по коду команды определяет, сколько байтов нужно записать или считать.

В процессе обработки информации осуществляется взаимодействие ОЗУ и процессора. Из ОЗУ в процессор поступают команды и данные, а обратно передаются для хранения результаты выполнения команд. Обращение или выбор соответствующего байта при считывании записи осуществляется с помощью селек-

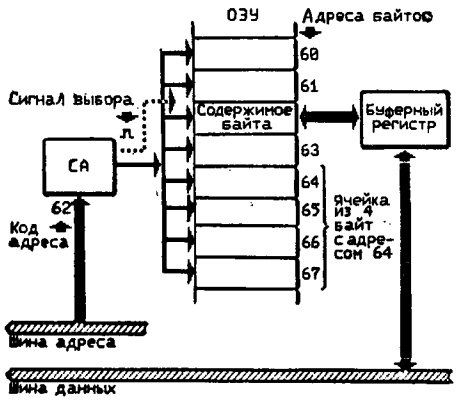


Рис. 1.6. Структурная схема ОЗУ

тора адреса (СА) — устройства, преобразующего код адреса, поступающего по адресным шинам из процессора, в сигнал выбора адресуемого байта (рис. 1.6).

Содержимое выбранного байта при считывании переносится в *буферный регистр*, а затем по *шине данных* передается в процессор. При записи информация с шин данных передается в буферный регистр, а затем в выбранный байт. При этом буферный регистр служит для согласования времени обращения к ОЗУ и скорости передачи информации по шине.

Основная память включает в себя особый вид памяти — *постоянное запоминающее устройство* (ПЗУ), из которого информацию можно только считывать. Такие устройства содержат различные стандартные

программы, обеспечивающие взаимодействие компьютера с пользователем. Постоянные запоминающие устройства с встроенными в него программами производятся на заводе-изготовителе аппаратных средств. Пользователь не имеет возможности изменить содержимое ПЗУ.

Адресный принцип обращения к ОЗУ позволяет осуществлять запись-считывание любого байта. Такое обращение принято называть произвольным доступом к памяти. Однако в компьютере используются

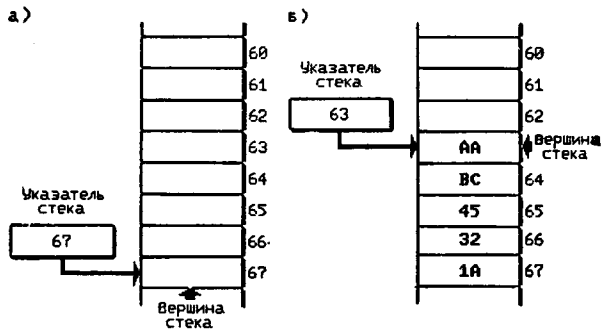


Рис. 1.7. Содержимое стека и указателя стека до (*а*) и после (*б*) записи информации

и устройства памяти, в которых считывание информации осуществляется только в порядке, обратном записи. Такой вид памяти получил название стековой. Эта память может быть реализована в виде нескольких регистров, связанных друг с другом цепями передачи данных, или организована в части ОЗУ. Как правило, используется последнее (рис. 1.7). В этом случае информация запоминается в последовательных байтах (ячейках) памяти, а адрес текущего доступного для чтения байта (вершины стека) находится в специальном регистре процессора — указателя стека. При каждой записи в стеке новой информации производится увеличение (уменьшение) указателя стека, и он указывает на байт, информация в котором была записана последней. При считывании информации из стека производится уменьшение (увеличение) указателя стека. Подробное описание стека приводится в гл. 3.

1.5. СТРУКТУРНАЯ СХЕМА ЦЕНТРАЛЬНОГО ПРОЦЕССОРА. КОМАНДЫ ПРОЦЕССОРА И ИХ ВЫПОЛНЕНИЕ

Структурную схему ЦП рассмотрим на примере микропроцессора. Микропроцессор выполнен на одном кристалле кремния в виде одной интегральной микросхемы, в которой реализованы все основные функции процессора. Компьютеры, построенные на микропроцессорной базе, получили название микрокомпьютеров.

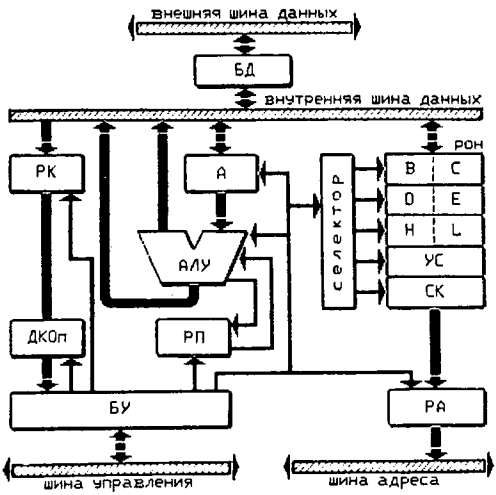


Рис. 1.8. Структурная схема восьмиразрядного микропроцессора

На рис. 1.8 приведена обобщенная структурная схема восьмиразрядного микропроцессора, который обрабатывает информацию последовательно по одному байту. Существуют микропроцессоры, обрабатывающие сразу 2 или 4 байта информации. Их называют соответственно 16- и 32-разрядными микропроцессорами.

Необходимыми составляющими любого микропроцессора являются устройство управления, ряд рабочих регистров и арифметико-логическое устройство. Обмен информацией между ними осуществляется по внут-

ренним шинам. Микропроцессоры могут включать в себя и другие компоненты, расширяющие их функциональные возможности.

Устройство управления (УУ). К основным функциям устройства управления относятся выборка команд программы из основной памяти, их расшифровка и исполнение. Оно состоит из блоков управления и следующих регистров: счетчика команд, регистра команд, указателя стека и регистра признаков.

Счетчик команд (СК), называемый также программным счетчиком, служит для образования и хранения адресов, указывающих на размещение команд, выбираемых из основной памяти. Команды имеют различную длину и могут занимать от одного до нескольких байтов памяти. При каждой передаче байта из памяти в процессор блок управления увеличивает содержимое счетчика команд на единицу. После считывания всех байтов команды счетчик команд будет содержать адрес первого байта следующей команды. Перед исполнением программы в счетчик команд помещается адрес первого байта первой команды программы.

Регистр команд (РК) принимает и хранит первый байт выбираемой из памяти команды на время расшифровки ее кода операции (КОп). Обычно под код операции в команде отводится восемь двоичных разрядов, что позволяет реализовать до $2^8=256$ различных кодов операций. Независимо от длины команды код операций всегда размещается в первом байте команды. Расшифровка первого байта команды производится дешифратором кода операций (ДКОп). В результате расшифровки определяется вид операции и количество байтов в команде. Под воздействием этой информации блок управления (БУ) вырабатывает последовательность управляющих сигналов, обеспечивающих выполнение этой операции.

Указатель стека (УС) представляет собой 16-разрядный регистр, служащий для формирования адреса стековой памяти.

Регистр признаков (РП) используется для фиксации определенных ситуаций, возникающих в результате выполнения операции. Код ситуации называется *признаком результата.* Признак может прини-

мать значение 0 или 1. Отметим некоторые из возможных признаков.

Признак переноса указывает на необходимость переноса единицы при сложении двух восьмиразрядных чисел в младший значащий разряд следующего байта. Признак нуля фиксирует появление нулевого результата операции, признак знака — знак результата операции. Эти и другие признаки используются для изменения хода обработки информации.

Рабочие регистры. Они служат для временного хранения информации. Различают два вида рабочих регистров: регистры адреса и арифметические регистры. Регистры адреса используются для адресации данных и команд в ОЗУ, арифметические регистры — для временного хранения обрабатываемой информации.

Рабочие регистры часто называют *регистрами общего назначения* (РОН), а специальный регистр, используемый для хранения результатов, полученных в арифметико-логическом устройстве (АЛУ), — *аккумулятором* (А).

Регистры общего назначения и аккумулятор обеспечивают самый быстрый доступ к хранящейся в них информации, что положительно сказывается на скорости выполнения операций и, следовательно, на быстродействии компьютера в целом. Для адресации РОН и А в коде команде отводится три двоичных разряда. Поэтому можно адресоваться к $2^3=8$ регистрам. Они, как правило, обозначаются буквами B, C, D, E, H и L. Регистры B и C, D и E, H и L объединены попарно, что позволяет обрабатывать как 1 байт, так и 2 байта информации сразу. Через регистровые пары обращаются к ячейкам основной памяти. В некоторых типах микропроцессоров для этих целей используется только регистровая пара H, L. Поэтому прежде чем обратиться к нужной ячейке памяти, необходимо в эту регистровую пару занести адрес этой ячейки.

Арифметико-логическое устройство. Оно предназначено для выполнения арифметических и логических операций над данными, однако само АЛУ способно выполнить лишь ограниченный набор этих операций. Этот набор может быть различным и зависит от типа микропроцессора. Как правило, АЛУ может выполнять арифметические операции сложения, вычитания и поразрядного сдвига чисел, а также некоторые логические операции, употребляемые для поразрядного сравнения двух- и восьмиразрядных цифровых кодов. Другие арифметические и логические

операции могут быть запрограммированы на базе операций, выполняемых АЛУ.

Арифметико-логическое устройство выполняет указанные операции либо над содержимым аккумулятора и одного из РОН, либо над содержимым аккумулятора и ячейки памяти, адрес которой содержится в одной из регистровых пар, например в H, L. В некоторых операциях АЛУ также используется значение какого-либо признака из регистра признака. Результат операции всегда фиксируется в аккумуляторе.

Для временного хранения сформированного процессором адреса памяти используется *регистр адреса* (РА). По этому адресу записывается или считывается байт данных или байт команды. Для передачи адреса по адресным шинам в память, его преобразования в сигнал выбора ячейки памяти требуется некоторое время, называемое временем доступа к памяти. В течение этого времени адреса памяти хранятся в регистре адреса.

Буфер данных (БД) служит для организации обмена информацией между внутренней и внешней шинами данных.

Рассмотрим полный цикл работы ЦП при выполнении команды.

Выборка каждого байта команды производится в два этапа. На первом этапе адрес байта команды, содержащейся в СК, передается в РА, а затем по адресным шинам в устройство основной памяти, где происходит выбор адресуемого байта. На втором этапе — передача содержимого адресуемого байта по шинам данных в процессор и увеличение содержимого СК на единицу. При этом, если выбирается не последний байт команды, то новое содержимое СК будет представлять адрес следующего байта текущей команды, а если выбирается последний байт, то адрес первого байта следующей команды.

При выборке первого байта команды его содержимое передается и фиксируется в РК. Дешифратор кода операций (ДКОп) по содержимому регистра команд, поступающему на его вход, выявляет вид операции, число байтов в команде, а также способ адресации, используемый в команде. На базе этой информации блок управления задает необходимую последовательность управляющих воздействий, приводящих к выборке следующих байтов команды или к использованию команды.

В качестве примера опишем процесс выборки и исполнения команды, по которой содержимое регистра общего назначения суммируется с содержимым аккумулятора А, а результат операции фиксируется в аккумуляторе. Эта команда однобайтовая с прямой регистровой адресацией. В коде команды $10000001_2 = 81_{16}$ первые пять старших разрядов образуют код операции; три последних — код адреса регистра С. Пусть до выборки и ис-

полнения команды содержимое СК составляет 0176_{16}, а содержимое регистра С и аккумулятора соответственно равны $2F_{16}$ и $3A_{16}$.

Выборка команды начинается с передачи адреса команды 0176_{16} в РА, а затем в устройство памяти для поиска ячейки памяти с этим адресом. После этого содержимое адресуемой ячейки 10000001_2 передается и записывается в РК. По коду операций и адресной части команды ДКОп распознает, что выборка команды закончена и над содержимым аккумулятора и регистра С необходимо произвести операцию суммирования. Затем блок управления вырабатывает последовательность сигналов, под воздействием которых содержимое аккумулятора $3A_{16}$ и регистра $2F_{16}$ поступает на входы АЛУ. Арифметико-логического устройство производит операцию суммирования над поступившими данными. Результат операции $2F_{16}+3A_{16}=69_{16}=01101001_2$ переносится в аккумулятор, замещая в нем прежнее данное $3A_{16}$. На этом выполнение команды заканчивается. Начинается выборка следующей команды с адресом 0177_{16}, образованным в СК при завершении выборки предыдущей команды.

1.6. ПЕРИФЕРИЙНЫЕ УСТРОЙСТВА КОМПЬЮТЕРА

Периферийные (внешние) устройства используются для связи пользователя или объекта управления с компьютером. Связь пользователя с компьютером обеспечивается с помощью дисплеев, перфоленточных и перфокарточных устройств, печатающих и внешних запоминающих устройств (ВЗУ), клавиатуры и т. п.

Дисплей. Он представляет собой телевизионный монитор или панель на жидких кристаллах, который служит для отображения выводимой информации из компьютера. Их подразделяют на символьные и графические. На экране символьного дисплея может отображаться только алфавитно-цифровая информация; на экране графического дисплея — как символьная, так и графическая информация в виде схем, графиков, рисунков и чертежей.

Как правило, дисплей используется и при вводе с клавиатурой. В этом случае пользователь может вести визуальное наблюдение на экране дисплея за вводимой в компьютер информацией и при необходимости производить ее редактирование.

Перфоленточные и перфокарточные устройства. Они используются для ввода и вывода информации. Характерным для них является наличие промежуточного носителя информации.

В устройствах перфоленточного вво-

да-вывода в качестве промежуточного носителя информации используется перфолента — узкая и длинная полоска бумаги, на которой информация кодируется пробивками круглых отверстий. Один символ на перфоленте кодируется набором пробивок, расположенных в одной строке в поперечном направлении ленты. При вводе информации с этого устройства в компьютер пробивки на движущейся бумажной ленте преобразуются в электрические сигналы; при выводе информации, наоборот, электрические сигналы преобразуются в пробивки на ленте.

В работу устройств перфокарточного ввода-вывода положены аналогичные принципы, только в качестве промежуточного носителя в них служит перфокарта. Она представляет собой картонный прямоугольник со срезанным верхним углом, разбитый по ширине на 80 колонок.

Каждая цифра, буква или другой символ кодируется на перфокарте в одной колонке путем пробивки прямоугольных отверстий. Отдельную перфокарту можно представить в виде вырезанной части перфоленты, а весь набор (колоду) перфокарт — в виде всей перфоленты. Запись и считывание информации с перфокарт осуществляются в порядке их расположения в колоде.

Перфоленты и перфокарты с программами и исходными данными подготавливаются на специальных устройствах — ленточных и карточных перфораторах.

В некоторых случаях в качестве устройства ввода-вывода используются телетайпы, объединяющие клавиатуру, устройство считывания с перфолент, ленточный перфоратор и печатающее устройство. Скорость приема — передачи этих устройств невелика и не превышает 10 символов.

Печатающие устройства. Эти устройства, называемые принтерами, служат для вывода программ, данных и результатов обработки на широкую бумажную ленту или отдельные листы. Существуют принтеры с посимвольным выводом и построчной печатью. Построчные принтеры имеют большую скорость, так как в них строка печатается сразу полностью, а не символ за символом. В построчных принтерах скорость печати может достигать до 1500 строк/мин.

Для вывода графической информации использу-

ются регистрирующие устройства — *графопостроители.* Они преобразуют цифровую информацию в графическую, выводимую в виде графиков и рисунков на бумаге.

Внешние запоминающие устройства. В качестве ВЗУ в компьютерах используются накопители на магнитных лентах, гибких и жестких магнитных дисках. Поверхностный слой носителей информации образует магнитное покрытие. Запись информации на это покрытие производится с помощью специальной головки, создающей на нем участки с различным направлением намагниченности, как это делается, например, в бытовых магнитофонах. Участок, намагниченный в одном направлении, соответствует единице; в другом направлении — нулю. Информация на магнитных лентах записывается на 7 или 9 дорожках, подобно тому, как это делается на перфолентах. Запись и считывание информации производятся при движении ленты относительно магнитных головок. При выборке необходимой информации приходится просматривать все записи на ленте от ее начала до места расположения нужной записи. Поэтому ВЗУ на магнитных лентах относятся к устройствам памяти с *последовательным доступом* к информации.

На гибких и жестких магнитных дисках запись информации производится вдоль концентрических окружностей на их поверхности, называемых *дорожками.* Число дорожек может быть различным. Они нумеруются, начиная от края к центру диска. Поверхность диска условно делится на сектора, что позволяет запись информации определить по номеру сектора и номеру дорожки. Выбор нужной записи на диске осуществляется путем позиционирования магнитной головки и вращения диска. Если, например, необходимо отыскать запись на 20-й дорожке в 5-м секторе, то для этого требуется подвести головку к дорожке с этим номером и с заданного сектора произвести считывание (запись) информации. Этот способ доступа к информации называется прямым.

Для увеличения емкости ЗУ на жестких дисках они объединяются по нескольку штук. Емкость памяти дисковых пакетов измеряется десятками мегабайт.

Запоминающие устройства на магнитных лентах обладают большей емкостью памяти, чем устройства

на гибких и жестких магнитных дисках. Однако это преимущество не всегда можно использовать из-за большого времени доступа к информации, особенно в тех случаях, когда в процессе решения задачи предусматривается многократное обращение к внешним устройствам памяти. Здесь несравненное преимущество имеют устройства памяти с прямым доступом, так как поиск нужной единицы информации на гибком или жестком диске осуществляется значительно быстрее, чем на магнитной ленте.

1.7. ПЕРСОНАЛЬНЫЕ КОМПЬЮТЕРЫ

Развитие микроэлектронной технологии способствовало созданию более совершенных компьютеров, обладающих небольшими габаритными размерами, сравнительно малой стоимостью, незначительным потреблением электроэнергии и повышенной надежностью эксплуатации.

В начале 70-х годов появились первые микропроцессоры, выполненные в виде одной или нескольких больших интегральных схем (БИС). На базе микропроцессоров стали создаваться микрокомпьютеры различного назначения. В дальнейшем из них выделился отдельный класс машин для индивидуального пользования — персональные компьютеры. Строгого определения персонального компьютера не существует. Однако можно выделить набор свойств, которым в настоящее время должен обладать персональный компьютер:

1) минимальная стоимость, доступная для индивидуального пользователя;

2) аппаратное обеспечение, включающее микропроцессор и оперативную память, клавиатуру и телевизионный монитор, периферийную память в виде небольших накопителей на жестких магнитных дисках (НМД) или на гибких магнитных дисках (НГМД) и печатающие устройства;

3) программное обеспечение, ориентированное на массового пользователя;

4) конструктивное исполнение, позволяющее максимально приблизить компьютер к рабочему месту пользователя, к источникам обрабатываемой информации.

Персональные компьютеры имеют различное быстродействие, объем ОЗУ и состав периферийного оборудования. Поэтому одни из них могут широко использоваться в профессиональной деятельности, другие — для обучения учащихся школ, ПТУ, техникумов и студентов вузов; третьи — для разнообразного бытового применения.

В настоящее время персональные компьютеры удовлетворяют потребности не только индивидуальных пользователей, но малых и больших организаций, где они используются для оснащения автоматизированных рабочих мест (АРМ) конструктора, технолога, преподавателя и учащегося.

Отечественными представителями персональных компьютеров являются микрокомпьютеры типа «Электроника БК-0010», «Электроника-85», «Искра-1030», ЕС 1841, ЕС 1842, диалоговые вычислительные комплексы (ДВК) и др.

1.8. ЛОКАЛЬНЫЕ ВЫЧИСЛИТЕЛЬНЫЕ СЕТИ И КОМПЛЕКТЫ УЧЕБНОЙ ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

Локальные вычислительные сети (ЛВС). Для оперативного использования информации, хранящейся в других, удаленных от пользователя компьютерах, они объединяются в вычислительные сети в пределах небольших территорий (учреждения, учебного заведения, учебного класса). Такие сети называются *локальными вычислительными сетями.* Связь между компьютерами может быть организована по-разному или, как говорят, иметь различную топологию: радиальную, магистральную и кольцевую. При радиальной топологии обмен информацией между каждой парой компьютеров осуществляется через центральный компьютер. При магистральной топологии предусматривается обмен информацией между любыми двумя компьютерами путем их временного подключения к общему каналу передачи данных. При кольцевой топологии обмен информацией обеспечивается путем однонаправленной передачи информации от компьютера-источника к компьютеру-приемнику. В том случае, если компьютер-приемник не готов к приему информации, то, обойдя кольцо, она возвращается к нему и поглощается им же.

С помощью средств ЛВС пользователь со своего компьютера может обращаться к другим компьютерам сети за данными, программами и использовать их для решения задач.

Локальные вычислительные сети при оснащении их соответствующим оборудованием и программным обеспечением могут иметь доступ к сетям, находящимся на другом, более высоком уровне иерархии. Для согласованного взаимодействия компьютеров ЛВС разрабатывается комплекс процедур-протоколов, определяющих некоторый набор правил, которыми следует руководствоваться при организации передачи сообщений.

Комплекты учебной вычислительной техники (КУВТ). Примером объединения компьютеров в ЛВС могут служить *комплекты учебной вычислительной техники* (КУВТ). Они представляют собой совокупность аппаратных и программных средств, ориентированных на использование в учебном процессе общеобразовательных школ, средних ПТУ, межшкольных учебных комбинатов, педагогических и других учебных заведений.

Комплект учебной вычислительной техники диалоговой вычислительной системы, управляемой посредством программ, размещенных в памяти компьютеров, входящих в состав рабочих мест учащихся (РМУ) и рабочего места преподавателя (РМП), содержит двенадцать РМУ и одно РМП.

В состав РМУ входит компьютер, телевизионный монитор и клавиатура. В компьютере РМУ предусмотрена возможность подключения кассетного магнитофона.

В состав РМП кроме указанных устройств входят накопитель на гибком или жестком магнитном диске и малогабаритное печатающее устройство — принтер. С помощью принтера необходимая информация из компьютера преподавателя может быть выведена на бумажный носитель информации.

Функционирование КУВТ обеспечивается за счет программ, входящих в состав РМУ и РМП. Они предоставляют учащемуся и преподавателю следующие возможности:

ввод-вывод программ и данных с клавиатуры РМУ и РМП;

исполнение программ на РМУ и РМП;

обращение к локальной сети с РМУ и РМП;

вывод информации на печатающее устройство РМП;

ввод-вывод информации на кассетный магнитофон РМУ;

ввод-вывод информации на накопитель на жестком и гибком диске РМП, в том числе программ и данных с РМУ;

загрузку программ и данных с РМП одновременно на все РМУ;

загрузку программ и данных с РМП на одно или группу РМУ;

загрузку программ и данных с РМУ на РМП;

обслуживание запросов с РМУ на взаимодействие с РМП.

В настоящее время во многих учебных заведениях нашей страны в учебном процессе широко используются вычислительные комплекты как зарубежного, так и отечественного производства. К ним следует отнести японский КУВТ ЯМАХА; болгарский ПРАВЕЦ; отечественные комплекты — КУВТ-86, КОРВЕТ, АГАТ, УКНЦ и др.

УПРАЖНЕНИЯ

1.1. Переведите двоичное число $X=111101101_2$ в восьмеричное, десятичное и шестнадцатеричное.

1.2. Переведите шестнадцатеричное число $X=FA_{16}$ в десятичное, восьмеричное и двоичное.

1.3. Определите, какое количество символов можно закодировать с помощью четырехразрядного двоичного кода.

1.4. Используя таблицу кодов ASCII (см. рис. 1.6), закодируйте слово BAZ в двоичной и шестнадцатеричной формах.

1.5. Определите количество адресуемых байтов памяти и запишите адрес последней ячейки, если для адресации памяти используется восьмиразрядный двоичный код.

1.6. Определите десятичный номер байта памяти, если он задан шестнадцатеричным адресом FF_{16}.

1.7. Запишите содержимое счетчика команд после выборки из памяти двух трехбайтовых команд, если перед этим его содержимое было равным $1AFO_{16}$.

1.8. Определите содержимое регистра команд после выборки из памяти команды с кодом $10000101\ 100000001_2$.

1.9. Содержимое аккумулятора и содержимое адресуемого байта памяти соответственно равны $2A_{16}$ и 16_{16}. Определите содержимое аккумулятора после выполнения команды, по которой суммируется содержимое аккумулятора и указанного байта памяти.

Глава 2

ТЕХНОЛОГИЯ РЕШЕНИЯ ЗАДАЧ С ИСПОЛЬЗОВАНИЕМ КОМПЬЮТЕРА

В этой главе последовательно рассматриваются все этапы, из которых состоит процесс решения задач с использованием компьютера.

2.1. ЭТАПЫ РЕШЕНИЯ ЗАДАЧ

Следствием развития новых информационных технологий является чрезвычайно быстрое расширение области применения компьютеров для решения разнообразных задач, связанных с обработкой информации различного характера. Мы рассмотрим лишь три вида информации и соответственно три класса задач, для решения которых используются компьютеры:

1. Вычислительные задачи, связанные с обработкой числовой информации. К ним относится, например, задача решения системы линейных уравнений большой размерности.

2. Задачи по обработке символьной информации, связанные с созданием и редактированием текстов. С решением таких задач связан труд секретаря-машинистки.

3. Задачи по обработке графической информации, т. е. схем, чертежей, графиков, эскизов и т. д. К таким задачам относится, например, задача разработки конструктором чертежей новых изделий.

Процесс решения задач с использованием компьютера можно условно разбить на ряд этапов:

1. Постановка задачи (описание результатов и необходимых исходных данных).

2. Построение модели (описание соотношений между исходными данными и результатами).

3. Разработка или выбор готового метода решения задачи с помощью компьютера (представление процесса определения решения в виде последовательности операций, которые пользователь может реализовать на компьютере).

4. Реализация метода на компьютере (представление найденной последовательности операций в специальном виде и их выполнение).

5. Анализ полученных результатов (сравнение полученных результатов с предполагаемыми и, в случае расхождений, установление их причин).

В разных задачах трудоемкость тех или иных этапов может быть весьма различной. В некоторых случаях отдельные этапы настолько просты в выполнении, что они просто незаметны. Рассмотрим более подробно все этапы на примере наиболее подходящей для этого вычислительной задачи.

2.2. ПРИМЕР ВЫЧИСЛИТЕЛЬНОЙ ЗАДАЧИ

Постановка задачи. Участок заготовительного цеха, специализирующийся на раскрое длинномерных заготовок и перешедший на полный хозрасчет, выпускает три различных продукта (продукт 1, продукт 2, продукт 3), каждый из которых представляет собой стальной пруток определенной длины. Для этого закупаются заготовки двух различных размеров (заготовка 1, заготовка 2). При этом объемы продуктов *1*, *2* и *3*, которые можно получить из 1 т заготовок 1, отличаются от объемов этих же продуктов, получаемых из 1 т заготовок 2. Соответствующие показатели приведены в табл. 2.1.

Таблица 2.1. **Выход продукции на единицу веса заготовок**

Вид продукта	Выход продуктов из 1 т заготовок 1	Выход продуктов из 1 т заготовок 2
1	0,2	0,3
2	0,2	0,1
3	0,3	0,3

Из таблицы видно, что из 1 т заготовок *1* можно изготовить 0,2 т продукта *1*, 0,2 т продукта *2* и 0,3 т продукта *3*; остальные 0,3 т составляют отходы. Из заготовок *2* выход продукта *3* и количество отходов те же, выход продукта *1* более высокий — 0,3 т, а продукта *2* более низкий — 0,1 т, чем из заготовок *1*.

Определить, какое количество заготовок требуется, чтобы обеспечить работу цеха в течение одного дня.

Таким образом, требуется найти две неизвестные

величины x_1 и x_2, обозначающие требуемые количества заготовок *1* и *2* соответственно. Теперь, поняв, что именно мы намерены извлечь из вычислений, зададимся вопросом «Что нам известно?». Исходные данные представлены в табл. 2.1. Но содержат ли они все, что нам нужно для вычисления x_1 и x_2? Ответ отрицательный, так как для перешедшего на полный хозрасчет участка величины x_1 и x_2 зависят еще от прибыли, получаемой этим участком в случае использования заготовок *1* и *2*.

Полная прибыль участка при использовании только 1 т заготовок 1 складывается из выручки цеха в результате продажи 0,2 т продукта *1*; 0,2 т продукта *2* и 0,3 т продукта *3*. Аналогично вычисляется прибыль участка, получаемая за счет использования только 1 т заготовок 2. Предположим, что прибыль от использования 1 т заготовок 1 равна 500 руб., а от использования 1 т заготовок 2 составляет 600 руб. Из этого, однако, вовсе не следует, что участок должен использовать только заготовки 2, дающие более высокую прибыль, так как величины x_1 и x_2 зависят еще и от ограничений на производственные возможности самого участка. Для простоты будем считать, что в течение одного дня продукт 1 не может выпускаться участком в количестве, превышающем 1,8 т, продукт 2 — в количестве, превышающем 1,2 т, а продукт 3 — в количестве, превышающем 2,4 т.

Включив эти ограничения в исходные данные, предположим, что теперь они отображают все известные нам существенные факторы, которые могут повлиять на величины x_1 и x_2.

Заметим также, что по смыслу задачи величины x_1 и x_2 должны быть неотрицательны.

Таким образом, опустив все подробности решаемой задачи и выделив факторы, существенно влияющие на ее решение, можно переходить к следующему этапу.

Построение модели. Всякий раз, когда приступают к изучению сложного реального объекта, приходится сосредоточивать внимание лишь на его существенных факторах, опуская второстепенные. Например, при постановке задачи обеспечения работы заготовительного цеха не принимались во внимание такие факторы, как обеспеченность цеха квалифицированными кадра-

ми, затраты на заработную плату рабочих и пр. Таким образом, сложный реальный объект подменяют некоторым упрощенным, легче поддающимся изучению. Именно таким упрощенным объектом, приближенно отображающим действительность, в вычислительных задачах является любая математическая модель, в качестве которой могут служить функция, система уравнений, система неравенств и т. п. Построить математическую модель производства на примере рассматриваемого участка заготовительного цеха — это значит представить в математической форме все его существенные свойства, выделенные при постановке задачи.

Пусть по-прежнему величины x_1 и x_2 обозначают количество заготовок 1 и 2, необходимых для работы участка в течение одного дня. Тогда количества P_1, P_2 и P_3 произведенных из этих заготовок продуктов 1, 2 и 3 равны соответственно

$$P_1 = 0{,}2x_1 + 0{,}3x_2; \quad P_2 = 0{,}2x_1 + 0{,}1x_2;$$
$$P_3 = 0{,}3x_1 + 0{,}3x_2. \qquad (2.1)$$

Следовательно, выявленные на этапе постановки задачи ограничения на производственные возможности участка можно записать в виде неравенств для P_1, P_2 и P_3:

$$0{,}2x_1 + 0{,}3x_2 \leqslant 1{,}8; \quad 0{,}2x_1 + 0{,}1x_2 \leqslant 1{,}2;$$
$$0{,}3x_1 + 0{,}3x_2 \leqslant 2{,}4. \qquad (2.2)$$

Установленные на первом этапе условия неотрицательности величин x_1 и x_2 записываются в виде неравенств

$$x_1 \geqslant 0, \quad x_2 \geqslant 0. \qquad (2.3)$$

Для наглядности интерпретируем задачу геометрически (рис. 2.1, *а* — *г*). При этом точки плоскости, координаты которых x_1 и x_2 удовлетворяют условиям (2.2) и (2.3), представлены заштрихованной областью. Утолщенные прямые линии состоят из точек, координаты которых x_1 и x_2 удовлетворяют неравенствам (2.3) и уравнениям

$$0{,}2x_1 + 0{,}3x_2 = 1{,}8; \quad 0{,}2x_1 + 0{,}1x_2 = 1{,}2;$$
$$0{,}3x_1 + 0{,}3x_2 = 2{,}4.$$

Стрелки, проведенные от каждой из этих линий, направлены внутрь областей, где выполняются соответствующие неравенства (2.2), а сами линии служат границами этих областей.

Согласно постановке задачи при использовании x_1 т заготовок 1 и x_2 т заготовок 2 суммарная прибыль W (руб./день) составит

$$W = 500x_1 + 600x_2. \qquad (2.4)$$

Для хозрасчетного участка наилучшими являются такие величины x_1 и x_2, при которых суммарная прибыль (2.4) максимальна при выполнении условий (2.2) и (2.3). Таким образом, задача сводится к мак-

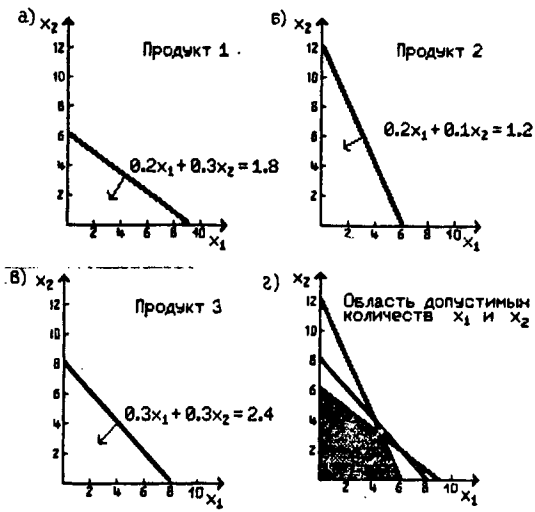


Рис. 2.1. Допустимые количества заготовок

симизации функции W при наличии ограничений (2.2) и (2.3). Эта задача называется *задачей линейного программирования*. На практике эти задачи могут включать в себя до нескольких сотен ограничений, каждое из которых содержит не две, а несколько тысяч переменных. Поэтому при построении (вообще следует упускать из виду, что чрезмерное стремление говоря, всякой) математической модели никогда не учесть как можно больше свойств реального объекта

на этапе постановки задачи может привести к такой сложной модели, которая окажется недоступной для решения даже с помощью современных компьютеров. С другой стороны, если математическая модель слишком упрощена, то она может стать совсем неадекватной реальному объекту, т. е. непригодной для его изучения. Так, если мы упростим получившуюся модель производства на участке заготовительного цеха, исключив из формул (2.2) два первых неравенства, то «наилучшими» количествами x_1 и x_2 будут $x_1=0$ и $x_2=8$, при которых будет произведено

$$P_3 = 0{,}3x_1 + 0{,}3x_2 = 0{,}3 \cdot 0 + 0{,}3 \cdot 8 = 2{,}4 \text{ т}$$

продукта 3 и получена максимальная суммарная прибыль

$$W_{\max} = 500x_1 + 600x_2 = 500 \cdot 0 + 100 \cdot 8 = 4800 \text{ руб.}$$

Однако, согласно формулам (2.1), при таких значениях x_1 и x_2 количество произведенного продукта 1 должно быть равно

$$P_1 = 0{,}2x_1 + 0{,}3x_2 = 0{,}2 \cdot 0 + 0{,}3 \cdot 8 = 2{,}4 \text{ т.}$$

Это количество заведомо превышает возможный для данного участка выход продукта 1, равный 1,8 т. Следовательно, если совсем не использовать заготовки 1 ($x_1=0$ т), а перерабатывать $x_2=8$ т заготовок 2, то часть заготовок на самом деле останется к концу рабочего дня непереработанной, и прибыль окажется значительно меньше 4800 руб. Значит, исключив из формул (2.2) два первых неравенства, получим математическую модель, не соответствующую возможностям реального производства.

Вернувшись снова к математической модели, выраженной неравенствами (2.2) и (2.3), ответим на важный вопрос: «Нельзя ли упростить модель без ущерба для решения задачи?»

Сложив первое и второе равенства (2.2), получим неравенство

$$(0{,}2x_1 + 0{,}3x_2) + (0{,}2x_1 + 0{,}1x_2) \leqslant 1{,}8 + 1{,}2,$$

т. е.

$$0{,}4x_1 + 0{,}4x_2 \leqslant 3.$$

Умножив обе части этого неравенства на 0,75, получим

$$0{,}3x_1 + 0{,}3x_2 \leqslant 2{,}25. \qquad (2.5)$$

Таким образом, из двух первых неравенств (2.2) следует неравенство (2.5), обозначающее, согласно последнему из неравенств (2.1), что продукт 3 не может выпускаться участком в количестве, превышающем 2,25 т/день. Тогда из двух первых неравенств (2.2) следует, что продукт 3 не может выпускаться участком в количестве, превышающем 2,4 т/день, т. е. из справедливости двух первых неравенств (2.2) неизбежно следует справедливость третьего неравенства:

$$0{,}3x_1 + 0{,}3x_2 \leqslant 2{,}4,$$

которое, таким образом, можно без ущерба удалить из математической модели как несущественное.

Заштрихованная область допустимых закупок (см. рис. 2.1) целиком расположена под прямой с уравнением

$$0{,}3x_1 + 0{,}3x_2 = 2{,}4.$$

Удалив третье из неравенств (2.2), мы перейдем к другой, более простой математической модели, столь же соответствующей реальному производству, как и исходная модель (2.2), (2.3).

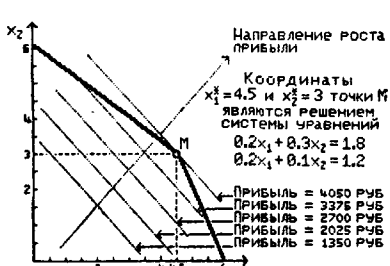


Рис. 2.2. Суммарная прибыль

Разработка или выбор готового метода решения задачи. В рассматриваемом примере решение задачи о рациональной организации работы участка может быть найдено графически.

Каждая из множества параллельных прямых, изображенных на рис. 2.2, соответствует различным комбинациям значений x_1 и x_2, приводящим к одному и тому же значению прибыли, вычисляемой по формуле $W=500x_1+600x_2$. На самой верхней из этих прямых в точке M с координатами $x_1^*=4{,}5$, $x_2^*=3$,

лежащей в области допустимых с точки зрения математической модели значений x_1 и x_2, достигается максимальное значение прибыли 4050 руб. Следовательно, значения $x_1=4{,}5$, $x_2=3$ являются искомым решением задачи.

Из рис. 2.2 видно, что не существует никаких других количеств x_1 и x_2 заготовок, допустимых с точки зрения математической модели и дающих прибыль не менее 4050 руб.

Реализация метода на компьютере. Если число переменных в задачах линейного программирования больше двух, то графически решить задачу с высокой точностью не представляется возможным. Поэтому для задач этого типа разработаны эффективные методы решения, которые сводятся к выполнению последовательности арифметических действий с исходными данными. Для них составляются соответствующие программы.

Как отмечалось в гл. 1, компьютер может хранить в своей памяти лишь ограниченное количество десятичных цифр числа. Это неизбежно приводит к округлению тех чисел в памяти компьютера, у которых количество цифр превышает максимальный предел, определенный техническими характеристиками компьютера. Между тем все алгоритмы решения задач линейного программирования ориентированы на применение компьютера, поэтому важным является влияние ошибок округления для этих алгоритмов. Когда в длинных цепочках вычислений последующие вычисления все время опираются на результаты предыдущих, ошибки округления могут расти до такой степени, что результат вычислений не будет иметь никакого отношения к тому, что должно получиться (см. гл. 3). С учетом этого во многие компьютерные программы, предназначенные для решения задач линейного программирования, вводят специальные процедуры контроля за погрешностями, появляющимися вследствие округления. Однако даже в этом случае требуемая точность не всегда обеспечивается.

Итак, для реализации метода прежде всего нужно найти подходящие программы среди существующих. Если это не удается, то приходится разрабатывать новую программу. Как правило, это делается не в машинных командах, а на специальном языке. В настоя-

щее время для разработки программ существуют сотни «понимаемых» компьютерами языков. Широко распространены языки ФОРТРАН и АЛГОЛ для решения математических задач; КОБОЛ для решения экономических задач; ПЛ/1 для решения математических и экономических задач; БЕЙСИК для решения несложных математических задач и обучения информатике (программирование на языке БЕЙСИК будет рассмотрено в гл. 5). В последнее время на базе языка АЛГОЛ был разработан язык ПАСКАЛЬ, а на базе языка ПАСКАЛЬ — язык АДА.

Языка, который был бы идеальным для всех случаев, не существует. Поэтому важная задача заключается в том, чтобы определить, какой язык из имеющихся на данном компьютере является наилучшим для разработки каждой конкретной программы.

По окончании разработки программы приступают к отладке и тестированию. Основная цель отладки — выявление и исправление ошибок, связанных с нарушением правил написания программы на том или ином языке (этот процесс описан в гл. 7). По существующим оценкам отладка занимает 20...40% времени разработки программы и практически состоит из многократных попыток выполнения программы на компьютере и анализа получающихся результатов. После того как программа становится работоспособной, ее тестируют, т. е. проверяют правильность логики работы программы диапазона допустимых значений исходных данных. Наконец, по отлаженной и протестированной программе осуществляют вычисления.

Анализ полученных результатов. Нередко результаты решения задачи на компьютере даже с использованием верной программы оказываются далекими от ожидаемых. Причиной неприемлемых результатов решения может быть неадекватность исходной математической модели, т. е. слишком грубое описание реального объекта, или непригодность выбранного метода решения задачи. Тогда в процессе решения задачи с помощью компьютера используется обратная связь — от анализа результатов решения возвращаются обратно к этапу построения математической модели (и даже к постановке задачи, если в новой модели меняются данные) или к этапу выбора метода решения (рис. 2.3). Такие возвраты могут приводить к многократно-

му изменению задачи, модели или выбранного метода решения.

Если результаты решения оказываются приемлемыми, то их можно использовать для управления тем реальным объектом, который был источником задачи. Так, результаты $x_1=4{,}5$, $x_2=3$, полученные в задаче о рациональной организации работы участка по

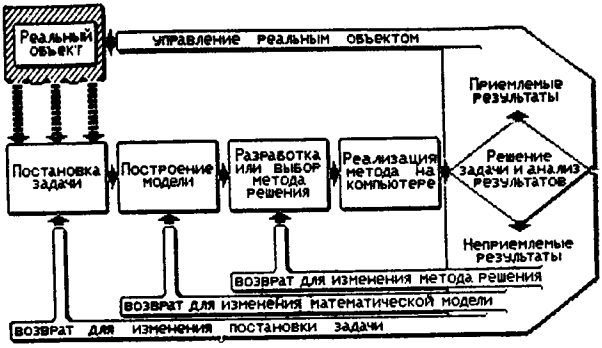


Рис. 2.3. Этапы решения задач с использованием компьютера

раскрою заготовок, можно использовать для принятия решения о ежедневной поставке на участок 4,5 т заготовок 1 и 3 т заготовок 2.

Общих способов анализа решения не существует, так как в каждом случае причины неудовлетворительного результата связаны со спецификой конкретной задачи.

2.3. ЗАДАЧИ ПО ОБРАБОТКЕ СИМВОЛЬНОЙ ИНФОРМАЦИИ

Рассмотрим задачи по обработке символьной информации на примере обработки текстов, привязываясь к последовательности этапов решения задач с использованием компьютера (рис. 2.3).

Реальным объектом в задачах по обработке текстов является широко распространенный процесс работы секретаря-машинистки над текстом.

Задача обработки текстов формулируется так:

обеспечить возможность создавать, сохранять на длительное время и корректировать любые тексты. Таким образом, и исходными данными и результатами этой задачи являются тексты.

Постановка рассматриваемой задачи приводит не к математической, а к функциональной модели, которая должна представлять собой перечень функций, необходимых для преобразования исходных текстов в результирующие. Такими необходимыми функциями являются:

удаление, вставка или изменение любых частей текста;

произвольное перемещение по тексту;

выравнивание текстов по правому полю с соблюдением правил переноса слов;

отыскание в тексте заданной фразы и замена ее на другую заданную фразу;

разметка текста: выделение слов курсивом, подчеркивание слов и т. д.;

запись готового текста или его отдельных частей во внешнюю память, например на магнитный диск, или, наоборот, возможность сборки готового текста из различных его частей, ранее записанных в памяти.

Специальные программы, выполняющие эти функции, называются *текстовыми редакторами* (ТР) или *текстовыми процессорами.*

В настоящее время текстовые редакторы интенсивно разрабатываются. Интерес к ним велик, потому что текстовые редакторы нужны всем. Поэтому этапы выбора метода и его реализация на компьютере сводятся в данном случае к поиску наиболее подходящего ТР среди существующих.

Анализ результатов применения найденного ТР может приводить к возврату на предыдущие этапы (рис. 2.3), т.е. к поискам других, более подходящих ТР. Например, популярный текстовый двухоконный редактор (TOR) для компьютера «Ямаха» способен выполнить значительно больше функций, чем перечислено выше. Однако TOR не приспособлен, например, для обработки текстов, содержащих сложные математические формулы. Поэтому для решения задач обработки математических текстов приходится возвращаться на предыдущие этапы, т. е. к поискам других ТР.

2.4. ЗАДАЧИ ПО ОБРАБОТКЕ ГРАФИЧЕСКОЙ ИНФОРМАЦИИ

Преимущество представления информации в графической форме (в виде схем, чертежей, эскизов, графиков и т. д.) проявляется особенно ярко тогда, когда числовой информации много, и в то же время важны не столько сами числа, сколько общая картина. Сравните, например, табл. 2.1 и рис. 2.2. По рисунку сразу видно, что имеется некая четырехугольная область допустимых значений x_1 и x_2 и что максимальная прибыль достигается в единственной точке M этой области.

В качестве примера рассмотрим процесс разработки так называемого графического редактора по схеме, приведенной на рис. 2.3.

Реальным объектом в задачах обработки графической информации являются плоские геометрические фигуры (изображения), которые можно воспроизвести на дисплее.

Задача по обработке графической информации формулируется так: *обеспечить возможность создавать, сохранять на длительное время и корректировать любые изображения.* Таким образом, и исходными данными, и результатами этой задачи являются изображения.

Постановка рассматриваемой задачи предполагает следующие основные функции, необходимые для преобразования исходных изображений в результирующие:

1. Изображение на дисплее графических примитивов: точек, отрезков прямых, прямоугольников, окружностей и эллипсов, букв и цифр различных шрифтов.

2. Закраска отдельных элементов изображения.

3. Работа с произвольными фрагментами изображения: выделение их на экране, удаление, сжатие, растяжение, перемещение или копирование фрагмента в нужное место изображения, симметричное отображение и т. д.

4. Запись готовых изображений во внешнюю память, например, на магнитный диск, или, наоборот, чтение готового изображения из внешней памяти.

Специальные программы, выполняющие эти функции, называются *графическими редакторами.*

Остальные этапы решения: построение модели, выбор метода, его реализация на компьютере, анализ результатов — проводятся аналогично соответствующим этапам разработки текстового редактора в § 2.3.

Функции графического редактора реализуются компьютером на основе соответствующих математических моделей. Например, симметричное отображение фигуры относительно прямой с уравнением $y=x$ строится по формулам (рис. 6.1, *д*).

2.5. УПРАВЛЕНИЕ ТЕХНИЧЕСКИМИ УСТРОЙСТВАМИ С ПОМОЩЬЮ КОМПЬЮТЕРА

Особенностью постановки задачи о рациональной организации работы участка по раскрою материалов являлось то, что после вычисления ежедневно требующихся участку количеств x_1 и x_2 заготовок 1 и 2 работа компьютера прекращалась до наступления следующего дня, а подача вычисленных количеств заготовок в цех осуществлялась уже с участием людей (см. § 2.2).

Иначе можно организовать работу участка, если компьютер постоянно управляет подачей оптимальных количеств x_1 и x_2 заготовок 1 и 2. При такой организации работы исходной информацией должна служить не табл. 2.1, предполагающая постоянство длин заготовок 1, 2 и длин выпускаемых участком продуктов 1, 2, 3 в течение дня, а таблица, в которой выход продукции на единицу веса заготовок периодически (например, ежечасно) корректируется с помощью специальных датчиков, установленных в цехе. Вычисленные компьютером количества x_1 и x_2 через специальные исполнительные механизмы можно передавать на устройства, регулирующие подачу заготовок 1 и 2. Таким образом, управляющий компьютер без вмешательства людей будет обеспечивать оптимальный режим по заложенной в него программе, составленной на основе математической модели реального производства, а выработанное компьютером решение через исполнительные механизмы будет влиять по обратной связи на управление реальным объектом. Следовательно, управление реальным объектом происходит по-прежнему согласно схеме, показанной на рис. 2.3.

Таким же образом работают компьютеры, управ-

ляющие движением самолетов и поездов, работой электростанций, телефонных сетей, металлорежущих станков, роботов и других технических устройств. Хотя эти компьютеры взаимодействуют через датчики и исполнительные механизмы с техническими устройствами, а не с людьми, человек все же может в любой момент получить от компьютера ответ на запрос о ходе управляемого компьютером процесса.

УПРАЖНЕНИЯ

1.1. Решить графически 10 задач о рациональной организации работы описанного участка, если в дополнение к ограничениям (2.2), (2.3) известно, что:

а) относительная прибыль составляет 400 руб. при использовании заготовки 1 и 600 руб. при использовании заготовки 2;

б) относительная прибыль составляет 600 руб. при использовании заготовки 1 и 200 руб. при использовании заготовки 2;

в) относительная прибыль составляет 600 руб. при использовании заготовки 1 и 300 руб. при использовании заготовки 2;

г) относительная прибыль составляет 500 руб. при использовании заготовки 1 и 800 руб. при использовании заготовки 2;

1.2. а) поставщики могут доставлять в цех ежедневно не более 4 т заготовки 1;

б) поставщики могут доставлять в цех ежедневно не более 2,5 т заготовки 2;

в) поставщики могут доставлять ежедневно не более 4 и 2,5 т заготовок 1 и 2 соответственно;

г) поставщик требует закупить у него ежедневно не менее 5 т заготовки 1;

д) поставщик требует закупить у него ежедневно не менее 4 т заготовки 2;

е) поставщики требуют закупить у них не менее 5 и 4 т заготовок 1 и 2 соответственно.

1.3. Некоторые данные табл. 2.1 пересчитаны с учетом того, что:

а) из 1 т заготовок 1 можно выпустить 0,25 т продукта 1;

б) из 1 т заготовок 2 можно выпустить 0,2 т продукта 1;

в) из 1 т заготовок 1 можно выпустить 0,35 т продукта 3;

г) из 1 т заготовок 2 можно выпустить 0,15 т продукта 3.

Глава 3

ОРГАНИЗАЦИЯ И ПРЕДСТАВЛЕНИЕ ДАННЫХ

В этой главе с общих позиций рассматривается та информация, которая обрабатывается по программе или получается в результате такой обработки.

3.1. БАЗОВЫЕ ТИПЫ ДАННЫХ

Данные различаются между собой способом кодирования, а также набором компьютерных операций, которые могут быть выполнены с ними. Например, с закодированными целыми числами можно выполнять арифметические операции, а с символами это лишено смысла.

С точки зрения пользователя, наиболее существенной характеристикой данных является набор допустимых операций, так как для решения задачи ему необходимо выполнить (с помощью компьютера) последовательность некоторых операций с теми или иными данными. Поэтому в информатике основой для классификации данных служит набор допустимых операций с ними, а не способ представления в памяти и в других устройствах компьютера (хотя это тоже имеет значение).

Все данные, для которых допустим определенный набор компьютерных операций, объединим в один класс и дадим этому классу имя (название). Такой поименованный класс назовем *базовым типом данных.* Для описания базового типа нужно каким-либо образом охарактеризовать каждую допустимую операцию, т. е. сформулировать правило определения значения результата для всех возможных значений участвующих в ней данных. Перечень всех возможных значений данных и характеристики допустимых для них операций представляют собой *функциональное описание базового типа.*

Сам по себе компьютер «не знает», какого типа информацию он перерабатывает. Если, например, при сложении двух целых чисел в программе вместо адреса первого числа ошибочно указан адрес байта, начиная с которого хранится последовательность символов STRING, то произойдет следующее:

— в АЛУ будут вызваны 2 байта, содержащие символы ST (шестнадцатеричный код B3B4, двоичная форма которого 1011001110110100);

— содержимое этих байтов интерпретируется как целое число (—19532);

— это целое число будет сложено со вторым целым числом.

Полученный таким образом результат никакого отношения к истинному не имеет, но обнаружить подобную ошибку в программе при ее отладке и тестировании очень сложно (см. гл. 7).

Если бы компьютер «знал», какого типа информация хранится по ошибочно указанному адресу, то можно было бы предусмотреть выдачу сообщения о том, что в этом месте программы

записана недопустимая операция (сложение символов с целым числом).

Таким образом, для отладки и тестирования программ необходимо иметь средства, которые позволяют указывать базовый тип данных и хранятся в том или ином месте памяти. Эти средства описываются в гл. 5.

Конечно, каждый базовый тип кроме функционального описания имеет определенную *физическую реализацию* в компьютере, т. е. способы кодирования и доступы к значениям данных.

Обычно пользователь имеет возможность оперировать с несколькими базовыми типами данных. С какими именно, определяется не столько моделью компьютера, сколько применяемыми средствами программирования. Опишем те базовые типы данных, которые практически всегда предоставляются пользователю.

а)

A	NOT A
TRUE	FALSE
FALSE	TRUE

Значение результата отрицания противоположно исходному значению

б)

A	B	A OR B
TRUE	TRUE	TRUE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE

Значение результата сложения FALSE тогда и только тогда, когда оба исходных значения FALSE

в)

A	B	A AND B
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE
FALSE	TRUE	FALSE
FALSE	FALSE	FALSE

Значение результата умножения TRUE тогда и только тогда, когда оба исходных значения TRUE

Рис. 3.1. Определение логических операций:

а — NOT (логическое отрицание), *б* — OR (логическое сложение), *в* — AND (логическое умножение), A и B — какие-либо значения типа BOOLEAN

Тип BOOLEAN (булев, логический). Имеется всего два значения данных этого типа: TRUE (*истина*) и FALSE (*ложь*). Допустимые для них операции обозначаются NOT («не», *логическое отрицание*), OR («или», *логическое сложение*), AND («и», *логическое умножение*). Результат выполнения операций также относится к типу BOOLEAN и описывается с помощью таблиц (рис. 3.1).

В данном случае функциональное описание базового типа BOOLEAN сводится к перечислению значений этого типа и результатов операций.

Что касается физического представления, то FALSE можно кодировать как 0 в одном бите, а TRUE как 1. Так обычно и поступают в некоторых устройствах АЛУ. Но доступ к памяти происходит не к отдельным битам, а к целым байтам. Поэтому кодирование FALSE может заключаться в записи нулей во все разряды одного или двух байтов (соответствует целому числу 0), а кодирование FALSE — записью единиц (соответствует целому числу 1).

Тип CHARACTER (символьный). В качестве значений данных этого типа выступают символы из таблицы кодов ASCII, (см. гл. 1). Эти значения упорядочены по шестнадцатеричному коду, так что здесь можно определить операции сравнения (проверки отношений) «меньше», «меньше или равно», «больше», «больше или равно», «равно», «не равно», которые обозначаются соответственно <, <=, >, >=, =, <>. Они характеризуются следующим образом:

определены для любых пар символов;

значение результата каждой операции относится к типу BOOLEAN, т. е. может быть TRUE или FALSE;

результатом любой из операций будет TRUE, если для шестнадцатеричных кодов справедливо соответствующее математическое отношение, и FALSE — в противном случае.

Например, результатом операции A>a будет FALSE, так как для кодов символов A и a (шестнадцатеричных чисел A1 и E1) такое отношение не выполняется. С другой стороны, результатом сравнения 5<=A является TRUE (здесь 5 не число, а символ с кодом 51), так как для шестнадцатеричных кодов 51<A1. Этим завершается функциональное описание базового типа данных CHARACTER. Способ их физического представления описан ранее в гл. 1.

Отметим, что если значения результатов сравнения имеют тип BOOLEAN, то с ними можно осуществлять логические операции. Например, результатом «составной» операции A>a AND 5<=A будет FALSE, так как результаты операций сравнения, которые выполняются перед логическими операциями, имеют значения FALSE и TRUE соответственно. В общем случае подобные «составные» выражения называются *логическими.*

Тип INTEGER (целый). Данные этого типа пред-

ставляют собой *целые числа*. Для них допустимы обычные арифметические операции сложения, вычитания, умножения и смены знака (обозначения соответственно +, —, *,—). Результатом этих операций являются целые числа.

Кроме того, для данных типа INTEGER определена операция деления нацело, которая обозначается знаком \. Ее результат получается как целая часть результата математической операции деления, например

$$1\backslash 2 = 0; \quad -(10\backslash 3) = -3; \quad (1\backslash 3) + (5\backslash 3) = \\ = 0 + 1 = 1; \quad (1 + 5)\backslash 3 = 2.$$

Последние два примера показывают, что при делении нацело раскрытие скобок может привести к изменению результата.

Целые числа упорядочены естественным образом, поэтому для типа INTEGER допустимы такие же операции сравнения, как и для типа CHARACTER: результатом сравнения является значение TRUE при выполнении соответствующего математического отношения для чисел и значение FALSE — в противном случае.

Один из способов физического представления данных типа INTEGER рассмотрен в гл. 1. Он основан на том, что целые числа представляются в двух байтах специальным кодом. При этом код 0111111111111111 соответствует числу 32767 и следующие за ним целые положительные числа не могут быть представлены в компьютере. Аналогично код 1000000000000000 соответствует минимальному отрицательному числу —32768. Таким образом, при этой физической реализации значения данных типа INTEGER заключены в пределах от +32767 до —32768.

Ограниченность значений данных целого типа имеет место при любом физическом представлении. Это приводит в некоторых случаях к невозможности выполнения арифметических операций из-за того, что их результат нельзя представить в компьютере (например, операции 30000+2768, —30000—2769 или 600·600 не имеют естественного результата). В этих случаях в АЛУ вырабатывается специальный признак такой ситуации, который можно использовать для прекращения обработки информации и выдачи пользователю сообщения. Специальный признак вырабатывается и для операции деления нацело на ноль, результат которой вообще не определен.

Тип REAL (вещественный). Здесь данные — любые вещественные числа, допустимые операции — обычные арифметические операции (включая деление), а также операции сравнения. Пример физического представления данных базового типа REAL рассмотрен в гл. 1. Под код отводится четыре байта, три из которых занимает *мантисса* (рис. 3.2).

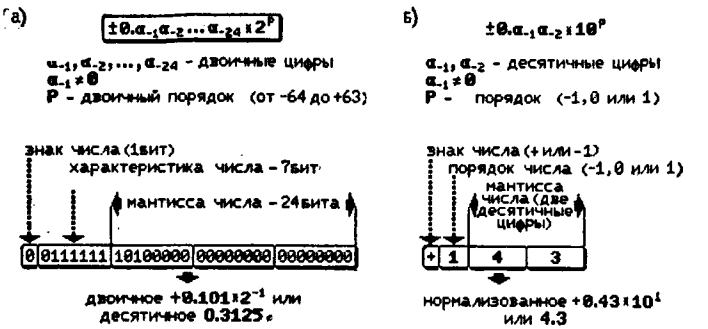


Рис. 3.2. Представление данных типа REAL:

а — двоичное представление в компьютере, *б* — десятичная модель представления

3.2. ОСОБЕННОСТИ КОМПЬЮТЕРНОЙ АРИФМЕТИКИ

Для данных вещественного типа компьютерные арифметические операции, вообще говоря, не обладают привычными математическими свойствами. Это может привести к негативным последствиям при решении вычислительных задач, в которых чаще всего используются вещественные данные.

Особенности компьютерной арифметики проиллюстрируем на десятичной модели представления (рис. 3.2). Эта модель отражает характерные свойства представления: фиксированность количества цифр мантиссы или значащих цифр (их две) и ограниченность величины порядка (от —1 до +1). Она позволяет обозреть все возможные значения данных (записаны в обычной форме с двумя значащими цифрами: 9.9, —9.8, ..., —1.0, —0.99, —0.98, ..., —0.10, —0.090, —0.080, ..., —0.010, 0.00, +0.010, ..., +9.9).

На рис. 3.3 несколько таких модельных компьютерных чисел отмечены на числовой прямой стрелками.

Для того чтобы обычное математическое число X′ представить в компьютере, его необходимо заменить значением X типа REAL. Этот процесс называется округлением. Один из возможных способов округления — усечение числа, т. е. отбрасывание «лишних» цифр. В модели на рис. 3.3 число — 0.0024 заменяется на 0.00, число 1.182 — на 1.1 и т. п.).

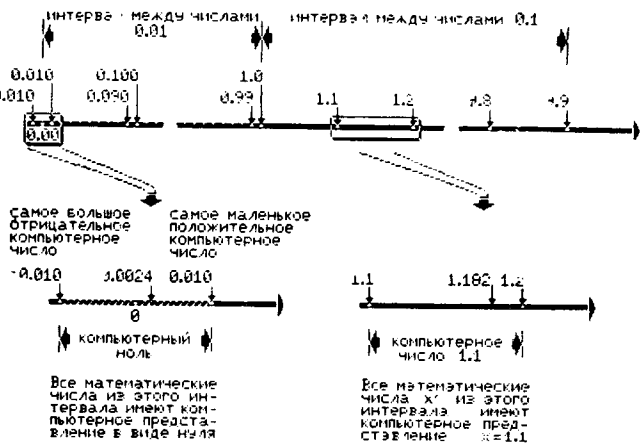


Рис. 3.3. Компьютерное представление математических вещественных чисел.

Компьютерные числа расположены дискретно и неравномерно. Компьютерный ноль — это совокупность математических чисел, которые представляются нулем

Другой, более сложный способ — обычное округление, при котором математическое число заменяется ближайшим вещественным значением (—0.0024 попрежнему заменяется на 0.00, но 1.182 — на 1.2).

Сравним два способа по оценке разности между записанным в нормализованной форме математическим числом $X' = 0.a_{-1}a_{-2}a_{-3}...\cdot 10^p$ (где $p=-1,0,1$) и соответствующим значением вещественного типа X.

В нашей модели для округления усечением абсолютная погрешность округления

$$|X' - X| \leqslant 1 \pm 0.\ a_{-1}\ a_{-2}\cdot 10^p - (\pm 0.\ a_{-1}\ a_{-2}\ a_{-3}\cdot\ ..10^p =$$
$$= 0.00\ a_{-3}\cdot ...10^p \leqslant 0.01\cdot 10^p.$$

При $X \neq 0$ относительная погрешность округления

$$\frac{|X - X'|}{|X|} \leqslant \frac{0.01 \cdot 10^p}{0.1 \cdot 10^p} = 0{,}1.$$

(В большинстве случаев при близких X и X' замена в знаменателе x' на x вполне корректна.) Отсюда

$$|X - X'| \leqslant 0.005 \cdot 10 \text{ и } |X - X'| / |X| \leqslant 0.05.$$

Одна из особенностей компьютерной арифметики состоит в том, что произведение двух ненулевых вещественных чисел может быть равно нулю. Например, для нашей модели $0{,}20 \cdot 0.020 = 0.004$ (подобный результат получается в регистре результата АЛУ, так

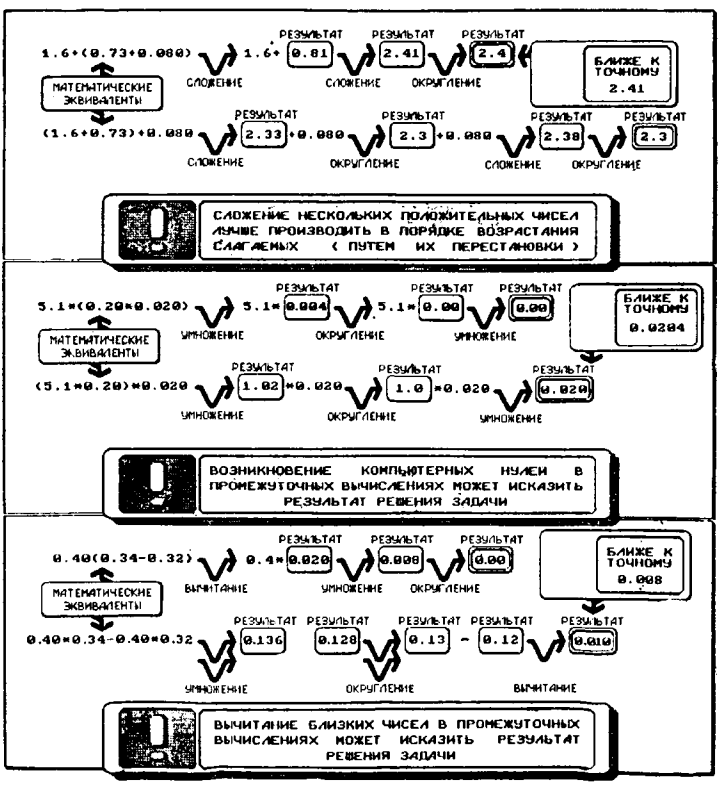


Рис. 3.4. Искажение результатов при вычислениях.

Здесь на каждом шаге разделены выполнение арифметической операции и округления ее результата

как в 2 раза больше битов, чем необходимо для представления действительных чисел). После приведения к обычной форме представления действительных чисел, т. е. округления, получается 0.00.

Еще одна особенность заключается в том, что математически допустимое изменение порядка выполнения операций может привести к изменению результата компьютерных вычислений (рис. 3.4). Подобные особенности компьютерной арифметики необходимо учитывать при анализе результата решения вычислительных задач.

Из рис. 3.4 видно, что уменьшение влияния погрешности округления иногда может быть достигнуто за счет преобразования вычислительных формул, т. е. внесения изменений в алгоритм решения задач и в программу.

Существуют другие способы уменьшения погрешности округления, использующие возможности самого компьютера (точнее применяемых программных средств).

Во-первых, практически всегда имеется еще один базовый тип данных DOUBLE PRECISION (двойная точность). Значениями данных этого типа являются тоже действительные числа, но при их физическом представлении под мантиссу добавочно отводится столько же байтов, сколько под число типа REAL. Таким образом, количество значащих цифр в представлении увеличивается примерно в 2 раза и погрешность округления уменьшается.

Так, при вычислении с четырьмя значащими цифрами получим точные результаты независимо от порядка выполнения операций:

$$1.600 + 0.7300 + 0.08000 = 2.410,$$

$$5.000 \cdot 0.2000 \cdot 0.1200 = 0.1224$$

$$\text{и } 0.4000 \cdot (0.3400 - 0.3200) = 0.008800.$$

Однако для типа DOUBLE PRECISION время выполнения арифметических операций увеличивается в среднем примерно в 4 раза.

Во-вторых, для осуществления вычислений с высокой точностью можно пользоваться базовым типом INTEGER с применением масштабирования, так как при арифметических операциях с целыми числами округления не нужны. Рассмотрим два примера.

Для вычисления суммы 1.6+0.73+0.08 умножим каждое число на масштабный коэффициент 100. Тогда для чисел целого типа получим 160+73+8=241 независимо от порядка выполнения операций. После деления суммы на 100 и округления результат будет 2,4.

Для вычисления произведения 51·0.2·0.02 умножим каждое действительное число на 100 и перемножим: 51·20·2=2040. Независимо от порядка операций после деления произведения на 1 000 000 и округления получим 0,02.

Конечно, для реальных компьютеров число цифр мантиссы составляет 6...7 (или даже 15), а округление часто осуществляется более рационально. Поэтому может показаться, что погрешность вычислений не должна быть слишком большой. Для того

x	S (тип REAL)	S (тип DUBLE PRECISION)	exp(-x) (точное значение)
1.0	0.367880	0.367879	0.367879
11.0	0.013726	0.000017	0.000017
21.0	436.683000	0.000000	0.000000
31.0	861683.000000	0.070905	0.000000
41.0	7690240000.000000	1797.503223	0.000000

Рис. 3.5. Таблица приближенных значений функции $y=\exp(-x)$.

Ниже показанной границы результаты вычислений неприемлемы

чтобы рассеять иллюзии, на рис. 3.5 приведена таблица показательной функции $y=\exp(-x)$, значения которой вычислены на компьютере по приближенной формуле

$$\exp(-x) \approx 1 - x + \frac{x^2}{2} - \frac{x^3}{6} + \ldots + (-1)^n \frac{x^n}{n!} = S.$$

Эта формула устанавливается в курсе высшей математики и здесь же доказывается, что для каждого значения x количество слагаемых n можно подобрать так, чтобы разность между $\exp(-x)$ и вычисляемой суммой S (т. е. погрешность метода) была как угодно мала. В данном случае для каждого X количество слагаемых подбиралось так, чтобы погрешность метода равнялась 0.000001.

Из таблицы на рис. 3.5 видно, что при больших значениях результат вычислений абсурден. Это можно заметить и не зная точного значения вычисляемой величины (как это бывает в ре-

альных задачах), а используя известное свойство о приближении $\exp(-x)$ к нулю и неограниченном увеличении x.

Так как неустранимая погрешность отсутствует (исходное значение x задается точно), а погрешность метода мала, то катастрофическое искажение результата происходит из-за округлений. При этом использование данных типа DOUBLE PRECISION не избавляет от этого эффекта, а только отодвигает его в сторону больших x.

Существуют другие методы вычисления функции $y=\exp(-x)$. Результаты расчетов по одному из них приведены в последнем столбце таблицы на рис. 3.5, в качестве точного значения $\exp(-x)$. Они показывают, что изменение алгоритма может значительно уменьшить погрешность округления.

3.3. ПОНЯТИЕ О СТРУКТУРАХ (АБСТРАКТНЫХ ТИПАХ) ДАННЫХ

При постановке многих задач и разработке алгоритмов их решения исходную информацию, промежуточные и окончательные результаты удобнее представлять не в виде базовых типов данных, а с помощью более сложных структурных единиц. Например, при обработке текстов удобно описывать задачи и алгоритмы с помощью строк символов и различных операций с ними: присоединения друг к другу, сравнения между собой и т. п.

Понятие строки можно описать через базовый тип данных CHARACTER. *Строка* — это последовательность символов (например, «на», «сос», «father», «family»). Операции со строками выражаются через операции с символами следующим образом (рис. 3.6, *а — в*):

операция присоединения двух строк (обозначается знаком «+») заключается в образовании новой строки, представляющей собой последовательность символов сначала первой строки, а затем второй (например, «на»+«сос» есть «насос» , но «сос»+«на» есть «сосна»);

операция сравнения двух строк заключается в последовательном слева направо попарном сравнении отдельных символов и присваивании окончательному результату логических значений TRUE или FALSE (например, результатом операции сравнения father$<$ $<$family будет FALSE, ибо здесь две первые пары символов совпадают, а для третьей пары результат сравнения $t<m$ будет FALSE).

Аналогично описываются и другие возможные операции сравнения, при которых нужно иметь в виду, что при разной длине строк более короткая как бы дополняется символами «пробел».

В рассмотренном примере сначала перечислены операции со строками (присоединения, сравнения и т. п.). Далее элементы строки описаны через данные более простой структуры (базовый тип CHARACTER),

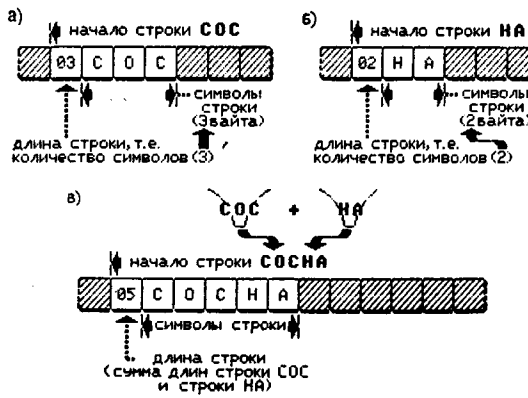


Рис. 3.6. Схема представления строк и реализации операции «+».

В качестве длины строки используется шестнадцатеричный код количества символов. Поскольку под него отводится 1 байт, то максимальная длина строки 255 символов (FF в шестнадцатеричной системе)

а операции со строками — через операции с этими данными. Затем приводится представление строки и реализация операций с ней в компьютере. Наконец, этому типу данных можно дать имя (например, STRING — английское слово «строка»).

По аналогии дадим общее описание понятия структуры данных, или, как еще говорят, абстрактного типа данных.

Под *структурой (абстрактным типом) данных* понимается поименованная совокупность данных, которые имеют один и тот же набор допустимых операций. Перечень и характеристики всех операций представляют собой *функциональное описание* структуры.

Описание отдельных составляющих элементов ис-

ходной структуры, представляющих собой более простые структуры и даже базовые типы, а также операций с данными исходной структуры через операции над составляющими простыми элементами называется *логическим описанием* исходной структуры (вообще говоря, структуры данных допускают различные логические описания).

Последний уровень описания структуры данных — *физическое представление*, которое дает способы хранения структур в памяти, доступа к ним, а также выполнения операций.

Конечно, введенные ранее базовые типы данных являются структурами, для которых отсутствует логическое описание, поскольку они самые элементарные.

Рассмотрим пример постепенной детализации структуры данных при логическом описании.

Допустим, что при решении задач, связанных с подсчетом успеваемости, используются такие данные, как экзаменационные ведомости. В число операций, которые можно осуществлять с ведомостями, входят вычисление среднего балла, выборка фамилий студентов, получивших определенную оценку, и т. п. Этот перечень представляет собой функциональное описание структуры «Ведомость».

Представим «Ведомость» как совокупность записей, содержащих фамилии и оценки. При этом операция вычисления среднего балла представляет собой выборку и суммирование оценок из всех записей, а выборка фамилий студентов — просмотр записей и сравнение оценок с некоторой заданной. Аналогично можно описать и другие необходимые операции. Таким образом, структура «ведомость» представлена через структуру «Запись».

В свою очередь структура «Запись» состоит из двух упорядоченных элементов: данных типа STRING, представляющих собой фамилию студента и описанных в первом примере с помощью базового типа CHARACTER, и данных REAL, представляющих экзаменационную оценку и являющихся базовым типом. Операции выборки оценок, сравнения оценок с заданной, выборки фамилий и другие описываются через известные операции со строками и вещественными чис-

лами. На этом детализация исходной структуры данных «Ведомость» завершается.

Для полноты описания структуры «Ведомость» дополним его физическим представлением (рис. 3.7).

В соответствии с физическим представлением при просмотре записей адрес следующей записи определяется путем добавления к адресу просмотренной записи количества ее элементов и увеличением получаемого значения на 4 (4 байта занято оценкой типа REAL). Для выборки фамилии из записи выбираются символы, начиная с того, который имеет адрес на единицу больше, чем адрес начального байта строки. Для выборки оценки адрес соответствующего действительного числа определяется путем увеличения адреса начального байта строки на количество символов в строке и еще на единицу.

Рис. 3.7. Схема представления взаимосвязанных структур «Ведомость», «Запись», STRING (фамилии) и REAL (оценки)

Понятие структуры данных (или абстрактного типа данных) является важным, по крайней мере, по двум причинам. Во-первых, в некоторых языках программирования предусматриваются операции со структурами данных, отличными от базовых типов. Более того, в ряде языков существуют способы описания новых структур, необходимых пользователю. Во-вторых, в программное обеспечение компьютеров входят специальные средства для манипулирования сложными структурами данных (например, системы управления базами данных, текстовые и графические редакторы).

3.4. ПРИМЕРЫ СТРУКТУР ДАННЫХ: СТЕКИ И МАССИВЫ

В качестве первого примера достаточно сложной структуры данных рассмотрим стек, о котором упоминалось в гл. 2.

Образно **стек** можно представить как стопку документов, т. е. информации, в которую поступающие документы последовательно добавляются сверху и извлекаются для обработки тоже сверху. Поэтому, как показано на рис. 3.8, первым обрабатывается тот документ, который поступил последним.

Функционально стек представляет собой совокупность данных одинаковой структуры (или одного и того же базового типа), которая характеризуется следующими операциями:

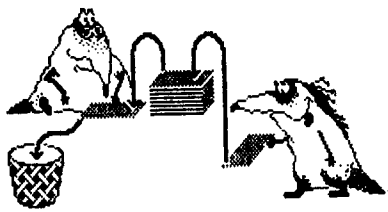


Рис. 3.8. Обработка информации на основе стека

«созданием» пустого, т. е. не содержащего никаких элементов, стека (в этой операции нет исходных данных, а результатом является пустой стек);

«добавлением» элемента к стеку (исходные данные — элемент и стек, а результат — стек);

«извлечением» элемента, добавленного в стек последним (исходные данные — стек, результат — элемент и стек);

«проверкой» стека на отсутствие в нем элементов (исходные данные — стек, результат имеет тип BOOLEAN: при отсутствии элементов TRUE, а при наличии FALSE).

На этом примере проиллюстрируем формальный способ функционального описания структуры, при котором операции характеризуются набором определенных свойств (аксиом).

Введем обозначение S для структуры, которая будет определена как стек, и E для структуры (базового типа), к которой относятся элементы стека.

При описании операций сначала укажем их название (в кавычках), затем исходные данные (в скобках после названия) и, наконец, результаты (в скобках после знака →):

«создание»→(S)

«добавление» (E, S)→(S)

«извлечение» ((S)→(E, S)
«проверка» (S)→(BOOLEAN)

Стеком называется структура, для которой введенные операции удовлетворяют следующим соотношениям (аксиомам):

«проверка» («создание») = TRUE
«проверка» («добавление» (E, S)) = (FALSE)
«извлечение» («добавление» (E, S)) = (E, S)
«добавление», («извлечение» (S)) = S, если «проверка» (S) = = FALSE.

Первые две записи означают, что при операции «проверка» результат TRUE получается только для пустого стека, а следующие две показывают взаимную противоположность операций «извлечение» и «добавление».

Логическое описание стека может быть, например, связано с введением указателя, который показывает последний добавленный в стек элемент (вершину стека). При этом операции со стеком связываются с состоянием указателя так, как это показано на рис. 3.9.

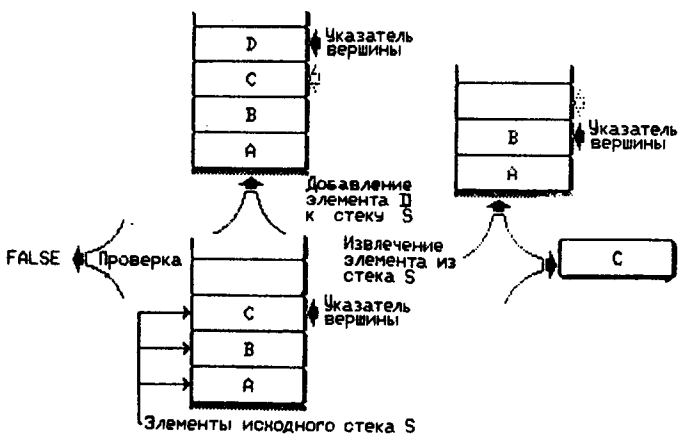


Рис. 3.9. Логическое описание стека

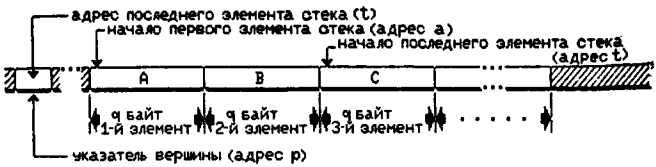
Один из способов физического представления стека, когда его элементы записываются в память подряд один за другим, показан на рис. 3.10. Указателем вершины является определенное место в памяти, в котором хранится адрес начального байта последнего записанного в стек элемента.

В качестве примера применения стека рассмотрим способ вычисления значений арифметических выражений, записанных в так

называемой обратной польской записи. Суть ее состоит в том, что знаки операций ставятся после величин, над которыми эти операции производятся:

$$A + B \to AB +,\ A * B \to AB *,\ (A + B) * C + A * B \to AB + + C * AB * +$$

Такой записью пользуются, например, при вычислениях на микрокалькуляторах.



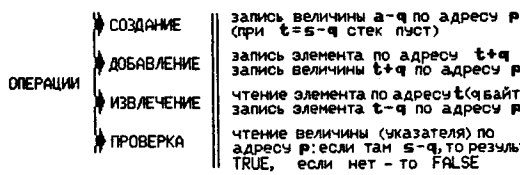


Рис. 3.10. Физическое представление стека в памяти

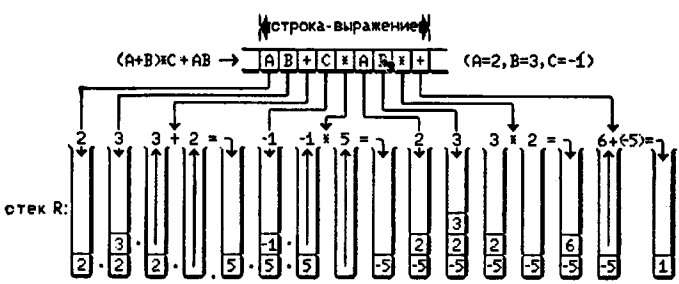


Рис. 3.11. Вычисление значения арифметического выражения с использованием стека

Для вычисления результата создается специальный стек S. Далее строка (тип STRING) обратной польской записи последовательно просматривается слева направо. Если при этом встречается буква, то обозначаемая ею величина помещается в стек S. Если же в строке встречается знак операции, то она производит-

ся с двумя последовательно извлекаемыми элементами стека S и полученный результат возвращается в стек.

На рис. 3.11 для выражения AB+C*AB*+ при A=2, B=3 и C=—1 изображены последовательные состояния стека R.

Пример другой структуры данных — массив, который можно представить как папку с пронумерованными документами (информацией), каждый из которых может быть прочитан или заменен на другой (рис. 3.12).

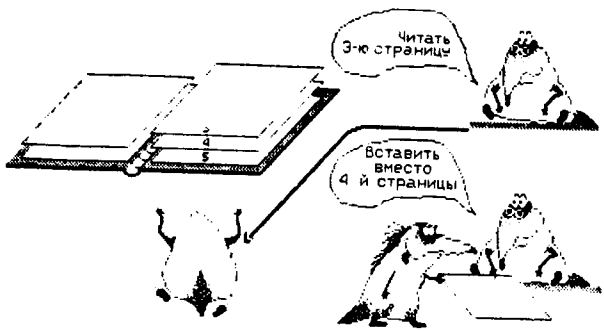


Рис. 3.12. Обработка информации на основе массива

С функциональной точки зрения простейший массив T — это совокупность N элементов одинаковой структуры, каждому из которых поставлен в соответствие порядковый номер 1, 2, ..., N, называемый *индексом*. Для массива T определены следующие операции:

«создание» массива T, имеющего N элементов, значения которых еще не определены;

«чтение» элемента с заданным индексом i=1, 2,..., N, результатом которого является элемент T(i), имеющий этот индекс, а также исходный массив T;

«замена» элемента массива с данным индексом i на данный элемент G, в результате которой получается новый массив, отличающийся от исходного наличием элемента G на i-м месте (T(i)=G).

Логическое описание массива представлено на рис. 3.13, а физическое представление — на рис. 3.14.

В качестве примера использования массива покажем, как с его помощью можно моделировать стек (в отличие от «стека»

структура «массив» является обязательной для большинства языков программирования). Для этого нужно выразить операции со стеком через операции с массивом;

созданию стека соответствует операция создания массива и присваивания значения 1 некоторой величине i целого типа; эта величина играет роль указателя вершины, точнее, ее значение является индексом доступного для записи элемента массива;

добавлению элемента в стек соответствует операция замены элемента массива с индексом i и увеличение величины i на 1 (если это возможно, т. е. $i \leqslant N-1$, где N — количество элементов в массиве);

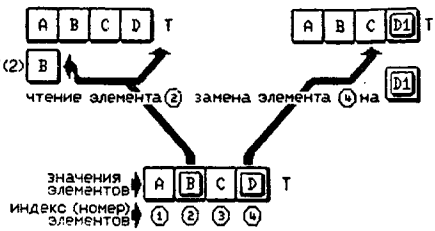


Рис. 3.13. Логическое описание массива.

Операция создания массива определяется количеством элементов и организует пустой массив (т. е. заполненный произвольными элементами)

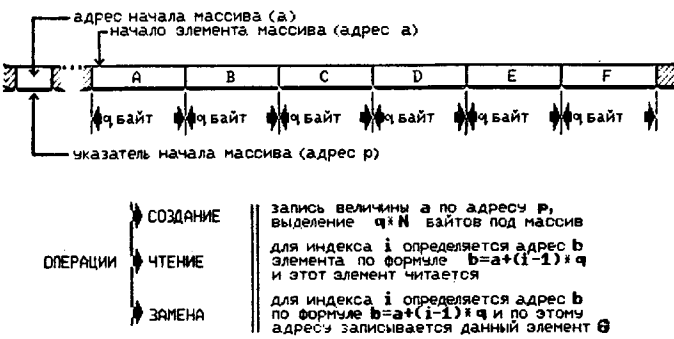


Рис. 3.14. Физическое представление массива

извлечению элемента из стека соответствует операция чтения элемента с индексом $i-1$ и уменьшение величины i на 1 (эта операция возможна, если $i>1$);

проверка стека на пустоту соответствует логической операции сравнения i с 1.

Рассмотрим типичный случай, когда элементами массива являются простейшие массивы (естественно,

имеющие одинаковое количество элементов). Логическое описание массива показано на рис. 3.15.

Операция создания массива определяется двумя параметрами: количеством элементов во «внешнем» массиве (строке) и в каждом «внутреннем» массиве (колонке).

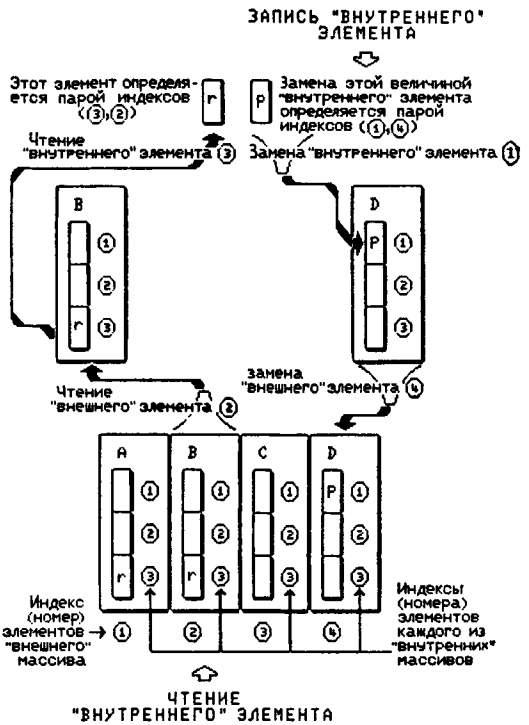


Рис. 3.15. Логическое описание массива, элементами которого являются массивы

Если элементами массива являются числа (целые или действительные), то его можно интерпретировать как вектор, номера компонент которого представляют собой индексы соответствующих элементов. Если массив трактуется как матрица (таблица), то номера строки и колонки элементов представляют собой па-

ры индексов элементов массива, позволяющие однозначно определить эти элементы.

Элементы массива могут определяться тройкой, четверкой и т. д. индексов. Вообще, принято называть количество индексов *размерностью* массива и говорить об одномерных, двумерных, трехмерных и т. д. массивах.

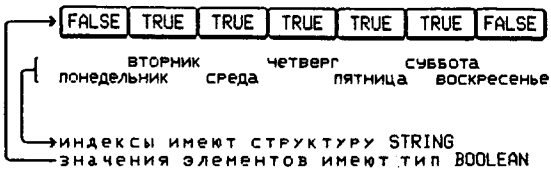


Рис. 3.16. Массив, в котором хранится информация о днях работы и отдыха работающих по скользящему графику

В качестве индекса для определения того элемента массива, с которым нужно произвести какую-либо операцию, можно брать не целое число, а какой-либо другой базовый тип или даже более сложную структуру данных.

Пример одномерного массива с индексом типа STRING и элементами типа BOOLEAN приведен на рис. 3.16.

Различные примеры использования массивов и других структур, а также базовых типов данных будут рассмотрены в других главах этого пособия.

УПРАЖНЕНИЯ

3.1. Доказать (путем перебора возможных значений), что для любых величин A, B, C типа BOOLEAN следующие пары логических выражений имеют одинаковые значения (эквивалентны):

A OR B и B OR A
A AND B и B AND A
— свойство коммутативности;

(A OR B) OR C и A OR (B OR C)
(A AND B) AND C и A AND (B AND C)
— свойство ассоциативности;

A AND (A OR B) и A
A OR (A AND B) и A
— свойство поглощения;

A AND (B OR C) и (A AND B) OR (A AND C)
A OR (B AND C) и (A OR B) AND (A OR C)
— свойство дистрибутивности;

A AND (NOT A) и FALSE — свойство противоречия;
NOT (NOT A) и A — свойство двойного отрицания.

Эти эквиваленты выражают характерные свойства логических операций NOT, OR, AND.

3.2. Определить, сколько имеется компьютерных вещественных чисел при представлении их t двоичными разрядами и порядком в пределах от p до P.

3.3. Записать в обратной польской записи следующие выражения:

$$A - B * C/(A + C)$$

$$A - B * C/A + C$$

Изображать динамику состояний стека результата для A=2, B=3, C=6.

3.4. Структура «очередь» характеризуется тем, что из нее можно извлечь только первый элемент (записанный раньше всех имеющихся), а записать — самый последний. Таким образом, это модель реальной очереди на обслуживание. Попытаться дать формальное описание «очереди», ее логическую структуру и какое-нибудь физическое представление.

3.5. Элементы двумерного массива A — вещественные числа, которые размещаются в памяти друг за другом по столбцам. Считая, что в массиве N строк и M колонок, найти формулу для вычисления адреса элемента массива A (i, j) по значениям индексов i и j (адрес начала A задан).

Глава 4

АЛГОРИТМЫ И ТЕХНОЛОГИЯ ИХ РАЗРАБОТКИ

В данной главе рассматриваются алгоритмы и основные алгоритмические структуры, типовые алгоритмы обработки информации и приводятся примеры типовых алгоритмов обработки одномерных массивов и алгоритмов со вложенными циклами.

4.1. АЛГОРИТМЫ И ОСНОВНЫЕ АЛГОРИТМИЧЕСКИЕ СТРУКТУРЫ

Алгоритм — это система правил, описывающая последовательность действий, которые необходимо выполнить, чтобы решить задачу. Понятие алгоритма в информатике является фундаментальным, таким, каким являются понятия точки, прямой и плоскости в геометрии, множества — в математике, пространства и времени — в физике, вещества — в химии. Как и для всякого фундаментального понятия, для алгоритма невозможно дать абсолютно строгого определения. Поэтому формулировка, приведенная выше, лишь приближенно описывает алгоритм.

Разработанный алгоритм может исполнять как человек, так и техническое устройство (например, компьютер, робот), которые понимают правила записи алгоритма и умеют выполнять предписываемые им действия. Придерживаясь указаний алгоритма, исполнитель получит пригодный результат даже в том случае, когда он не понимает существа решаемой задачи.

Например, выполняя инструкции, имеющиеся в кабине междугородного телефона-автомата, можно дозвониться до абонента, проживающего в другом городе.

Выполняя инструкции, записанные в виде программы (это тоже форма записи алгоритма), компьютер получит результат, удовлетворяющий пользователя.

Основные характеристики алгоритма. Всякий алгоритм обработки информации характеризуется дискретностью, определенностью, результативностью и массовостью.

Дискретность означает, что выполнение алгоритма разбивается на последовательность законченных действий — шагов. Каждое действие должно быть завершено исполнителем прежде, чем он перейдет к выполнению следующего. Значения величин в каждом шаге алгоритма получаются по определенным правилам из значений величин, определенных на предшествующем шаге.

Под *определенностью* понимается то обстоятельство, что каждое правило алгоритма настолько четко и однозначно, что значения величин, получаемые на каком-либо шаге, однозначно определяются значениями величин, полученными на предыдущем шаге, и при этом точно известно, какой шаг будет выполнен следующим. Это означает также, что исполнитель, например компьютер, должен быть в состоянии выполнить каждую команду алгоритма в строгом соответствии с ее назначением.

Результативность (или *конечность*) алгоритма предполагает, что его исполнение сводится к выполнению конечного числа действий и всегда приводит к некоторому результату. В качестве одного из возможных результатов является и установление того факта, что задача решения не имеет.

Под *массовостью* понимается, что алгоритм решения задачи разрабатывается в общем виде так, чтобы

его можно было применить для целого класса задач, различающихся лишь наборами исходных данных. При этом исходные данные могут выбираться из некоторой области, называемой областью применимости алгоритма. В свойстве массовости заключена основная практическая ценность алгоритмов.

Формы представления алгоритмов. Существуют различные формы представления алгоритмов. Основными среди них являются:

словесное описание алгоритма на естественном языке (вербальная форма);

построчная запись алгоритма;

блок-схема;

запись на каком-либо языке программирования.

Рассмотрим особенности каждой из названных форм и в качестве примера представим в каждой форме один и тот же алгоритм для определения наибольшего общего делителя (НОД) двух целых положительных чисел (алгоритм Евклида).

Словесное описание. Словесное описание имеет минимум ограничений и является наименее формализованным. Однако при этом алгоритм получается и наименее строгим, допускающим появление неопределенностей. Алгоритм в вербальной форме может оказаться очень объемным и трудным для восприятия человеком.

Например, если числа равны, НОД равен одному из них. В противном случае надо из большего числа вычесть меньшее, полученную разность запомнить вместо значения большего числа и повторить все сначала.

Построчная запись алгоритма. Это запись на естественном языке, но с соблюдением некоторых дополнительных правил. Сформулируем эти правила.

1. *Шаги (предписания) алгоритма нумеруются.*

2. *Исполнение алгоритма происходит в порядке возрастания номеров шагов, начиная с первого (если не встречается никаких специальных указаний).*

3. Типичными шагами алгоритма являются:

чтение (ввод) данных, которое записывается в виде

Чтение A, B, ...,

где A, B,... — обозначение исходных данных;

обработка данных (вычисления) по формулам, записанная в виде

V=...,

где V — переменная, значение которой определяется как результат, полученный после выполнения операций, указанных в правой части.

По-существу знак «=» обозначает специальную операцию, которая называется *присваиванием*;

сообщение (вывод) результата, который имеет вид

Запись x, y, ...,

где x, y, ... — обозначение переменных, значение которых необходимо узнать;

проверка условия, запись которого

Если ... идти к *N*,

где вместо многоточия записывается условие, при выполнении которого осуществляется переход к шагу с номером N (если условие не выполняется, то производится переход к следующему по порядку шагу);

переход к шагу с номером N:

Идти к N;

конец вычислений:

Останов.

Пример:
```
[1] Чтение A, B
[2] Если A=B, идти к [8]
[3] Если A=B, идти к [6]
[4] B=B—A
[5] Идти к [2]
[6] A=A—B
[7] Идти к [2]
[8] НОД=A
[9] Запись НОД
[10] Останов
```

Построчная запись алгоритма позволяет избежать неопределенностей в алгоритме, не требует, по-существу, никаких специальных знаний (понимание правил, перечисленных выше, трудности не составляет) и в то же время обеспечивает отработку навыков логически строгого изложения хода решения задачи (последовательности вычислений, возможных вариантов перехода к различным шагам алгоритма и т. д.) и облегчает последующее изучение алгоритмических языков.

Однако построчная запись алгоритма воспринимается человеком тяжело и требует большого внимания при изучении (или записи) алгоритмов в этой форме представления.

Изображение алгоритма в виде блок-схемы отличается высокой степенью наглядности. Блок-схема состоит из соединенных между собой стрелками (линиями потока) блоков различного вида (рис. 4.1, *а — и*) и необходимого количества комментариев.

Выполнение алгоритма всегда начинается с блока начала и оканчивается при попадании на блок конца. Порядок вычислений определяется стрелками.

В блоке обработки данных содержится описание тех действий, которые должны быть выполнены над объектами при попадании на этот блок по входящей в него стрелке. Здесь вычисляются выражения и присваиваются новые значения переменным.

Проверка условия изображается с помощью блока принятия решения, внутри которого записывается это условие. В результате проверки выбирается одна из двух стрелок, определяющая направление дальнейших вычислений.

Внутри блока ввода или вывода пишется (необязательно) слово «Ввод» или «Вывод» и перечисляются переменные, значения которых должны быть введены или выведены в данном месте схемы.

Комментарии используются в тех случаях, когда пояснение не помещается внутри блока. Совокупность комментариев должна делать блок-схему понятной для любого пользователя.

Нередко возникает необходимость применения уже имеющихся (разработанных кем-то) алгоритмов. В этом случае можно использовать блок «предопределенный процесс». Если же часть алгоритма подлежит уточнению (детальной проработке) в дальнейшем, то в блок-схеме эта часть изображается с помощью блока «детализированная программа».

При большой насыщенности блок-схем блоками допускается прерывать стрелки, а затем продолжать их в нужном месте (т.е. как бы удалять часть стрелки, пересекающую блок-схему). В этом случае начало и конец удаленных участков обозначаются соединителями, внутри которых записываются (для каждой

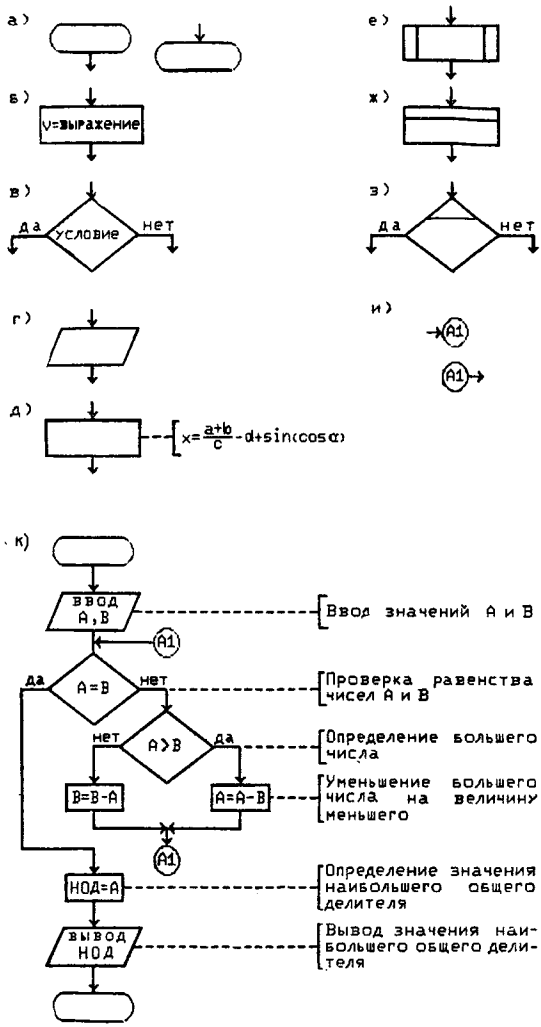


Рис. 4.1. Обозначения на блок-схемах:

а — начало, конец; *б* — обработка данных; *в* — проверка условия; *г* — ввод-вывод; *д* — блок обработки данных с комментарием; *е* — предопределенный процесс; *ж*, *з* — детализированная программа; *и* — соединитель; *к* — пример блок-схемы для алгоритма Евклида

стрелки) одни и те же обозначения: буква, буква и цифра или цифра.

Помимо этих блоков допустимо использование и некоторых других, однако перечисленных выше вполне достаточно для разработки учебных блок-схем.

Пример записи алгоритма Евклида показан на рис. 4.1, *к*.

Запись алгоритма на языке программирования. Эта запись (программа 4.1) представляет собой форму изображения алгоритма в том случае, когда исполнителем алгоритма является компьютер. Языки программирования имеют более жесткие правила, чем, например, правила описания алгоритма на естественном языке, так как их должна «понимать» машина. Однако лишние сложности записи алгоритма на таких языках окупаются возможностью их автоматического «прочтения» и исполнения.

Программа 4.1.

Запись алгоритма Евклида на языке БЕЙСИК

```
10 INPUT  "Введите числа A и B"; A,B
15 REM Ввод чисел A и B
20 IF A = B THEN 80
25 REM Проверка равенства чисел A и B
30 IF A>B THEN 60
35 REM Больше ли A, чем B?
40 B = B - A
45 REM Уменьшение числа B
50 GOTO 20
55 REM Переход на проверку условия равенства A и B
60 A = A - B
65 REM  Уменьшение числа A
70 GOTO 20
75 REM Переход на проверку условия равенства A и B
80 NOD = A
85 REM Присваивание значения переменной NOD
90 PRINT "NOD="; NOD
95 REM Вывод значения NOD
100 END
105 REM Конец вычислений
```

В этом примере после слов REM приводятся комментарии, расшифровывающие действия команд в предыдущих строках примера, благодаря чему легко

обнаруживается тесная связь между командами языка программирования (в данном случае языка БЕЙСИК) и предписаниями построчной записи.

Основные типы алгоритмических структур. Их всего три: линейная, разветвляющаяся и циклическая.

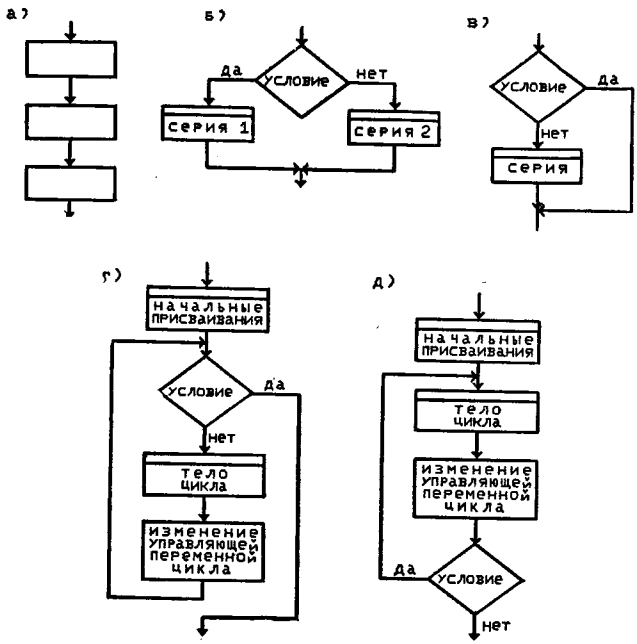


Рис. 4.2. Основные типы алгоритмических структур:

а — линейная; *б* — разветвляющаяся, в которой выполняется либо одна, либо другая серия шагов; *в* — разветвляющаяся, в которой серия шагов либо выполняется, либо пропускается; *г* — циклическая, в которой повторяющиеся шаги и изменение переменной цикла выполняются после проверки условия окончания цикла (цикл «пока»); *д* — циклическая, в которой повторяющиеся шаги и изменение переменной цикла осуществляются до проверки условия окончания цикла (цикл «до»)

Линейная структура предполагает однократное выполнение одной и той же последовательности шагов при любых наборах исходных данных. Для разветвляющейся структуры также характерно однократное выполнение последовательности шагов, однако состав этой последовательности опреде-

ляется результатами проверки некоторого условия, т. е. зависит от обрабатываемой информации. Циклическая структура обеспечивает многократное выполнение одной и той же последовательности шагов тела цикла с модифицируемой (изменяемой) информацией.

На рис. 4.2 основные типы алгоритмических структур показаны в виде фрагментов блок-схем.

Заметим, что для одного набора исходных данных линейная и разветвляющаяся структуры выполнятся только по одному разу, цикл «до» — по крайней мере один раз (так как первая проверка условия окончания цикла происходит после того, как тело цикла выполнено), цикл «пока» может не выполниться ни разу.

Как для цикла «пока» так и для цикла «до» перед входом в цикл обязательно начальное *присваивание*, т.е. задание начальных значений тем переменным, которые используются для управления количеством повторений цикла и для накапливания результата.

Можно выделить циклы с известным заранее количеством повторений и так называемые *итерационные* циклы, в которых число прохождений тела цикла заранее не определяется. Для циклов с известным количеством повторений задаются начальное и конечное значения переменной цикла. Ее значение изменяется при каждом повторении цикла по некоторому закону, который также задается. Этим и определяется количество повторений.

В итерационных циклах закон изменения переменной цикла в явном виде не описывается.

4.2. ТИПОВЫЕ АЛГОРИТМЫ ОБРАБОТКИ ИНФОРМАЦИИ

Несмотря на разнообразие решаемых задач по обработке информации, нередко оказывается, что алгоритмы решения многих из них распадаются на те или иные типовые задачи. Поэтому значение различных типовых алгоритмов, умение выявлять их внутри задач существенно облегчает процесс разработки алгоритмов: уменьшает вероятность появления ошибок, сокращает время на алгоритмизацию. Навыки в использовании типовых алгоритмов позволяют решать многие прикладные практические задачи приспособ-

лением алгоритмов к условиям задачи путем их незначительной модификации.

Ниже рассматриваются некоторые типовые алгоритмы обработки информации.

Задачи с линейной алгоритмической структурой — это самый простой класс задач. Блок-схема таких задач представляет собой линейную последовательность выполняемых блоков (команд).

Пример 4.1. На рис. 4.3 приведена блок-схема алгоритма определения площади треугольника по длинам его основания и опущенной на него высоты.

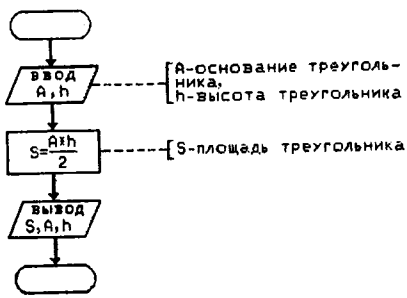


Рис. 4.3. Блок-схема алгоритма определения площади треугольника

Другие задачи этого класса отличаются от приведенного примера только количеством блоков в блок-схемах.

Задачи данного класса в чистом виде встречаются очень редко. Как правило, линейные последовательности действий (серии команд) используются в виде компонентов в разветвляющихся или циклических алгоритмах.

Задачи с разветвляющейся алгоритмической структурой. Рассмотрим несколько примеров задач с таким алгоритмом.

Пример 4.2. Абсолютная величина (модуль) числа B определяется следующим образом:

$$|B| \begin{cases} B, & \text{если } B \geqslant 0, \\ -B, & \text{если } B < 0. \end{cases}$$

Блок-схема алгоритма вычисления модуля для произвольного числа B показана на рис. 4.4.

Для конкретного числа B выполнится или только левая ветвь (если $B \geqslant 0$) или только правая (если $B < 0$).

Пример 4.3. Площадь плоской фигуры, ограниченной прямоугольником, кругом или двумя концентрическими окружностями, определяется по формулам

$$F = \begin{cases} bh \text{ для прямоугольника,} \\ \pi d^2/4 \text{ для круга,} \\ \pi/4\,(d^2 - d_1^2) \text{ для концентрических окружностей.} \end{cases}$$

Алгоритм вычисления площади для перечисленных видов фигур приводится на рис. 4.5 (фигуры получаются как сечения некоторого тела).

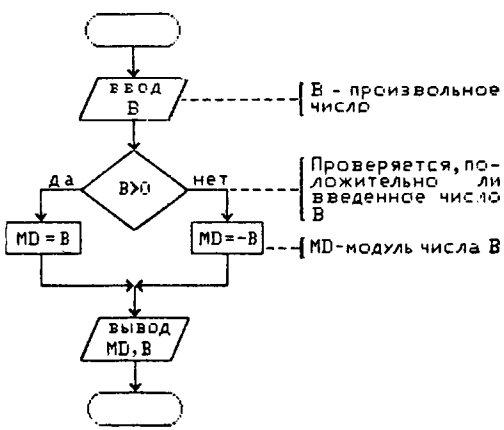


Рис. 4.4. Блок-схема алгоритма определения абсолютной величины числа

Если пользователь вводит значение кода, не входящее в допустимый интервал значений кода, то после вывода сообщения о неверном задании кода предусматривается повторное выполнение алгоритма. В известном смысле такой алгоритм защищен от ошибок пользователя при задании информации.

Так же, как и в примере 4.1, процесс вычислений пройдет только по одной из ветвей алгоритма — по той, которая соответствует введенному значению кода

К. Для того чтобы пройти по другой ветви, необходимо ввести и другое значение К.

На рис. 4.5 приведена блок-схема решения конкретной задачи. Между тем подобные алгоритмы характерны для целого класса задач, связанных с необходимостью выполнения той или иной последовательности шагов в зависимости от результатов проверки нескольких условий. Количество блоков проверки условий и выполняемых шагов в каждой серии определяется конкретной задачей.

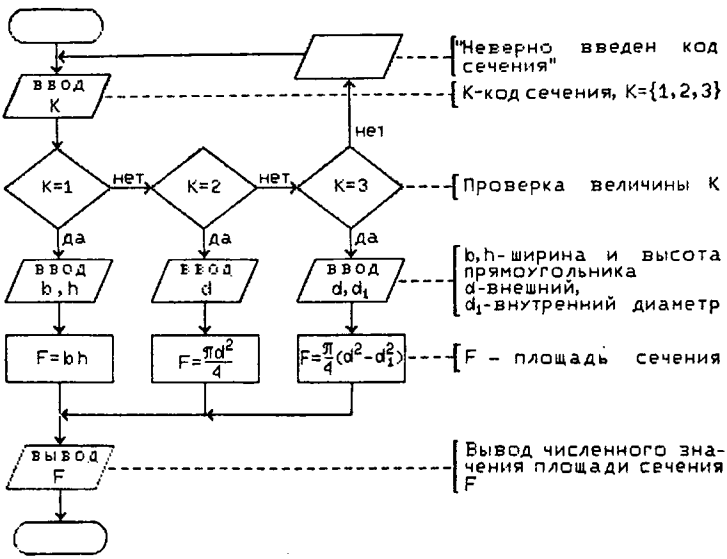


Рис. 4.5. Блок-схема алгоритма определения площади плоской фигуры, получаемой как сечение тела

Следует также обратить внимание на то, что количество блоков проверки условий в блок-схемах разветвляющихся алгоритмов всегда на единицу меньше, чем число различных ветвей. Так, в примере 4.1 для получения двух ветвей используется один блок проверки условия (один ромб), во втором — для получения четырех ветвей (включая ветвь, в которой К не соответствует множеству допустимых значений) применяются три таких блока.

Задачи с циклической алгоритмической структурой. Рассмотрим примеры типовых циклических алгоритмов.

Пример 4.4. Составить таблицу значений функции $F=x^2+6x+10$ при изменении x от x_0 до $x_к$ с шагом t. Блок-схема алгоритма показана на рис. 4.6.

Телом цикла в данной схеме являются два блока: определение F и печать F и x. Тело цикла выполняется до тех пор, пока значение переменной цикла x не превысит конечное значение $x_к$. В тот момент, когда x окажется больше $x_к$, выполнение цикли-

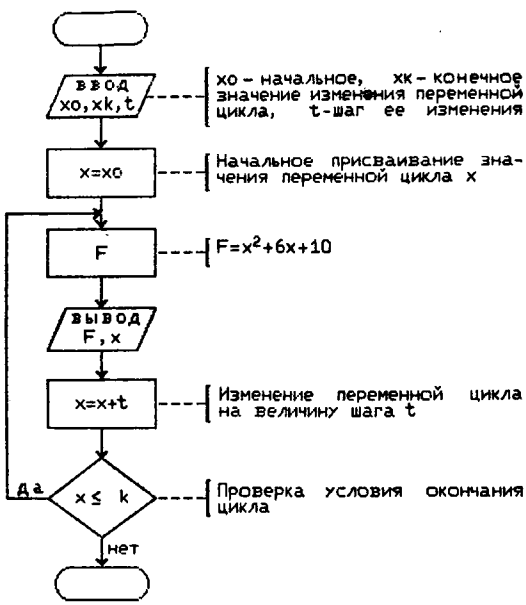


Рис. 4.6. Блок-схема алгоритма табулирования функции

ческого участка схемы закончится и управление передастся следующему за циклом блоку (в данном примере на конец блок-схемы), при этом переменная цикла сохранит свое значение (превысившее $x_к$).

Пример 4.5. Пригодность детали оценивается по размеру AF, который должен соответствовать интервалу (A—δ, A+δ). Определить, имеются ли в партии из N деталей бракованные.

Блок-схема алгоритма решения задачи показана на рис. 4.7. В данном алгоритме предусмотрен вывод сообщения в случае

обнаружения бракованной детали, когда фактический размер детали выходит за пределы, ограниченные полем допуска. Отсутствие сообщений о браке означает, что все детали являются годными.

Пример 4.6. Определить сумму и произведение N произвольных чисел (рис. 4 8).

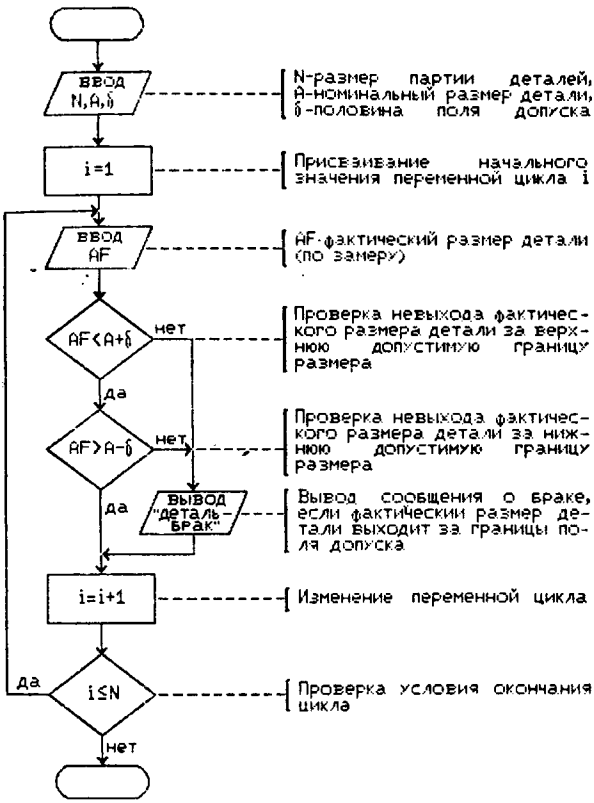


Рис. 4.7. Блок-схема алгоритма выявления бракованных деталей

Определение суммы и произведения нескольких величин требует начального присваивания переменным, обозначающим сумму и произведение (в данном примере S и P). Без такого присваивания результатов могут не соответствовать действительности из-за возможного использования этих величин каким-то другим образом до цикла.

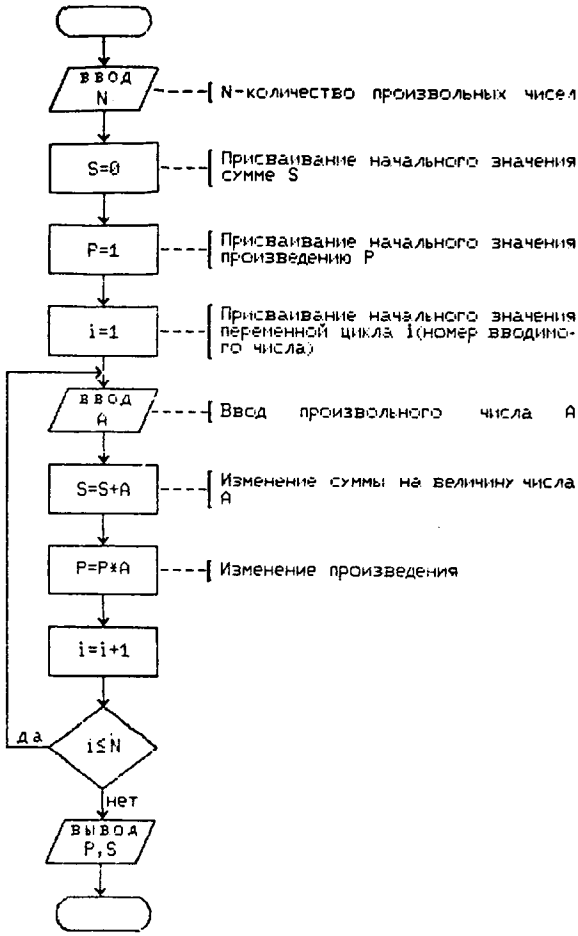


Рис. 4.8. Блок-схема алгоритма определения суммы и произведения произвольных чисел

Начальное значение для суммы равно нулю, для произведения — единице.

Как правило, рассмотренные алгоритмы используются в качестве составной части при решении различ-

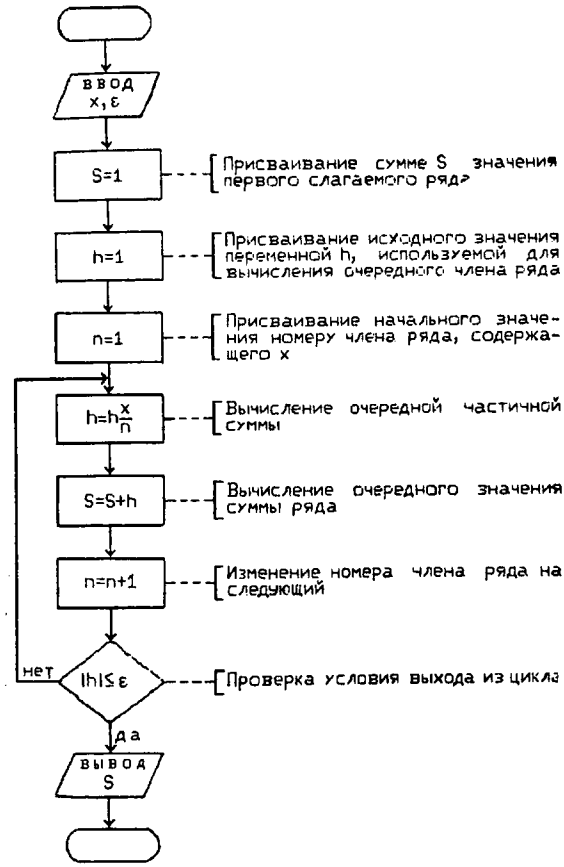


Рис. 4.9. Блок-схема итерационного алгоритма определения суммы сходящегося ряда

ных задач, связанных с определением средних величин (средней зарплаты, выработки, среднего роста, веса и т. п.), итоговых сумм, вычислением среднего геометрического и т. д. Характерной особенностью та-

ких алгоритмов является то, что в них ввод данных включается в тело цикла.

Пример 4.7. Вычислить сумму сходящегося ряда $S = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots + \frac{x^n}{n!} + \ldots$ с заданной точностью ξ (рис. 4.9).

Условием обеспечения заданной точности является достижение очередным членом ряда величины, по модулю меньшей ξ. Для определения членов ряда использования рекуррентная формула

$$h_{n+1} = h_n \frac{x}{n+1}.$$

Алгоритм является представителем класса итерационных алгоритмов, в которых каждое новое значение некоторой величины вычисляется на основе уже имеющегося предыдущего.

4.3. ПРИМЕРЫ ТИПОВЫХ АЛГОРИТМОВ ОБРАБОТКИ ОДНОМЕРНЫХ МАССИВОВ

Главной особенностью алгоритмов обработки одномерных массивов является то, что обработка массивов должна производиться поэлементно. Это означает, что надо организовать такой циклический алгоритм, который обеспечит возможность обращения к необходимым элементам массива (или массивов) и их последующую обработку. Именно по этой причине в качестве переменной цикла обычно используется индекс элементов.

В рассматриваемых ниже алгоритмах обработки массивов процессы ввода и вывода элементов массива отделены от процессов их обработки. Для многих из этих алгоритмов такое требование не является обязательным. Можно создать более компактную схему, однако при этом снижается ее понятность, а она более необходима для начинающего пользователя, чем компактность алгоритма и соответствующей программы.

Пример 4.8. Ввести в качестве элементов массива результаты замера диаметров d для N деталей. Формальная постановка задачи — ввести массив элементов $d(i)$, где $i=1, 2, ..., N$ (рис. 4.10).

Конечно, в реальных задачах после ввода массива должна быть какая-то обработка элементов. Тогда вместо последнего блока схемы должен стоять первый блок алгоритма, обрабатывающего элементы введенного массива (например, суммирующего их).

Схема вывода элементов массива аналогична схеме ввода и отличается от последней только тем, что вместо блока ввода элементов массива [ввод d(i)] стоит блок вывода [вывод d(i)], а также отсутствует блок ввода N.

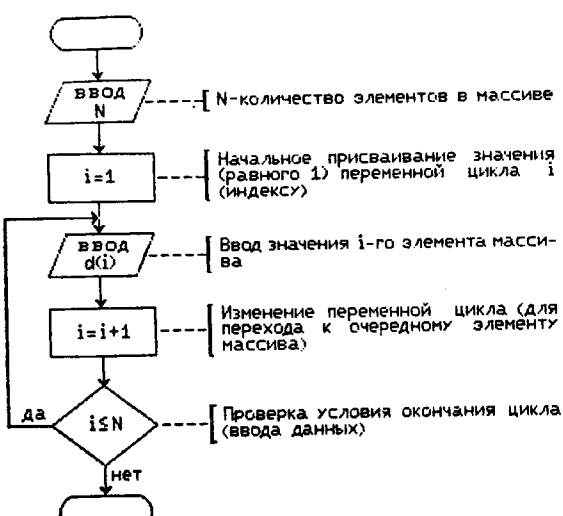


Рис. 4.10. Блок-схема алгоритма ввода значений элементов одномерного массива

Еще раз обратим внимание на тот факт, что в момент выхода из цикла переменная цикла имеет значение, превышающее максимально допустимое. Поэтому после прохождения цикла (после проверки условия $i \leqslant N$ и выхода по пути «нет») переменная имеет значение $i=N+1$, которое может быть использовано в дальнейших вычислениях.

Пример 4.9. Определить сумму S и произведение P элементов одномерного массива A(i), i=1, 2, ..., N (рис. 4.11).

Здесь рассматриваются, по-существу, две самостоятельные задачи — определение суммы и произведения элементов массива.

Объединение этих двух задач в одну выполнено с целью экономии места чисто формально. Вычеркнув все действия, связанные с величиной P (или S), можно получить алгоритм определения только суммы (или произведения).

Можно привести достаточно много задач, базирующихся на использовании данного алгоритма. К таким задачам можно отнести, например, определение:

суммарной выработки коллектива рабочих;

суммарного веса изделия, состоящего из набора определенных деталей;

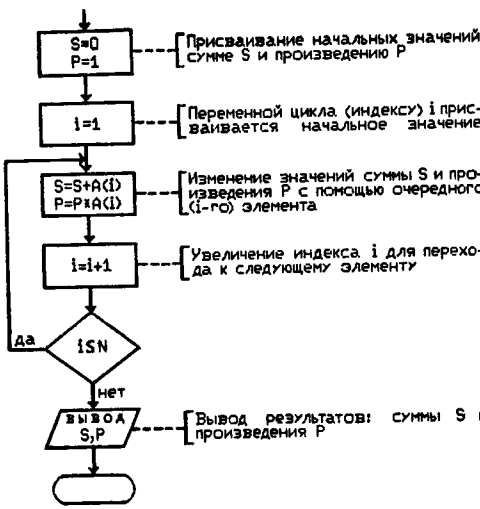


Рис. 4.11. Фрагмент блок-схемы определения суммы и произведения элементов одномерного массива (без ввода элементов массива)

суммы заработка коллектива рабочих (служащих);

коэффициента нагрузки, вычисляемого как произведение отдельных коэффициентов;

суммы баллов, набранной по результатам сессии студентами академической группы, и т. д.

Следует еще раз обратить особое внимание на то, что определение суммы или произведения (а также количества) всегда требует начального присваивания: сумма (количество) «обнуляется», а произведение принимается равным единице.

Нередко студенты делают попытку «обнулять» и произведение. Естественно, что в этом случае результирующее произведение всегда равно нулю, ибо результат умножения любого числа на нуль равен нулю.

Пример 4.10. Определить сумму и количество элементов массива A(i), где i=1,2, ..., N, удовлетворяющих некоторому заданному условию P (рис. 4.12).

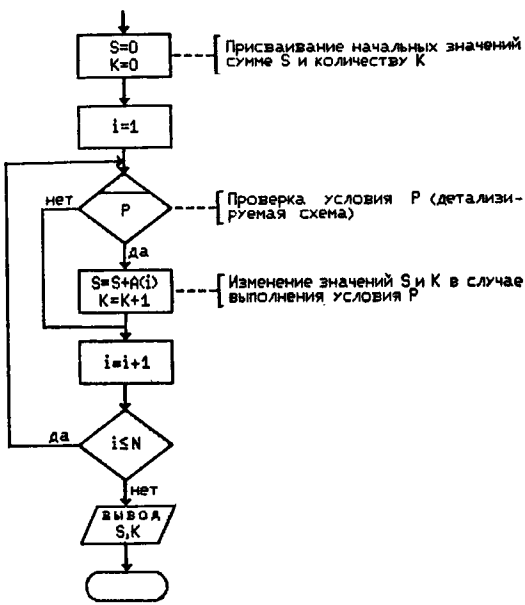


Рис. 4.12. Фрагмент блок-схемы определения суммы и количества элементов одномерного массива, удовлетворяющих условию P (без ввода элементов массива)

Сумме S и количеству элементов K в качестве начального значения присваивается нуль. Однако в случае выполнения условия P к сумме S добавляется значение очередного элемента, а к счетчику количества K добавляется единица, отмечающая факт очередного выполнения условия P.

Некоторые возможные варианты для блока проверки условия P приведены на рис. 4.13.

В реальных задачах условия P могут быть другими.

Алгоритм можно (рис. 4.13) использовать при ре-

шении самых разнообразных задач, например при определении количества:

людей, принадлежащих данной возрастной категории;

деталей, срок службы которых превысил нормативный;

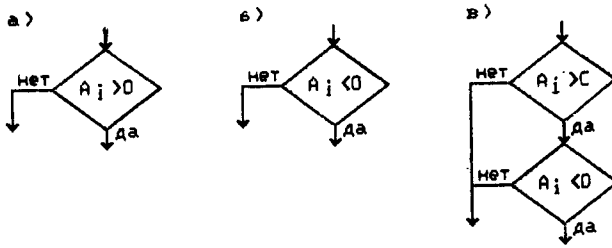


Рис. 4.13. Возможные варианты блока проверки условия P:

а — учитываются только положительные элементы; *б* — учитываются только отрицательные элементы; *в* — учитываются только элементы из интервала (C, D), концы интервала должны быть определены ранее

работников, выполнивших норму, и их среднюю заработную плату;

деталей, размер которых не соответствует допуску, и т. д.

Пример 4.11. Сформировать массив B(j), где j=1,2, ..., M, из элементов массива A(i), i=1,2, ..., N, удовлетворяющих некоторому заданному условию P.

Фрагмент блок-схемы (без ввода исходного массива A и вывода результирующего массива B) показан на рис. 4.14.

Важной особенностью этой схемы является наличие двух индексов. Один из них (индекс i) служит для обращения к элементам исходного массива, другой (индекс j) является индексом того элемента B(j) результирующего массива, которому присваивается значение A(i) при выполнении условия P. Массивы A и B совпадут только в том случае, если все элементы массива A удовлетворяют условию P. Отметим, что величина M заранее не определена.

Что касается детализации блока проверки условия P, то все, сказанное для предыдущего примера, справедливо и в данном случае.

Алгоритм (рис. 4.14) может быть использован при решении различных задач: составления списка отличников; отбора станочного оборудования для опреде-

ленных условий производства; составления ведомости замены оборудования, агрегатов, деталей вследствие исчерпания ими установленного ресурса эксплуатации; составления перечня товаров с истекшим сроком хранения и др.

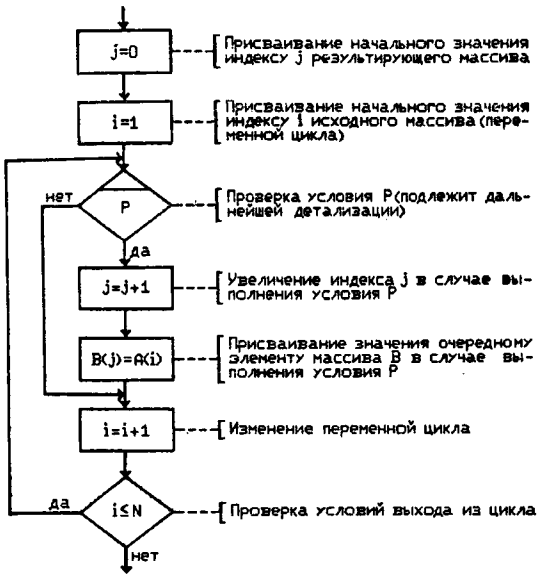


Рис. 4.14. Фрагмент блок-схемы алгоритма формирования массива из элементов исходного массива, удовлетворяющих заданному условию P

Пример 4.12. Найти значение индекса максимального (минимального) элемента массива R(i), где i = 1, 2, ..., N, и значение этого элемента (рис. 4.15).

Если в блоке сравнения очередного элемента с максимумом заменить знак больше (>) на знак меньше (<), то по данной схеме определится минимальный элемент массива с запоминанием его индекса. Если в массиве имеется несколько элементов с одним и тем же значением, равным максимальному, то запомнится индекс только первого из них.

Если условие $R(i) > M_{max}$ заменить на $R(i) \geqslant M_{max}$, то при обнаружении элементов, равных текущему значению максимума, будет запоминаться значение элементов с большим индексом. Иначе говоря, если несколько элементов массива имеют одно и то же значение и оно является максимальным, то будет запомнен индекс последнего из этих элементов. Какое из условий использовать, определяется по смыслу задачи.

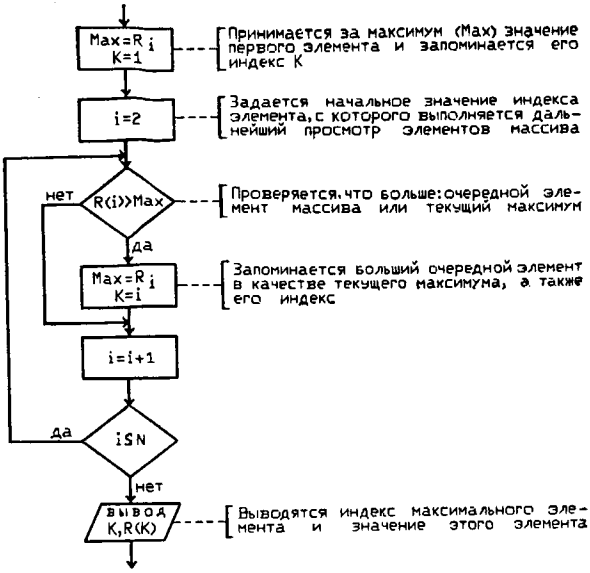


Рис. 4.15. Фрагмент блок-схемы алгоритма поиска индекса и значения максимального элемента (без ввода элементов массива)

Область применения алгоритма, приведенного на рис. 4.15, широкая. Это и определение самого лучшего по баллам студента или наиболее высокооплачиваемого сотрудника, и выявление наиболее эффективного или самого легкого (тяжелого) материала из имеющихся в наличии и т. д.

4.4. ПРИМЕРЫ АЛГОРИТМОВ СО ВЛОЖЕННЫМИ ЦИКЛАМИ

Если один цикл находится в теле другого, то он называется *вложенным* (внутренним). Внешний цикл сам может оказаться вложенным по отношению к какому-либо другому циклу. *Глубина вложения,* т. е. количество вложений циклов один в другой, определяется условиями задачи. Использование вложенных циклов позволяет компактно описывать довольно сложные алгоритмы.

На использовании вложенных циклов базируется целый класс задач обработки 2-, 3-,..., N-мерных массивов.

Пример 4.13. Сформировать дополнительный столбец для двумерного массива A, имеющего M строк и N столбцов. Элементами столбца должны быть суммы элементов соответствующих строк данного массива A (рис. 4.16).

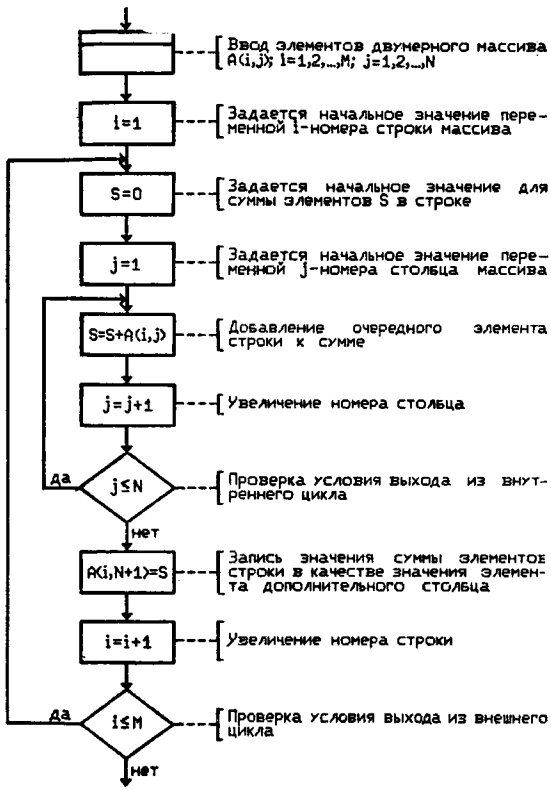


Рис. 4.16. Фрагмент блок-схемы алгоритма формирования дополнительного столбца для двумерного массива

Ввод элементов массива A также осуществляется с использованием вложенных циклов: организуется внешний цикл, изменяющий номера строк массива, внутри которого также в цикле вводятся элементы, принадлежащие строке.

Пример 4.14. Сдвинуть элементы массива A, состоящего из N элементов, вправо на M позиций; при этом M элементов из конца массива переместить в начало (циклический сдвиг элементов массива). Численный пример для N=6 и M=2 показан на рис. 4.17, *а*, *б*.

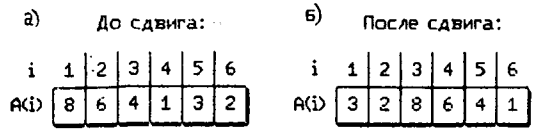


Рис. 4.17. Состояние элементов массива до и после двукратного циклического сдвига вправо

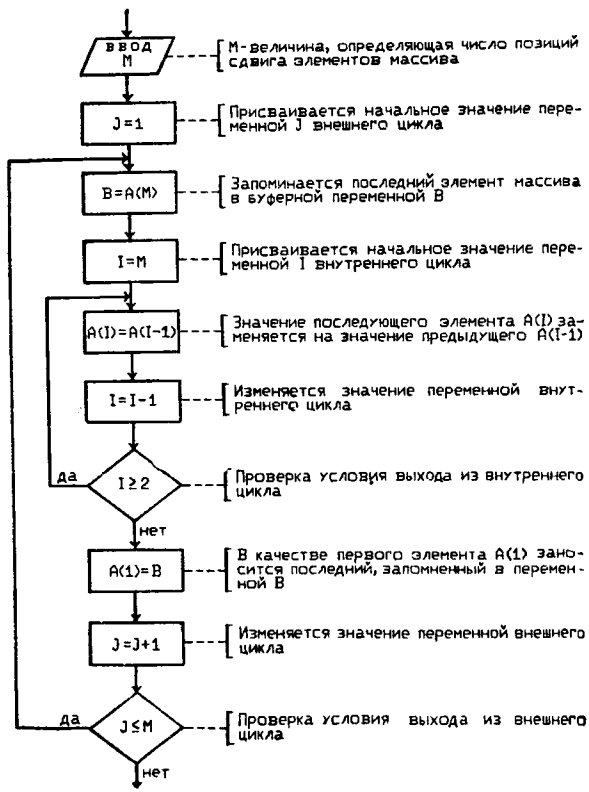


Рис. 4.18. Фрагмент блок-схемы алгоритма циклического сдвига вправо элементов одномерного массива

Фрагмент блок-схемы решения задачи (без ввода элементов исходного и вывода элементов результирующего массивов) показан на рис. 4.18.

Особенностью алгоритма является то, что перемещение элементов выполняется, начиная с конца: запоминается последний элемент, остальные элементы, начиная с предпоследнего, сдвигаются на одну позицию вправо, затем в качестве первого записывается запомненный последний элемент. Попытка начать внутренний цикл со значения переменной i=1 приведет к искажению исходного массива. Действительно, в этом случае значе-

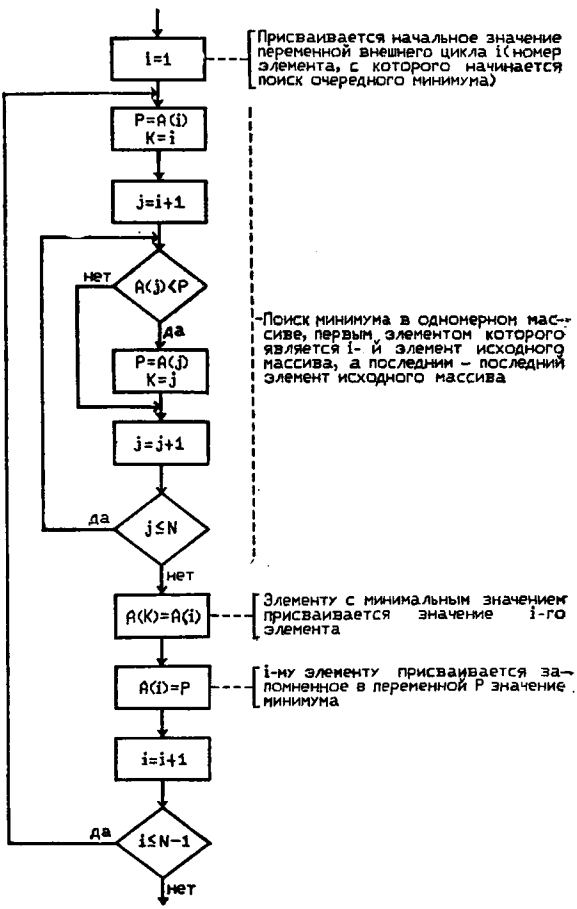


Рис. 4.19. Фрагмент блок-схемы алгоритма упорядочения элементов одномерного массива по возрастанию

ние первого элемента будет записано в качестве второго, далее, значение второго — в качестве третьего, значение третьего — в качестве четвертого и т. д. до предпоследнего элемента. Фактически это приведет к тому, что значение первого элемента исходного массива будет записано во всех элементах массива (кроме первого элемента результата).

Наглядный пример использования алгоритмов данного типа для решения прикладных задач — изменение очереди на профилактический ремонт оборудования и другие виды циклического обслуживания.

Пример 4.15. Расположить элементы массива A, имеющего N элементов, в порядке возрастания. Один из алгоритмов решения задачи представлен на рис. 4.19.

Алгоритм основан на поиске минимального элемента в массиве или в части этого массива. Вначале ищется минимальный элемент массива и меняется местами с первым элементом, затем находится минимальный элемент из оставшихся элементов (кроме первого) и меняется местами со вторым элементом. Процесс продолжается до тех пор, пока в массиве не останется один элемент; этот последний элемент является наибольшим, и он станет последним элементом упорядоченного массива.

Аналогично решается задача упорядочения массива по убыванию: для этого достаточно заменить условие проверки во внутреннем цикле с $A(i) \geqslant P$ на $A(j) < P$.

Данный алгоритм можно принять за основу, например, при решении задач составления списка:

очередности на получение жилья на предприятии в соответствии со стажем;

на распределение студентов по местам работы по сумме баллов за время обучения;

замены деталей в соответствии с временем их работы и т. д.

4.5. ВСПОМОГАТЕЛЬНЫЕ АЛГОРИТМЫ

При разработке новых алгоритмов нередко возникают ситуации, когда в разных местах алгоритма необходимо обращение к одной и той же последовательности шагов обработки данных. Для такой последовательности создают отдельный алгоритм, называемый *вспомогательным*, а в основном алгоритме предусматривают обращение к нему. В качестве вспомогательного алгоритма могут использоваться также алгоритмы, разработанные для других задач. Вспомогательный алгоритм делает структуру алгоритма

более понятной и облегчает заимствование чужого опыта и знаний.

При представлении алгоритмов с помощью блок-схем для обращения к вспомогательным алгоритмам используется блок «предопределенный процесс» (см. рис. 4.1, *е*), внутри которого должно быть записано название (имя) вспомогательного алгоритма. Это же имя должно быть записано в блоке начала самого вспомогательного алгоритма.

Другим требованием является необходимость согласования имен переменных в основном и вспомогательном алгоритмах. Значения переменных основного алгоритма, которые используются во вспомогательном алгоритме (фактические параметры), требуется присвоить соответствующим переменным (формальным параметрам) вспомогательного алгоритма, а затем значения результатов, полученных во вспомогательном алгоритме, присвоить соответствующим переменным основного алгоритма.

Всякий алгоритм, содержащий обращение к вспомогательному, сам может быть вспомогательным алгоритмом для какого-то другого алгоритма.

Использование вспомогательных алгоритмов создает предпосылки к созданию и применению библиотеки алгоритмов. При этом работа пользователя сводится к поиску библиотечных (проверенных и испытанных) алгоритмов и созданию основной программы очень простой структуры, состоящей в основном из обращений к вспомогательным библиотечным алгоритмам.

Пример 4.16. Сформировать одномерный массив A из минимальных элементов столбцов двумерного массива (матрицы) B, имеющего N строк и M столбцов.

Фрагмент алгоритма решения задачи, использующего вспомогательный алгоритм МИН, представлен на рис. 4.20.

В данном примере для вспомогательного алгоритма используются те же имена переменных, что и для основного алгоритма. В таком случае говорят, что параметры вспомогательного алгоритма являются глобальными, т. е. определенными еще до обращения к вспомогательному алгоритму.

Вообще отработке знаний и умений согласования параметров основного и вспомогательного алгоритмов необходимо уделять большое внимание, так как любая попытка применения нужного алгоритма сразу ставит перед необходимостью решения этой задачи.

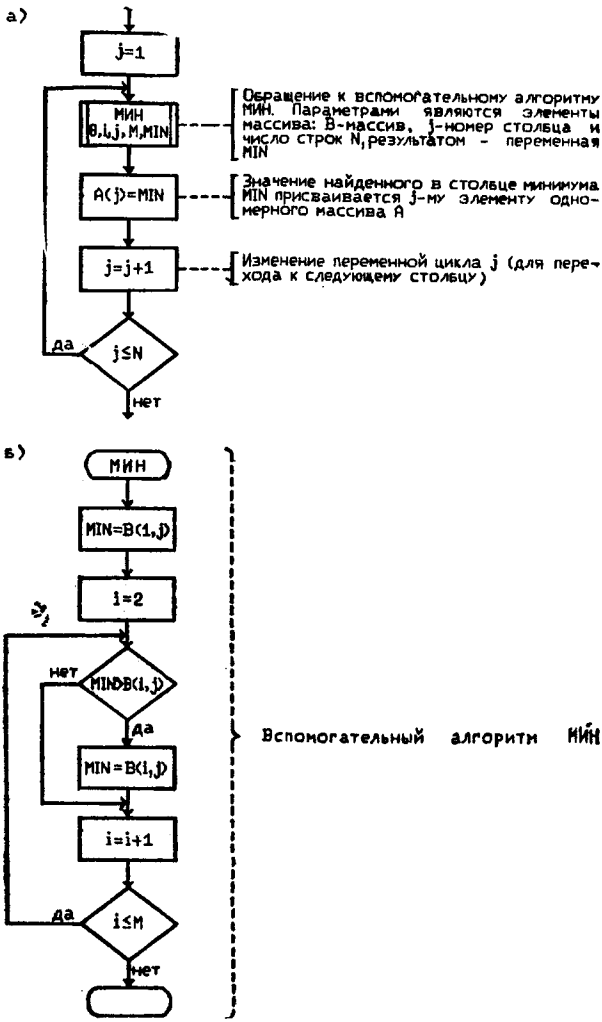


Рис. 4.20. Фрагмент блок-схемы алгоритма формирования одномерного массива из минимальных элементов столбцов двумерного массива:

а — основной алгоритм; *б* — вспомогательный алгоритм МИН

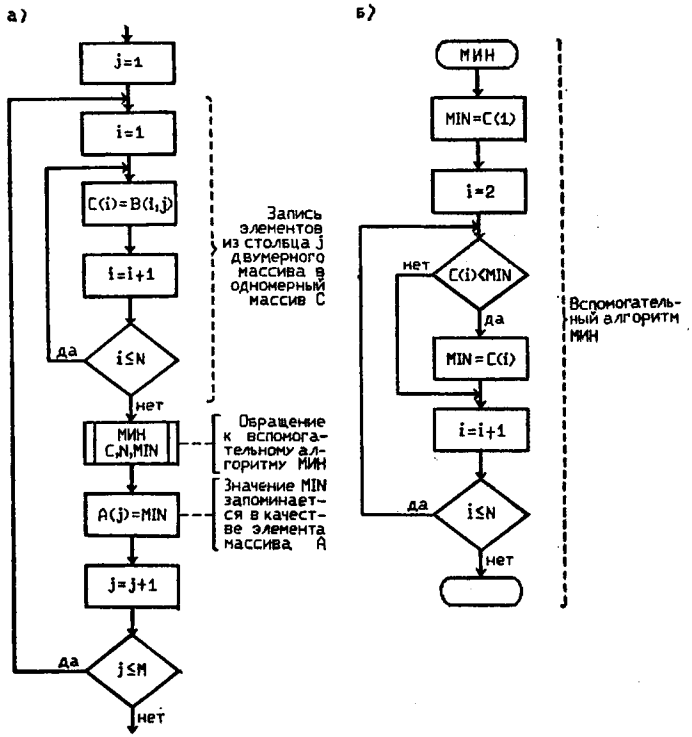


Рис. 4.21. Фрагмент блок-схемы алгоритма формирования одномерного массива из минимальных элементов столбцов двумерного массива с использованием буферного массива:

а — основной алгоритм; *б* — вспомогательный алгоритм MIN

В схеме, показанной на рис. 4.20, *б*, вспомогательный алгоритм создан специально для основного алгоритма, блок-схема которого приведена на рис. 4.20, *а*. Однако в этом вспомогательном алгоритме решается задача определения минимума для одномерного массива. Поэтому более естественным было бы использование во вспомогательном алгоритме имен элементов одномерных массивов. В то же время основной алгоритм предназначен для обработки двумерного массива, поэтому в нем должна быть предусмотрена передача значений элементов соответствующих столбцов двумерного массива основного алгоритма элементам одномерного массива вспомогательного алгоритма.

Алгоритм решения задачи для такого случая использования вспомогательного алгоритма показан на рис. 4.21.

4.6. РАЗРАБОТКА АЛГОРИТМОВ «СВЕРХУ — ВНИЗ»

Для специалистов, решающих с помощью ЭВМ те или иные задачи, разработка алгоритма решения является важнейшим делом. Существуют различные методы разработки алгоритмов (и программ), но наиболее важным является *метод пошаговой детализации* (или *метод разработки «сверху—вниз»*). При этом методе первоначально продумывается и фиксируется множество данных и результатов алгоритма без детальной проработки отдельных его частей. Тогда блок-схема любого алгоритма выглядит так, как показано на рис. 4.22.

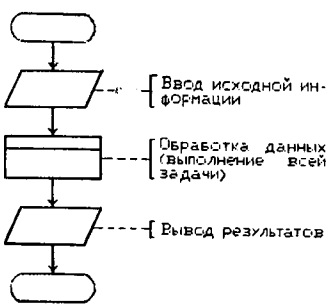


Рис. 4.22. Начальное состояние блок-схемы алгоритма при разработке методом «сверху — вниз»

Задачу разбивают на автономные части, каждая из которых существенно проще. Может оказаться необходимым повторять процесс детализации многократно, но это определяется только сложностью решаемых задач. Участки блок-схем, требующие дальнейшей детализации, обозначаются блоками «детализируемая программа», целиком заимствуемые части алгоритма обозначаются блоками «предопределенный процесс» (см. рис. 4.1).

Конечным уровнем детализации алгоритма можно считать такой, при котором в алгоритме нет действий более крупных, чем: обращение к библиотечному (вспомогательному) алгоритму; вычисление арифметического выражения и присваивание значения переменной; сравнение арифметических выражений (или переменных); ввод (вывод) данных и т. п.

Пример 4.17. Дан массив оценок успеваемости группы студентов по предмету O(i), i=1,2,..., N. Определить количество студентов, оценка которых выше среднего балла группы.

Блок-схема алгоритма решения задачи (первый этап детализации) приведена на рис. 4.23.

После первого этапа составления алгоритма получается перечень относительно крупных задач, каждая из которых (кроме последней) подлежит детализации. Блок-схема алгоритма (второй этап детализации) приведена на рис. 4.24.

Детализировать можно сразу все блоки предыдущей детализации или по частям (это определяется размером схемы). Желательно, чтобы на один лист входила или вся схема или каждая ее отдельная детализация.

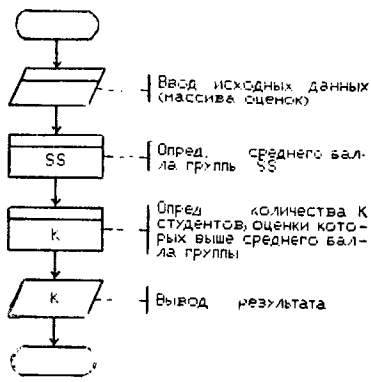


Рис. 4.23. Блок-схема алгоритма определения количества студентов, оценки которых выше среднего балла группы (I этап детализации)

В данном примере два этапа детализации алгоритма. Однако часто встречаются задачи, которые требуют три, четыре и более этапов. Блок-схема таких задач значительно превышает размеры листа. В этом случае каждую новую детализацию можно выполнять на отдельном листе, оформляя ее как самостоятельный алгоритм с именем (подобно тому, как это сделано в примере 1 § 4.5, только вместо блока «предопределенный процесс» надо использовать блок «детализируемая программа», внутри которого должно быть написано имя детализируемого алгоритма).

При разработке алгоритмов решения прикладных задач разработчики стремятся скопировать структуру алгоритма с «ручного» способа решения задачи, с «безмашинных» учебников, пособий и т. д. Однако при таком способе разработки блок-схема нередко выглядит как хитросплетенная сеть, обозреть которую (а тем более исправить) даже самому разработчику становится достаточно трудно.

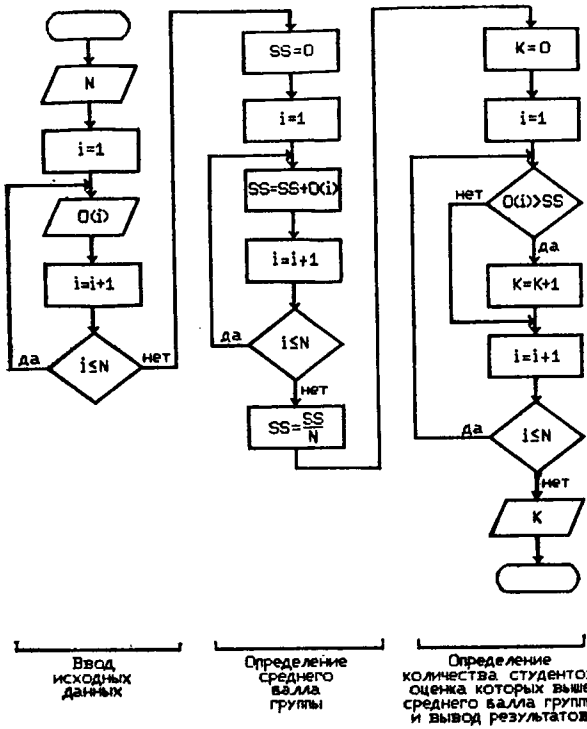


Рис. 4.24. Блок-схема алгоритма определения количества студентов, оценки которых выше среднего балла группы (II этап детализации)

Главным требованием к алгоритму, несомненно, является его работоспособность. Однако создавая алгоритм, необходимо помнить о дальнейшей работе над ним, об отладке программы, которая будет создана по этому алгоритму, а также о вероятных пользователях, которым, возможно, потребуется этот алгоритм. Поэтому одним из важнейших требований к алгоритму является его простота и понятность.

Исходя из этих требований, особенно удобным представляется при разработке алгоритмов использование основных алгоритмических структур (см. рис.

4.2). Их важной особенностью является то, что они имеют один вход и один выход и могут соединяться друг с другом в любой последовательности. Это дает наглядную и простую структуру алгоритма, по которой далее легко составить программу.

Обычно при составлении блок-схемы процесс вычислений идет сверху вниз, возвращаясь назад только

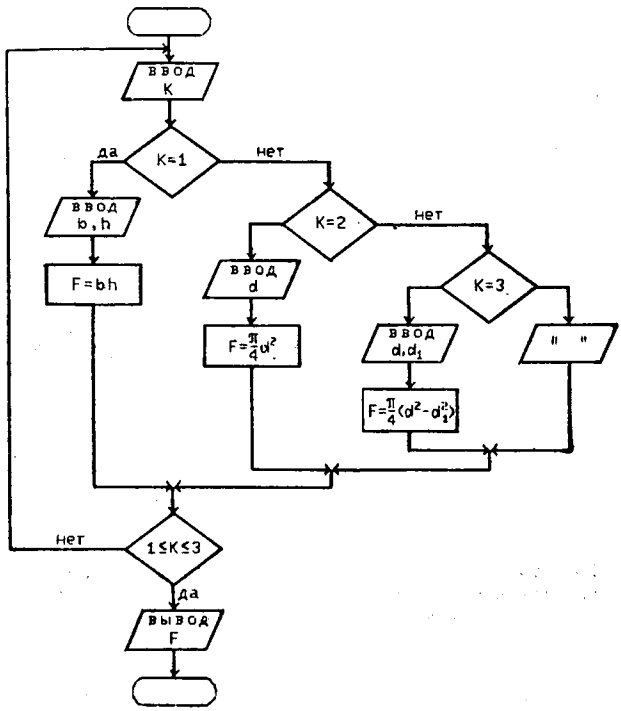


Рис. 4.25. Структурная схема определения площади плоской фигуры

в циклах, что позволяет анализировать алгоритм как обычный текст, т. е. сверху вниз.

Если технологию разработки алгоритмов «сверху—вниз» совместить с использованием только структурных схем (см. рис. 4.2), то получается новая технология, которая называется *структурным программированием сверху—вниз*. Разработанные в соответствии с ней алгоритмы обладают в некотором смысле

свойством правильности. В таких алгоритмах, как правило, меньше ошибок, хотя в целом эта технология, как и другие, не гарантирует от неточностей, связанных с невнимательностью.

Конечно, для разработки структурированного сверху—вниз алгоритма (и программы) требуется затратить больше усилий, чем для получения неструктурированного алгоритма. Однако дополнительные затраты окупаются, так как в структурированных алгоритмах и программах легче разобраться.

В заключение отметим, что все представленные в данной главе алгоритмы разработаны и представлены в соответствии с требованиями структурного подхода. Исключением является схема на рис. 4.5. Соответствующая ей структурная схема показана на рис. 4.25.

Существуют различные способы замены неструктурных алгоритмов на структурные (дублирование блоков, введение переменной состояния, метод логического признака), однако лучше всего при создании алгоритма сразу использовать только основные структурные схемы.

УПРАЖНЕНИЯ

4.1. Составить алгоритм оформления документов абитуриента при поступлении в вуз.

4.2. Разработать алгоритм управления светофором, установленным на пересечении главной и второстепенной улиц. Он должен открывать движение по второстепенной улице на 45 с, если с момента последней остановки движения по главной улице прошло не менее 2 мин и 1) если по второстепенной улице движутся автомобили (о чем говорят специальные извещатели), или 2) если пешеход нажимает кнопку управления светофором.

4.3. Разработать алгоритм, определяющий вероятность выбора годной детали (размер D в пределах допуска) из партии в N деталей.

4.4. Разработать алгоритм приготовления торта по известному рецепту.

4.5. Определить количество студентов, имеющих максимальную сумму баллов по результатам сессии, состоящей из четырех экзаменов (если имеется список студентов и набранные ими баллы на экзаменах).

Глава 5

ПРОГРАММИРОВАНИЕ АЛГОРИТМОВ ОБРАБОТКИ ЧИСЛОВОЙ И СИМВОЛЬНОЙ ИНФОРМАЦИИ

В этой главе приводится общая характеристика языков программирования. Рассматриваются основы программирования на языке БЕЙСИК MSX, способы реализации на нем основных алгоритмических структур. Приводятся примеры программных конструкций для задач с типовыми алгоритмами решения. Излагаются основы взаимодействия пользователя с компьютером.

5.1. ОБЩАЯ ХАРАКТЕРИСТИКА ЯЗЫКОВ ПРОГРАММИРОВАНИЯ

Языки программирования являются одним из средств общения пользователя с компьютером. Все они (а их насчитывается свыше тысячи) имеют сходство с естественным языком, а именно: так же, как и естественные языки, базируются на некотором множестве допустимых символов (алфавите), из которых согласно синтаксическим правилам формируются слова (лексические единицы) и предложения языка (операторы)*. И все же языки программирования могут использоваться только для общения человека с компьютером, а не человека с человеком, так как они имеют чрезвычайно скудный запас слов. Обычно словарь языка программирования насчитывает не более 500 слов (число допустимых в нем слов). Синтаксические правила языка программирования допускают лишь однозначный способ построения предложений (операторов). Эти ограничения на язык налагает специальная программа, называемая *транслятором*, которая служит переводчиком с языка программирования на язык машинных команд (компьютер понимает только их). Более точно под *трансляцией* понимаются следующие действия компьютера:

синтаксический разбор текста: разделение текста

---

* Совокупность операторов (команд), описывающих алгоритм решения задачи, называют *программой*.

программы на отдельные лексические единицы (слова);

распределение памяти машины между переменными, используемыми в программе;

распознавание отдельных конструкций языка (операторов) и их замена последовательностями машинных команд.

Язык и транслятор с этого языка называют *системой.* Различают два типа систем: компилирующие и интерпретирующие.

Компилирующая (собирающая) система. Эта система предназначена для обработки текстов программ, распадающихся на несколько самостоятельных частей. Она позволяет программисту строить свою программу и из маленьких «кирпичиков» (операторов) и из больших блоков (подпрограмм), представляющих собой решения самостоятельных (например, типовых) задач.

Компилируя программу, компьютер выполняет следующие операции:

каждую часть исходной программы при помощи соответствующих программ-трансляторов переводит на язык машинных команд (программа, записанная в таком виде, называется *объектным модулем*);

при помощи программы-редактора связей объединяет все объектные модули в единую программу (называемую *загрузочным модулем);*

при помощи программы-загрузчика размещает программу пользователя в памяти и выполняет ее.

Процессом компиляции можно управлять. Управляющие команды записываются на специальном языке, называемом языком управления заданиями, используемом в операционной системе компьютера. В операционную систему входят транслятор, редактор связей, загрузчик.

Интерпретирующая система. Система предназначена для обработки текстов программ в режиме диалога с пользователем. При этом текст программы подготавливается компьютером к исполнению пооператорно по мере поступления записанных в программе операторов (команд) в компьютер. Диалоговый режим часто предоставляет пользователю возможность проверить правильность написанных операторов до

полного завершения программы, что значительно упрощает процесс обучения языку.

При решении научных и технических задач, требующих сложных многократных расчетов, более эффективны системы компилирующего типа (например, с языков ПАСКАЛЬ, ФОРТРАН, СИ). Однако для тех, кто делает первые шаги в программировании, целесообразно начать изучение языков, входящих в системы интерпретирующего типа (например, с языка БЕЙСИК, РАПИРА, ФОКАЛ), так как овладеть умением пользоваться структурными единицами, составляющими основу любого языка, проще при диалоговом программировании.

Рассмотрим язык интерпретирующего типа БЕЙСИК MSX, реализованный на ПЭВМ «Ямаха» и ряде других компьютеров, предназначенный для начинающих пользователей (Beginner's All — purpose Simbolic Instruction Code — многоцелевой язык символических инструкций для начинающих). Отметим, что сейчас имеется до десятка вариантов этого языка.

5.2. ПРЕДСТАВЛЕНИЕ ДАННЫХ В ЯЗЫКЕ БЕЙСИК

Данные. В языке БЕЙСИК допускается использовать следующие данные: константы, переменные и массивы. Они имеют одно назначение — фиксировать место в памяти под конкретные данные, но выполняют это назначение по-разному.

К о н с т а н т а характеризуется неизменностью своего значения. Поэтому можно предположить, что фиксирование памяти под константу осуществляется ее значением. Напротив, п е р е м е н н а я может изменять свое значение. В связи с этим для фиксирования места в памяти используется *имя переменной* — это одно- или двухсимвольная последовательность больших (прописных) латинских букв и цифр, начинающаяся с буквы. Например, имена A, B, N, Z9, BZ — правильны, а имена 5A и Z/ — неверны (первое начинается с цифры, а второе содержит символ операции).

БЕЙСИК MSX допускает буквенно-цифровые имена произвольной длины. Однако идентификация (определение) имени интерпретатором производится по первым двум символам. Этой особенностью нужно пользоваться осторожно, так как в программу могут

проникнуть трудно обнаруживаемые ошибки, связанные с фактическим совпадением разных имен, поэтому лучше всего ограничиваться именами, содержащими не более двух символов. Кроме того, эта версия языка БЕЙСИК допускает использование маленьких (строчных) букв, не отличая их от прописных.

Если, написав значение константы, фиксировать место в памяти и одновременно задать ее значение, то при вводе имени переменной фиксируется только место переменной в памяти, а ее значение остается неопределенным. Поэтому перед тем, как начать использовать переменную, необходимо каким-либо образом придать значение этой переменной.

Для массива характерны те же особенности, что и для переменной: изменяемость значения и фиксирование места в памяти под конкретные данные при помощи имени. Имя массива строится аналогично имени переменной. Но в отличие от переменной (далее будем называть ее также простой переменной) имя массива дается области памяти, где хранятся все элементы массива. Отдельными элементами массива можно пользоваться так же, как и простыми переменными.

Для обращения к элементам массива используется имя (имя элемента массива), построенное из имени массива и индекса, который записывается в круглых скобках сразу после имени массива. Например, имеется массив с именем G. Тогда G(1) — первый элемент массива G, G(10) — десятый элемент массива G и т. д.

Если в качестве индекса элемента массива ввести переменную (записать на месте индекса в имени элемента массива имя переменной), то при помощи такого имени можно обращаться уже к любому элементу массива (к какому конкретно, определяет значение переменного индекса).

При использовании индексированных переменных с переменным индексом нужно помнить, что они требуют предварительного задания уже двух значений: индекса и переменной. Кроме того, появление в программе индексированной переменной всегда порождает еще одну неопределенность. Она связана с тем, что компьютер не знает, сколько места в памяти ему следует отвести тому или иному массиву. Закрепле-

ние за именем массива необходимого пользователю объема памяти называется *описанием массива.*

Описание массива выполняет оператор, начинающийся со слова DIM (сокращение слова DIMension— размер), имеющий в простейшем случае такую структуру:

DIM <имя массива> (<максимальное значение индекса>),

где имя массива выбирается по правилам построения имен простых переменных; максимальное значение индекса указывает максимально допустимое в программе значение индекса (оно должно быть неотрицательным и может быть определено константой, переменной или арифметическим выражением).

В операторе DIM можно описать сразу несколько массивов, разделив их описания запятыми.

Для описания многомерных массивов в скобках после имени массива через запятую указывается несколько максимальных значений индексов. Количество измерений ограничено числом 255.

Если в языке БЕЙСИК MSX массив не описан в операторе DIM, то по умолчанию максимальное значение индекса по каждому измерению принимается равным 10. Минимальное значение индекса всегда равно 0. Местоположение оператора DIM в программе произвольно. Однако он всегда должен предшествовать использованию соответствующих переменных. Поэтому для удобства работы с программой рекомендуется все операторы DIM группировать в начале программы.

Примеры описания массивов:

1) DIM A(50), Z(6) — описывает два одномерных массива, состоящих из 51 и 7 элементов соответственно.

2) DIM D(10, 20) — описывает двумерный массив, содержащий в каждом из 11 строк по 21 элементу (итого 231 элемент).

3) DIM A(N), B(2×N) — описывает два одномерных массива из N+1 и из (2×N+1) элементов. Придавая N различные значения, при помощи указанного оператора можно описывать одномерные массивы любой длины.

Типы данных. В языке БЕЙСИК MSX предусмот-

рены следующие типы данных: целые, вещественные одинарной и двойной точности, строковые.

Целая константа — это число из интервала от —32768 до 32767. Для хранения каждой константы в памяти отводится 2 байта. Она записывается обычным образом: последовательность цифр со знаком или без него.

Вещественная константа одинарной точности — это десятичное число, записанное в естественной или экспоненциальной форме. Точка в записи служит разделителем дробной и целой частей числа. В записи числа допускается шесть цифр. Таким образом, десятичное число в естественной форме — это число из интервала от —999999 до —0.00001 или от 0.00001 до 999999. Десятичное число в экспоненциальной форме представляет собой величину вида KEm, где K — десятичное число в естественной форме, m — целое число, E — число «10». Запись KEm в языке БЕЙСИК обозначает математическую запись числа $K \cdot 10^m$ (например, запись 5.67E—3 есть число $5{,}67 \cdot 10^{-3}$ или 0.00567, а запись 5.67E3 — число $5{,}67 \cdot 10^3$ или 5670). Десятичное число в экспоненциальной форме используется либо для сокращения записи (см. приведенные примеры), либо для представления чисел, выходящих из интервала допустимых значений десятичных чисел с фиксированной точкой. Допустимый диапазон изменения десятичного числа в экспоненциальной форме — от 10E—64 до 10E+63 для положительных чисел и от —10E+63 и до —10E—64 для отрицательных чисел.

Для хранения вещественной константы одинарной точности требуется 4 байта памяти.

Вещественная константа двойной точности — это десятичное число, в записи которого может использоваться 14 разрядов. Таким образом, диапазон десятичных чисел в естественной записи от —99999999999999 до —0.0000000000001 для отрицательных чисел и от 0.0000000000001 до 99999999999999 для положительных чисел. Диапазон изменения десятичных чисел с двойной точностью совпадает с диапазоном изменения соответствующих чисел одинарной точности. При записи десятичного числа двойной точности в экспоненциальной форме вместо символа E используется символ D.

Для хранения вещественной константы двойной точности требуется 8 байт памяти.

Строковая (или символьная) константа — это последовательность символов (русских и латинских букв, цифр, знаков препинания и т. д.), заключенная в кавычки. Строковая константа может содержать от 0 до 255 символов. Строковая константа, не содержащая символов, называется пустой, ее записывают так: "". Не путать с "␣"! Эта константа не пустая, она содержит один символ — пробел).

Для размещения строковой константы в памяти требуется по одному байту на каждый ее символ.

Тип переменной. Тип переменной (массива, переменной с индексом) определяется именем. Существует два способа назначения типов переменных.

Первый способ: тип переменной указывается знаками «%», «!», «#» и «$» после имени:

% — переменная целочисленного типа;

! — переменная вещественного типа одинарной точности;

— переменная вещественного типа с двойной точностью;

$ — переменная строкового типа.

Например, имя A% определяет переменную целочисленного типа, а имя A$ — строкового типа.

Второй способ: объявление типа специальным оператором DEF*, который имеет вид:

DEF INT <список начальных букв имени> — для целочисленного типа;

DEF SNG <список начальных букв имен> — для вещественного типа с одинарной точностью;

DEF DBL <список начальных букв имен> — для вещественного типа с двойной точностью;

DEF STR <список начальных букв имен> — для строкового типа.

Например, операторы

```
DEF INT I, K
DEF DBL D
DEF STR S
```

объявляют, что все переменные, имена которых начинаются с букв I и K, — целочисленные, с буквы D —

* В дальнейшем будем именовать операторы по характерным (ключевым) словам, которые, конечно, не исчерпывают все синтаксические единицы, из которых оператор состоит.

вещественные двойной точности, с буквы S — строковые

Явное указание типа знаками %, !, # и $ после имени переменной перекрывает объявление типа при помощи оператора DEF. По умолчанию (знак типа после имени переменной не введен, начальная буква имени не объявлена оператором DEF) переменная считается вещественной с двойной точностью.

5.3. ОСНОВНЫЕ ОПЕРАЦИИ С ДАННЫМИ

Процесс обработки информации сводится к выполнению некоторых операций с данными. К числу таких операций относятся: ввод и вывод, вычисление значений арифметических выражений для числовых величин, значений для булевых величин, определение значений функций, операция присваивания и др. Рассмотрим основные операции с данными БЕЙСИКа, начиная с вычисления значений арифметических выражений.

Арифметическое выражение. Под арифметическим выражением будем понимать обычное алгебраическое выражение (дробь, функцию и т. д.), записанное на

Таблица 5.1. **Перечень операций в языке БЕЙСИК в порядке убывания приоритета**

Операции	Обозначение операций в БЕЙСИКе
Скобки	( )
Функции	См. рис. 5.3
Возведение в степень	^
Умножение, деление	*, /
Деление нацело	\
Определение остатка от деления нацело	MOD
Сложение, вычитание	+, −
Операции сравнения:	Имеют одинаковый приоритет
больше, меньше, равно	>, <, =
не равно	< > или > <
не меньше	> = или = >
не больше	< = или = <
Логические операции:	
не	NOT
и	AND
или	OR

языке БЕЙСИК. В арифметическое выражение могут входить числовые константы, числовые простые и индексированные переменные (элементы массива), функции с числовыми значениями (стандартные для языка или введенные программистом, см. § 5.7). Они записываются в одну строку и соединяются знаками арифметических операций и скобками.

Порядок вычислений в арифметическом выражении определяется приоритетом входящих в это выражение операций. В первую очередь выполняются операции, имеющие самый высокий приоритет. Относительные приоритеты операций в порядке убывания приведены в табл. 5.1. Там же указано их условное обозначение на языке БЕЙСИК.

Операции с равными приоритетами выполняются по порядку слева направо.

Приведем несколько примеров арифметических выражений.

Алгебраическое выражение	Арифметическое выражение на языке БЕЙСИК
a	A
1,25	1,25
$\frac{a+b}{c+d}$	(A + B)/(C + D)
$\frac{x}{y \cdot z}$	X/(Y * Z)
$\frac{1}{1-1/x}$	1/(1—1/(1—1/X))

Строковые выражения. Арифметические операции, рассмотренные выше, к строковым данным не применимы. Исключение составляет операция «+» — сложение. Однако ее действие приводит не к сложению строковых данных, а к их слиянию. Это действие принято называть *конкатенацией*. Например, действие конкатенации над константами **«ка»** и **«бан»** приведет к результату **«кабан»**, а над константами **«бан»** и **«ка»** — к результату **«банка»**, т. е. результат зависит от порядка следования строковых величин.

Логические выражения. Рассмотрим еще один тип данных, порождающих другой вид операций. Это бу-

левские данные. Как известно (см. гл. 3), булевские данные могут принимать лишь два значения (логические значения): истину (TRUE) или ложь (FALSE). Поэтому выражения, служащие для вычисления этих значений, принято называть *логическими выражениями.*

В языке БЕЙСИК булевские данные фигурируют только в логических выражениях, которые собираются из условных выражений при помощи логических операций NOT (не), AND (и), OR (или). В свою очередь, условное выражение представляет собой пару арифметических или строковых выражений, соединенную операциями сравнения (табл. 5.1).

Примеры записи логических выражений:

Математическая запись	Запись на языке БЕЙСИК
$a \geqslant b$	A >= B
$0 \leqslant x \leqslant 2$	0 <= X AND X <= 2
$x > 2$ и $x \neq 5$	X > 2 AND X <> 5
$x > 2$ или $x < 0$	X > 2 OR X < 0
$x \neq 5$	NOT (X = 5)

Напомним, что при сравнении строковых данных старшинство символов определяется кодом ASCII.

Присваивание значений переменным. Для того чтобы переменные величины получили определенные значения, эти значения должны быть либо введены, либо получены как результат выполнения каких-либо операций. В последнем случае значения присваиваются переменным с помощью специального оператора присваивания.

Общая форма записи оператора присваивания:

LET <переменная> = <выражение>,

где <переменная> — имя простой или индексированной переменной, которой будет присвоено значение выражения; <выражение> — арифметическое или строковое выражение. Дает значение, тип которого должен соответствовать типу <переменной>. Все данные, входящие в состав <выражения>, должны к моменту вычисления иметь определенные значения. В противном случае <выражение> будет вычислено при нулевых значениях этих данных.

При записи оператора присваивания слово LET можно опускать.

Примеры оператора присваивания:

A=3.1415 (вещественная переменная двойной точности A получит приближенное значение известной константы π).

X!=SIN(A) вещественная переменная одинарной точности получит значение sin A, при этом «лишние» разряды будут отброшены).

A$-«СТРОКА» (строковая переменная получит значение константы СТРОКА; переменные A и A$ в БЕЙСИКе MSX различны, т. е. под них отводят разные области памяти).

N%=N%+1 (целая переменная N получит увеличенное на единицу старое значение).

A(2)=(A2)*2 (индексированная переменная двойной точности получит значение, в 2 раза превышающее старое значение).

Два последних примера показывают, что при записи оператора присваивания знак «—» обозначает не операцию сравнения «равно», а действие присваивания: вычисление правой части до того, как результат этого вычисления скопируется в левую часть.

Ввод данных. Придавать значения переменным при помощи оператора присваивания не всегда удобно. Например, если возникло желание произвести вычисления по программе при других значениях переменных, то придется изменять программу. И это необходимо будет делать всякий раз, когда мы хотим произвести вычисления при других значениях переменных.

Поэтому при многократном исполнении программы с различными наборами исходных данных целесообразно использовать *оператор ввода данных* — оператор диалогового ввода INPUT, общая форма записи которого

INPUT" <строковая константа>"; <список переменных>

При этом часть «<строковая константа>»; может быть опущена.

Оператор INPUT обеспечивает ввод данных с клавиатуры. Исполнение программы приостанавливается, на дисплее компьютера появляется приглашение к вводу данных в виде текста <строковой константы>

со знаком вопроса. Значение <строковой константы> может быть произвольным (в частности, пустым). <Список переменных> — имена переменных (простых или индексированных), разделенные запятыми. При наборе на клавиатуре значения данных должны быть разделены запятыми, тип вводимых данных должен соответствовать типу переменных в <списке переменных>.

Нарушение соответствия типов переменной и константы может привести к необходимости повторного набора. Если данных введено меньше, чем предписывает оператор INPUT, то компьютер будет повторять приглашение к вводу (в виде двух знаков вопроса) до тех пор, пока все переменные <списка переменных> не получат соответствующие значения. Если значений данных введено больше, чем нужно, то компьютер проигнорирует лишние значения, сделав при этом соответствующее сообщение. Набор констант всегда завершается нажатием клавиши возврата каретки.

Например при исполнении оператора

INPUT" введите A, B"; A, B

на первой из свободных строк дисплея появится сообщение:

Введите A, B? (компьютер ожидает набора двух чисел через запятую и нажатия клавиши возврата каретки).

При этом возможен следующий диалог:

— Введите A, B? 5 (нажатие клавиши возврата каретки).
—??—2 (нажатие клавиши возврата каретки)

или

Введите A, B? 5, — 2 (нажатие клавиши возврата каретки)

В обоих случаях A получит значение 5, а B — значение — 2.

При вводе строковых данных в набираемой на клавиатуре строковой величине может встретиться запятая. Оператор INPUT воспримет ее как разделитель между величинами (списка вводимых величин). В результате введется не вся информация, а лишь только ее часть до запятой. Чтобы обойти эту проблему, в языке БЕЙСИК предусмотрена модификация оператора INPUT, предназначенная специально для ввода строковых данных независимо от их содержания. Эта модификация оператора ввода выглядит так:

LINE INPUT <строковая переменная>

Ввод данных списком с помощью ввода констант. Иногда, например, при отладке программы приходится исполнять программу многократно с одними и теми же исходными данными. Если данных много, то процедура ввода с клавиатуры отнимает много времени. В этом случае целесообразно на время отладки заменить оператор INPUT операторами ввода данных списком READ и DATA:

```
READ <список переменных>
DATA <список констант>
```

Оператор READ последовательно присваивает очередные значения <списка констант> оператора DATA переменным, имена которых через запятую указаны в <списке переменных>.

В операторе DATA константы записываются через запятую. Тип константы должен соответствовать типу переменных. Строковую константу можно записывать без кавычек.

При выполнении оператора READ компьютер устанавливает связь между операторами READ и DATA, характер которой можно пояснить с помощью указателя. Указатель всегда показывает очередную константу в операторе DATA, подлежащую вводу. Если указатель достиг конца списка данных в операторе DATA, а в операторе READ список не исчерпан, то либо указатель переместится в начало списка данных следующего оператора DATA, либо, если такового нет, выдает сообщение о нехватке данных (излишек данных не считается ошибкой!). Положением указателя можно управлять при помощи оператора RESTORE. Общий вид оператора

```
RESTORE <номер строки>,
```

где <номер строки> указывает на оператор DATA, в начало списка констант которого необходимо перевести указатель (см. § 5.4).

При записи оператора RESTORE параметр <номер строки> можно опустить. В этом случае указатель будет перенесен в начало списка констант первого в программе оператора DATA.

Операторы DATA можно размещать в любом месте программы. Однако для удобства работы с про-

граммой рекомендуется операторы DATA группировать вместе в конце программы.

Вывод данных. Для вывода вычисленных значений переменных на дисплей (на принтер) используется оператор PRINT (LPRINT)

PRINT <список вывода>,

где <список вывода> содержит имена переменных, константы, арифметические выражения, функции управления оператором PRINT.

Разделителями элементов <списка вывода> являются запятая «,» или точка с запятой «;».

Запятая вызывает переход к новой зоне для печатания следующего элемента <списка вывода>. На ПЭВМ «ЯМАХА» экран дисплея разбит на 24 строки; в каждой строке 2 зоны по 16 позиций; зоны начинаются с нулевой и 15-й позиции.

Точка с запятой вызывает печатание следующего элемента <списка вывода> вплотную к предыдущему. Если <список вывода> завершается запятой или точкой с запятой, то следующий оператор PRINT продолжит вывод на ту же строку дисплея, на которую выводил информацию предыдущий оператор PRINT (завершен перевод строки). Оператор с пустым списком вывода, т. е. оператор PRINT, вызывает переход к новой строке.

Функциями управления оператором PRINT являются: TAB (<позиция>), SPACE$ STRING (см. табл. 5.4). Функция TAB указывает позицию, начиная с которой будет напечатан следующий элемент <списка вывода> оператора PRINT. Например, операторы (пусть значение x=10, а y=20)

```
PRINT «x=»; TAB(8); x; «первое слагаемое»
PRINT «y=»; TAB(8); y; «второе слагаемое»
PRINT «сумма=»; x+y
```

напечатают:

```
x=10 первое слагаемое
y=20 второе слагаемое
сумма=30
```

Печатание списков таблиц, финансовых документов, бухгалтерских балансов и т. п., когда на вывод каждого значения отводится строго фиксированное

Таблица 5.2. Символы описания формата

Описание формата	Действие для числовых данных	Действие для строковых данных
#	Вывод одной цифры или нуля или пробела	
.	Вывод десятичной точки	
,	Вывод запятой через каждые три позиции слева от десятичной точки	
+ или —	Может быть указан до и после # . В этой позиции будет напечатан знак числа	
ΔΔΔΔ	Означает поле для вывода экспоненты при печатании числа с плавающей точкой	
*	Вывод * во всех неиспользуемых позициях слева от десятичной точки	
\\	—	Две обратные косые черты с N пробелами между ними — вывод N+2 первых символов строки
!	—	Вывод первого символа строки

число позиций, обеспечить стандартным оператором PRINT не удается. Для этого можно использовать оператор PRINT USING:

PRINT USING «<формат>»; <список вывода>,

где <формат> — группа символов, описывающая формат вывода элементов <списка вывода>. Действие <формата> распространяется на все элементы <списка вывода> оператора. Символы описания <формата> приведены в табл. 5.2.

Примеры использования форматов (A-12.3456).

Форматная печать	Результат вывода по формату
PRINT USING «##»; A	12
PRINT USING «##. ##»; A	12.35

```
PRINT USING «—##.##»; A          +12.35
PRINT USING «#. #^^^^»; A        1.235E+01
PRINT USING «*###. ##»; A        *12.35
PRINT USING «//»; «СТРОКА»       СТРОКА
```

Если при печати числа по формату оно выходит за рамки формата, то перед числом выводится знак %.

5.4. БЕЙСИК-ПРОГРАММА

Программа на языке БЕЙСИК состоит из последовательности строк, пронумерованных от 0 до 65535.

В одной строке программы можно записать несколько операторов. В этом случае операторы отделяются друг от друга двоеточием. Порядок исполнения операторов — слева направо. Максимальная длина строки 255 (на экране дисплея такая строка будет занимать около 7 экранных строк) символов. За счет записи нескольких операторов в строке можно экономить память или приводить программу к структурному виду. Однако программа, записанная в такой сжатой форме, плохо читается. Поэтому лучше каждый оператор записывать в отдельной строке.

Для улучшения читаемости программы можно в любом месте программы размещать строки-комментарии вида:

REM <комментарий>

В комментарии допустим любой текст, в том числе и ключевые слова языка БЕЙСИК. При исполнении программы оператор игнорируется.

Программа 5.1. **Вычисление площади круга**

```
10 REM первая программа
20 REM ввод данных
30 P = 3.141159
40 INPUT "введите значения радиуса"; R
50 REM вычисление площади круга по радиусу
60 A = P * R ^ 2
70 REM вывод данных
80 PRINT "радиус"; "площадь"
90 PRINT TAB (3); R; TAB (10); A
100 END
```

Вместо ключевого слова REM допустимо ставить просто кавычку (апостроф), интерпретатор воспринимает ее как сокращение слова REM.

Оператор END завершает исполнение программы 5.1. Он обычно является последним оператором программы.

Чтобы обеспечить возможность добавления новых строк в уже написанную программу, обычно номера строк вводимых программ указываются с некоторым интервалом (например, 5 или 10). Это позволяет модифицировать имеющиеся программы, не переписывая всю программу заново, а только вставляя пропущенные строки, используя свободные номера строк.

5.5. РЕАЛИЗАЦИЯ ОСНОВНЫХ АЛГОРИТМИЧЕСКИХ СТРУКТУР

Большое значение при разработке программы имеет ее ясность и простота. Наилучшим образом последние два качества обеспечиваются в том случае, когда программа разрабатывается на основе использования типовых (линейной, разветвляющейся или циклической) алгоритмических структур, описанных в гл. 4. Опишем средства языка БЕЙСИК для реализации этих структур.

Линейная структура. Любая программа представляет собой последовательность пронумерованных строк. Исполнение программы всегда начинается со строки с наименьшим номером, а заканчивается строкой с оператором END, которая обычно имеет в программе самый большой номер, т. е. является последней строкой программы. Такой характер выполнения может создать впечатление о том, что любая программа имеет линейную структуру. На самом деле это не так. Программа на языке БЕЙСИК может содержать и разветвляющиеся и циклические структуры. Они описываются операторами передачи управления и операторами цикла, которые изменяют естественный порядок выполнения строк в программе от строки с меньшим номером к строке с большим номером. Остальные операторы порождают программы только линейной структуры. В частности, к ним относятся рассмотренные выше операторы INPUT, READ, LET, PRINT.

Программа 5.2. **Вычисление площади треугольника**

```
10 REM Текст программы к примеру на рис.4.3
20 INPUT "Введите значения величин основания и высоты
   треугольника A и H"; A,H
30 S = A * H/2
40 PRINT "Площадь треугольника S=";S",при A=";A;" и
   H = ",H
50 END
60 REM Эта программа исполняется в строгом соответствии
   с возрастанием номеров строк
70 REM Если в одной строке записано несколько
   операторов, они исполняются в порядке их следования
   слева направо.
```

Разветвляющаяся структура. Для программирования разветвления существует три оператора передачи управления: GOTO (безусловной передачи управления или перехода), IF—THEN—ELSE (условной передачи управления) и ON—GOTO (выбора).

Общая форма записи оператора GOTO:

GOTO <номер строки>

Он передает управление строке с номером, указанным параметром <номер строки>. В качестве параметра <номер строки> допускается использовать только целочисленные значения данных, записанных в виде констант.

Оператор условной передачи управления IF—THEN—ELSE имеет более сложную структуру:

IF <логическое выражение> THEN <группа операторов 1>
ELSE <группа операторов 2>

Описанное ранее <логическое выражение> может быть либо истинным, либо ложным. Если <логическое выражение > истинно, то выполняется <группа операторов 1> и затем следующая строка. В противном случае будет исполнена <группа операторов 2> и затем следующая строка. В эти группы могут входить один или несколько любых операторов (в последнем случае операторы отделятся друг от друга двоеточием). Количество операторов в каждой группе ограничивается только общей длиной строки. Напомним, она

вместе с номером строки не должна превышать 255 символов.

<Группа операторов 2> в операторе IF—THEN—ELSE может отсутствовать. Оператор IF—THEN—ELSE приобретает вид

IF <логическое выражение> THEN <группа операторов 1>

В этом случае при ложном исходе логического выражения управление передается строке, следующей за строкой с оператором IF—THEN. Такая конструкция соответствует одной из форм разветвляющейся алгоритмической структуры (программа 5.3).

Программа 5.3

```
10 REM Программа для примера 1 на рис.4.4
20 INPUT "Введите значение числа B"; B
30 MD = -B
40 IF B>0 THEN MD=B
50 PRINT "Модуль числа"; B;"равен"; MD
60 END
```

В группах операторов (в одной или в обеих) могут снова использоваться операторы IF—THEN—ELSE (программа 5.4).

Программа 5.4

```
10 REM Текст программы для примера на рис.4.5
20 INPUT "Введите значение кода сечения K. K=1,2
   или 3"; K
30 IF K=1 THEN 40 ELSE IF K=2 THEN 60 ELSE
   IF K=3 THEN 80 ELSE PRINT "Неверно введен код
   сечения": GOTO 20
40 INPUT "Введите значения ширины и высоты
   прямоугольника B и H";B,H
50 F=B*H:GOTO 100
60 INPUT "Введите значение диаметра D"; D
70 F=3.14*D^2/4: GOTO 100
80 INPUT "Введите значения внешнего и внутреннего
   диаметров D и D1"; D,D1
90 F=3.14*(D^2-D1^2)/4
100 PRINT "Площадь сечения F равна"; F
110 END
```

Более компактный вариант программы.

Программа 5.5

```
10 REM Еще один вариант программы для примера на
   рис.4.5
20 INPUT "Введите значение кода сечения К. К=1,2
   или 3";K
30 IF K=1 THEN INPUT "B,H"; B,H: F=B*H ELSE IF K=2
   THEN INPUT"D"; D: F=3.14*D^2/4 ELSE IF K=3 THEN
   INPUT "D,D1"; D,D1: F=3.14*(D^2-D1^2)/4 ELSE
   PRINT "Неверно введено значение кода сечения":
   GOTO 20
40 PRINT "Площадь сечения F равна "; F
50 END
```

Общая форма записи оператора ON—GOTO:

ON <арифметическое выражение> GOTO <номер 1>, <номер 2>, ..., <номер *n*>.

Оператор ON—GOTO передает управление в зависимости от значения <арифметического выражения>, предварительно округленного до ближайшего целого. При значении <арифметического выражения>, равном 1, управление передается строке с <номером 1>, при значении, равном 2, — строке с <номером 2> и т. д.

Если <арифметическое выражение> принимает значение, выходящее из диапазона целых чисел от 1 до n, то управление передается оператору, следующему за оператором ON—GOTO.

В списке номеров строк оператора ON—GOTO должен быть указан хотя бы один номер.

Для описания разветвляющейся структуры вполне достаточно одного оператора IF—THEN—ELSE. Поэтому операторы GOTO и ON—GOTO в структурном программировании, о котором говорилось в гл. 4, либо вообще не используются, либо применяется в сочетании с оператором IF—THEN—ELSE.

Примеры использования оператора IF—THEN—ELSE:

1) поиск большего из двух чисел:

```
IF a>b THEN c=a ELSE c=b
```

2) поиск наибольшего d из трех чисел a, b, c

```
IF a>b THEN IF a>c THEN d=a ELSE d=c ELSE IF b>c
THEN d=b ELSE d=c
```

или

```
IF a>b AND a>c THEN d=a ELSE IF b>c THEN d=b
ELSE b=c
```

3) упорядочение чисел a и b по возрастанию

```
IF b>a THEN c=a : a=b : b=c
```

Здесь c — вспомогательная переменная, необходимая для создания резервной копии числа a.

Циклическая структура. При помощи рассмотренных операторов IF—THEN—ELSE и GOTO можно реализовать циклическую алгоритмическую структуру. Иногда это единственный способ записи циклов, например итерационных (программа 5.6).

Программа 5.6

```
10 REM Текст программы для примера на рис.4.9
20 INPUT "Введите значения величин X и E"; X,E
30 S=1:H=1:N=1
40 H=H*X/N:S=S+H:N=N+1
50 IF ABS(H)>E THEN 40
60 PRINT "Сумма ряда S=";S
70 END
80 REM ABS(H) - стандартная функция, определяющая
   абсолютную величину значения аргумента H,
   см. п. .8.
```

Но чаще всего цикл описывается специальными операторами FOR и NEXT:

```
FOR <переменная>=<начальное значение> TO <конечное значение> STEP <значение приращения>
NEXT <переменная>
```

Здесь <переменная> — имя числовой переменной, являющейся управляющей переменной цикла; <начальное значение>, <конечное значение> и <значение приращения> —значения констант, переменных или более сложных арифметических выражений.

Оператор FOR служит началом цикла. Конец цикла — оператор NEXT. Все операторы, заключенные между FOR и NEXT, образуют тело цикла и выполняются до тех пор, пока значение <переменной>, изменяемое от <начального значения> каждый раз на величину <значения приращения>, не достигнет <конечного значения> (программа 5.7). При положительных <значениях приращения> выход из цикла произойдет при первом нарушении условия: значение <переменной> < <конечного значения>, для отрицательных <значений приращения> цикл прекращается при условии значение <переменной>><конечного значения>.

Программа 5.7

```
10 REM Текст программы для примера на рис. 4.6
20 INPUT "Введите начальное XO и конечное XK
   значения переменной цикла и шаг ее изменения T";
   XO,XK,T
30 FOR X=XO TO XK STEP T
40 F=X^2+6*X+10
50 PRINT "При X=";X;" F="; F
60 NEXT
70 END
```

В случае равенства <значения приращения> единице, слово STEP вместе со значением можно опустить. <Значение приращения>, равное нулю, не допускается.

Операторы FOR и NEXT реализуют структуру цикла «до».

Допускается оператор NEXT записывать без имени управляющей переменной.

В случае одновременного закрытия нескольких циклов (в случае вложенных циклов) в операторе NEXT может быть указано несколько имен управляющих переменных. Имена записываются через запятую в порядке, обратном открытию циклов.

Максимальное количество вложенных друг в друга циклов — 255.

Другой пример использования операторов FOR и NEXT (программа 5.8).

Программа 5.8

```
10 REM Текст программы для примера на рис.4.7
20 INPUT "Введите значения размера партии
   деталей N, номинального размера детали A и
   половины поля допуска D";N,A,D
30 FOR I=1 TO N
40 INPUT "Укажите значение фактического размера
   детали"; AF
50 IF AF>A+D OR AF<A-D THEN PRINT "Деталь-брак"
60 REM В случае выхода фактического размера
   детали за пределы поля допуска печатается
   сообщение "Деталь-брак"
70 NEXT
80 END
90 REM Обратите внимание! В строке 70 оператор
   NEXT не содержит имени переменной цикла.
   Это - допустимая конструкция.
```

5.6. ПРОГРАММИРОВАНИЕ АЛГОРИТМОВ ОБРАБОТКИ МАССИВОВ

В качестве примеров использования операторов цикла рассмотрим реализацию алгоритмов обработки массивов (программа 5.9).

Программа 5.9

```
10 REM Текст программы для примера на рис.4.10
   (ввод элементов массива)
20 INPUT "Укажите количество элементов массива N"; N
30 DIM D(N)
40 PRINT "Введите значение очередного элемента"
50 FOR I=1 TO N
60 INPUT D(I)
70 NEXT I
80 ...
```

Поскольку оператор INPUT D(1) составляет тело цикла и будет выполняться N раз, то N раз была бы напечатана на экране дисплея и «подсказка» этого оператора, заполняя экран многократно одной и той же информацией. Для того чтобы этого не произошло «подсказка» вынесена из цикла (строка 40) и поэтому будет напечатана только один раз перед входом в цикл.

Программа 5.10 **Вывод элементов массива**

```
200 ...
210 FOR I=1 TO N
220 PRINT D(I);
230 NEXT I
240 END
```

Обратим внимание на разделитель в конце строки 220 (программа 5.10). Отсутствие разделителя «;» означало бы, что значения элементов массива будут напечатаны в один столбец у левого края экрана. Наличие его в строке 220 определяет вывод элементов массива в одну строку. Однако если количество элементов велико (общая длина выводимых чисел превышает длину строки), то вывод будет осуществляться в ту же строку «послойно», т. е. новые значения будут печататься снова с начала строки, уничтожая уже выведенные данные. Возможна следующая запись этой программы:

```
210 FOR I=1 TO N: PRINT D(I);: NEXT I: END
```

Ниже приведен текст программы 5.11 для примера с определением суммы и произведения элементов массива.

Программа 5.11

```
 70 ...
 80 S=0:P=1
 90 FOR I=1 TO N
100 S=S+A(I):P=P*A(I)
110 NEXT I
120 PRINT "Сумма S=";S,"Произведение P=";P
130 END
```

Программы алгоритмов обработки массивов кажутся очень похожими друг на друга. Сравните, например, предыдущий текст (программа 5.11) и текст программы определения суммы и количества элементов массива (программа 5.12), удовлетворяющих условию на рис. 4.12.

Программа 5.12

```
 70 ...
 80 S=0: K=0
 90 FOR I=1 TO N
100 IF A(I)>0 THEN S=S+A(I): K=K+1
110 NEXT I
120 PRINT "Сумма S=";S,"Количество K=";K
130 END
```

Сходство программ вызвано необходимостью использования операторов FOR и NEXT, организующих обращение к элементам массива для их последующей обработки.

Ниже приведен текст программы 5.13 для примера с формированием массива и элементов исходного массива.

Программа 5.13

```
 70 ...
 80 J=0
 90 FOR I=1 TO N
100 IF A(I)>C AND A(I)<D THEN J=J+1: B(J)=A(I)
110 NEXT I
120 ...
```

В данном тексте в качестве условия отбора элементов массива взята их принадлежность интервалу (C, D). Управляющая переменная цикла (индекс I) обеспечивает просмотр всех элементов исходного массива, а индекс J — запись элементов результирующего массива определяется значением переменной J, получаемым после выхода из цикла.

Определение индекса и значения максимального элемента (см. рис. 4.15) приведено в программе 5.14, а обработка двумерного массива (см. рис. 4.16)—в программе 5.15.

Программа 5.14

```
 70 ...
 80 MA=R(1): K=1
 90 FOR I=2 TO N
100 IF R(I)>MA THEN MA=R(I): K=I
110 NEXT I
120 PRINT "Индекс максимального элемента K=";K,
    "а его значение MA=";MA
130 ...
```

Программа 5.15

```
 10 INPUT "Укажите количество строк M и столбцов N
    двумерного массива";M,N
 20 DIM A(M,N+1)
 30 PRINT "Введите значения элементов массива"
 40 FOR I=1 TO M: FOR J=1 TO N
 50 INPUT A(I,J)
 60 NEXT J,I
 70 FOR I=1 TO M
 80 S=0
 90 FOR J=1 TO N
100 S=S+A(I,J)
110 NEXT J
120 A(I,N+1)=S
130 NEXT I
140 FOR I=1 TO N: FOR J=1 to N+1
150 PRINT A(I,J);
160 NEXT J: PRINT: NEXT I
170 ...
```

В программе 5.15 подробно приведен ввод (строки 10...60) и вывод (строки 140...160) элементов массива. Обратим внимание на то, что описание массива (строка 20) осуществляется сразу с учетом дополнительного (N+1)-го столбца, хотя вводится только N столбцов. Вывод элементов выполняется в виде прямоугольной таблицы (матрицы). Элементы строки массива выводятся на экран в одну строку. При переходе на новую строку действие точки с запятой в строке 150 отменяется пустым оператором PRINT в строке 160, что равносильно команде возврата каретки.

Ниже приведен текст программы для примера (см. рис. 4.18) с циклическим сдвигом элементом массива (программа 5.16) и для примера (см. рис. 4.19) с сор-

тировкой элементов в порядке возрастания (программа 5.17).

Программа 5.16

```
 70 ...
 80 INPUT "Введите число позиций сдвига элементов N";N
 90 FOR J=1 TO M
100 B=A(N)
110 FOR I=N TO 2 STEP-1
120 A(I)=A(I-1)
130 NEXT I
140 A(1)=B
150 NEXT J
160 ...
```

Программа 5.17

```
 70 ...
 80 FOR I=1 TO N-1
 90 P=A(I): K=I
100 FOR J=I+1 TO N
110 IF A(J)<P THEN P=A(J): K=J
120 NEXT J
130 A(K)=A(I): A(I)=P
140 NEXT I
150 ..
```

Обращаем внимание на обязательность (в данном примере) записи в операторе FOR величины шага изменения переменной цикла I, поскольку значение шага не равно +1.

5.7. ФУНКЦИИ И ПОДПРОГРАММЫ

Один из эффективных приемов разработки алгоритмов и программ описан (метод «сверху — вниз») в гл. 4. В его основе лежит понятие процедуры (программы). *Процедура* — это программа, написанная заранее для последующего многократного использования и реализующая ту или иную функцию (действие), направленную на решение какой-либо части поставленной задачи. В языке БЕЙСИК выделяют следующие виды процедур: стандартные функции, функции пользователя и подпрограммы.

Таблица 5.3. **Перечень стандартных функций языка БЕЙСИК**

Функция	Действие
ABS (X)	Дает абсолютное значение X (модуль X); знак X отбрасывается: ABS (3,7) дает 3.7
ATN (X)	Дает значение арктангенса X (arctg X); угол, тангенс которого равен X. Угол вычисляется в радианах, в диапазоне от $-\pi/2$ до $\pi/2$. Для перевода из радиан в градусы пользуйтесь соотношением: градусы$=180/\pi$ радианы, $180/\pi=$ $=57.29577951$
COS (X)	Дает косинус угла X (cos X); X считается заданным в радианах. Для перевода из градусов в радианы пользуйтесь соотношением: радианы$=$ $=\pi/180$ градусы, $\pi/180=1.745329252E-2$
EXP (X)	Дает e ($=2.718281828$) в степени X(e^X); значение X не должно превышать числа 145.0628608586. Не путать функцию с операцией возведения в степень! Запись e ∧X ошибочна!
FIX (X)	Отбрасывает дробную часть X; FIX (3.7) дает 3; FIX (—3.7) дает —3
INT (X)	Дает наибольшее целое число, не превышающее X, т. е. такое целое N, что $N<X<N+1$: INT(3.7) дает 3, INT (—3,7) дает —4
LOG (X)	Дает натуральный логарифм X(ln X); аргумент X должен быть строго положительным. При вычислении логарифма по основанию a $\lg_a X$ пользуйтесь соотношением $\lg_a X=\ln X/\ln a$
RND (X)	Дает число из последовательности случайных чисел, равновероятно распределенных в диапазоне от 0 до 1. При $X<0$ каждое новое обращение к функции обеспечивает переход к новой последовательности чисел; при $X=0$ дает значение предыдущего случайного числа; при $X>0$ дает значение случайного числа из одной и той же последовательности чисел
SGN (X)	Дает знак X согласно следующим правилам: —1, если $X<0$; 0, если $X=0$; +1, если $X>0$.
SIN (X)	Дает синус X (sin X); X считается заданным в радианах, см. также COS (X)
SQR (X)	Дает корень квадратный X ($\sqrt{X}$): значение X должно быть большим нуля
TAN (X)	Дает тангенс X(tgX); X считается заданным в радианах, см. также COS(X) и SIN(X)

Примечание. Здесь X обозначает арифметическое выражение.

Стандартные функции. Они реализованы в виде библиотеки программ, написанных на языке машинных команд и встроенных в транслятор. Каждая функция имеет свое имя и список формальных параметров (аргументов функции), заключенных в круглые скобки. Перечень стандартных функций приведен в табл. 5.3. Имя функции служит для поиска необходимой программы в библиотеке программ и определяет адрес ячейки памяти, куда записывается результат вычисления по программе. Формальные параметры выступают в качестве исходных данных программы. Они указывают на общее количество, тип и порядок следования исходных данных, принятый при реализации стандартной функции. При обращении к функции формальные параметры заменяются конкретными значениями данных, называемых *фактическими параметрами.*

При пользовании функциями из таблицы могут быть полезны следующие тригонометрические тождества:

$$\operatorname{sh}(x) = \frac{e^x - e^{-x}}{2}; \quad \operatorname{sh}(x) = \frac{e^x - e^{-x}}{2}; \quad \operatorname{th}(x) = \frac{\operatorname{sh}(x)}{\operatorname{ch}(x)};$$

$$\arcsin(x) = \operatorname{arctg} \frac{x}{\sqrt{1 - x^2}};$$

$$\arccos(x) = \frac{\pi}{2} - \operatorname{arctg} \frac{x}{\sqrt{1 - x^2}}.$$

Примеры обращения к функциям:

Математическая запись	Запись на языке БЕЙСИК
$4 \cos \frac{y}{8}$	4 * cos (Y /8)
$4 \cos^2 \frac{y}{8}$	4 * cos (Y /8)^2
$A - \sqrt{A + e^{B^2}}$	A — SQR (A + EXP (B^2))
$A - \sqrt[3]{A + \sin^2 B}$	A — (A + SIN (B)^2)^(1 /3)

Функции пользователя. Они отличаются от стандартных функций лишь способом определения. Функции пользователя задаются пользователем с помощью оператора DEFFN:

DEFFN <буква> (<простая переменная>)=<выражение>,

где <буква> — буква латинского алфавита, выполняющая роль имени функции; <простая переменная> — формальный параметр функции; <выражение> — арифметическое или строковое выражение.

Функция может быть определена для работы со строковыми или числовыми данными. Тип функции указывается знаками «%», «#», «!» после имени.

Имя переменной, указанной в качестве формального параметра, является локальным, т. е. может быть использовано для другой переменной вне описания функции. В выражение могут входить и другие переменные. Значения этих переменных будут уже глобальными, т. е. доступными под тем же именем в других местах программы.

Действие оператора DEFFN осуществляется при его упоминании при помощи функции:

FN <буква> (<имя простой переменной>),

где <буква> — имя функции.

Оператор DEFFN, являясь описательным оператором, может быть размещен в любом месте программы до использования функции.

Для примера опишем тригонометрические функции, не являющиеся стандартными:

Описание функции	Обращение к функции
DEF FNT (X) = cos (X)/SIN (X)	FNT (X)
DEF FNS (X) = 1/COS (X)	FNS (X)
DEF FNC (X) = 1/SIN (X)	FNC (X)
DEF FNH (X) = (EXP (X) — EXP × ×(—X))/2	FNH (X)
DEF FNQ (X) = (EXP (X) + EXP× ×(—X)/2	FNQ (X)
DEF FNA (X) = 2 * ATN ((1 — —SQR) 1 — X^2))/X	FNA (X)
DEF FNL (X) = LOG (X /LOG (10)	FNL (X)

Подпрограмма. Она представляет собой набор любых операторов для реализации логически законченной части задачи.

Операторы подпрограммы располагаются в строках, которые нумеруются обычным порядком. Как

правило, все подпрограммы группируются вместе после оператора END основной программы.

Обращение к подпрограмме осуществляется с помощью оператора GOSUB, в котором указывается номер первой строки подпрограммы

GOSUB <номер строки>

Оператор GOSUB передает управление подпрограмме, первый оператор которой стоит в строке <номер строки>. Затем выполняются операторы подпрограммы по порядку до тех пор, пока не встретится оператор RETURN, которым должен завершаться текст подпрограммы. После этого управление будет передано в основную программу оператору, следующему непосредственно за оператором GOSUB (рис. 5.1).

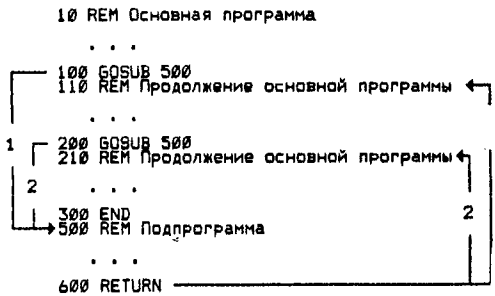


Рис. 5.1. Схема выполнения программы с подпрограммой

Как видим, в отличие от функций подпрограммы в языке БЕЙСИК не имеют списка формальных параметров. Поэтому здесь нет и локальных переменных. Все переменные подпрограмм в одинаковой мере доступны и программам, и всем подпрограммам. Изменение значений переменных в подпрограмме приводит к глобальному (повсеместному) изменению значений переменных с тем же именем во всех частях программы.

Ниже приведен текст программы 5.18 для примера (см. рис. 4.20) с формированием одномерного масси-

ва из минимальных элементов столбцов двумерного массива.

Программа 5.18

```
60
70 FOR J=1 TO N
80 FOR I=1 TO M: C(I)=B(I,J): NEXT I
90 GOSUB 250: A(J)=MI
100 NEXT J
110 ...


200 END
250 MI=C(1)
260 FOR I=2 TO M
270 IF C(I)<MI THEN MI=C(I)
280 NEXT I
290 RETURN
```

Подпрограмма может быть расположена в любом месте программы, однако удобнее всего размещать ее после оператора END основной программы, тогда не потребуется лишних операторов GOTO.

Составим программу 5.19 для (см. рис. 4.23) определения количества студентов, оценка которых выше среднего балла:

Программа 5.19

```
10 GOSUB 100: GOSUB 200: GOSUB 300
20 PRINT "Количество студентов, отметки которых выше
   среднего балла группы, K="; K
30 END
100 INPUT "Укажите  количество студентов в группе N"; N
110 PRINT "Введите оценки студентов"
120 FOR I=1 TO N: INPUT O(I): NEXT I
130 RETURN
200 SS=0
210 FOR I=1 TO N: SS=SS+O(I): NEXT I
220 SS=SS/N
230 RETURN
300 K=0
310 FOR I=1 TO N
320 IF O(I)>SS THEN K=K+1
330 NEXT I
340 RETURN
```

Здесь отдельные крупные задачи оформлены (в отличие от блок-схемы) в виде подпрограмм. Текст основной программы состоит преимущественно из обращений к подпрограммам и соответствует первому уровню детализации при программировании «сверху—вниз». На следующем этапе детализации разрабатываются подпрограммы к основной программе.

Для сложных программ могут требоваться и подпрограммы, детализирующие уже разработанные подпрограммы. Получается целая иерархическая система вложенных друг в друга и в конечном итоге в основную программу подпрограмм. Такая система обеспечивает возможность разрабатывать и отлаживать программу поэтапно.

5.8. ОБРАБОТКА СИМВОЛЬНОЙ ИНФОРМАЦИИ

Обработка символьной информации на языке БЕЙСИК связана преимущественно с использованием функций (табл. 5.4).

Таблица 5.4. **Перечень функций БЕЙСИКа для работы со строковыми данными**

Название операции	Функция	Действие
Значение кода	ASC (A$))	Дает код ASCII первого символа A$. Обратное действие выполняет функция CHR$
Преобразование кода	CHR$ (N)	Дает символ с кодом ASCII, равным N. Обратное действие выполняет функция ASC. Удобна при работе с непечатаемыми символами (например, с клавишами перемещения курсора).
Длина строки	LEN (A$)	Дает число символов в A$; LEN («строка») дает 6
Вырезка строки	LEFT$ (A$, N) RIGHT$ (A$, N)	Дает N левых (LEFT$) или правых (RIGHT $) символов A$; LEFT$ («строка», 3) дает «стр»; RIGHT$ («строка», 3) дает «ока»

Продолжение табл. 5.4

Название операции	Функция	Действие
	MID$ (A$, N, M)	Дает M символов A$, начиная с символа, занимающего N-ю позицию A$: MID$ («строка», 3,3) дает «рок»
Позиция	INSTR (A$, B$)	Дает начальную позицию первого вхождения B$ в A$; дает 0, если полного вхождения B$ в A$ нет; если указан параметр N, то поиск вхождения B$ в A$ начинается с N-й позиции; INSTR («колокол», «кок») дает 1; INSTR (3, «колокол», «кол») дает 5; INSTR («колокол», «кол») дает 0
Символьная запись числа	STR (A)	Дает строку, являющуюся символьной записью числовой константы A. Обратное действие выполняет функция VAL
Число	VAL (A$)	Дает число по его символьной записи в A$. Символьная запись должна содержать только цифры и знаки +, —, и «.». Пробелы в A $ при этом игнорируются. Обратное действие выполняет функция STR$. Полезна при обработке строк, содержащих символьную запись числа. Преобразованные при помощи функции VAL числовые строки можно обрабатывать арифметически
Особый ввод	INKEY$	Не имеет аргументов и обеспечивает ввод с клавиатуры одного символа
	INPUT$ (N)	Обеспечивает ввод с клавиатуры N символов (см. также п. 7.6)
Средства форматирования вывода	SPACE$ (N)	Дает строку из N пробелов
	STRING$ (NM),	Дает строку из N одинаковых символов с кодом ASCII, равным M

Окончание табл. 5.4

Название операции	Функция	Действие
Разное	STRING$ (N, A$)	Дает строку из N-кратно повторенного первого символа A $
	HEX$ (N) OST$ (N)	Дает строку, содержащую шестнадцатеричную (HEX$) или восьмеричную (OCT $) запись числа N

Примечание. Здесь A$, B$ — строковые данные (константа, переменная, элемент массива); N, M — числовые данные.

Рассмотрим несколько типовых примеров обработки символьной информации. Для краткости алгоритмы опишем в вербальной форме, но программы снабдим комментарием.

Процесс повторяется до тех пор, пока не встретится точка, знак вопроса или восклицания (программа 5.20).

Пример 5.1. Определить количество слов в предложении.

Алгоритм решения задачи сводится к сравнению каждого символа предложения с символами — разделителями слов («пробел», «,», «:», «—», «(», «)») и вычислению количества разделителей, которое на единицу меньше количества слов.

Программа 5.20

```
10 INPUT "Ведите текст предложения";  A$
20 K=0 : I=1' K - искомый результат - количество
   слов, I - номер проверяемого символа предложения
30 C$=" ,:-( )": D$=".?!":'   D$- шаблон разделителей конца предложения; C$ - шаблон разделителей слов
40 N=LEN(A$)' N - число символов в предложении
50 FOR I=1 TO N
60 IF INSTR (C$, MID$(A$,I,1))<> 0 THEN K=K+1'
   функция INSTR определяет вхождение вырезанного
   символа в шаблон разделителей слов
70 IF INSTR (D$,MID$(A$,I,1))<> 0 THEN 90
80 NEXT
90 PRINT "в предложении"  k-1 "слов"
100 END
```

Пример 5.2. Пусть имеется платежная ведомость, содержащая N строк. В каждой строке ведомости указана фамилия со-

трудника и его заработная плата. Составить список сотрудников, зарплата которых не превышает заданной величины А.

Алгоритм решения задачи предусматривает возможность просмотра каждой строки ведомости, преобразования величины зарплаты из символьной формы представления в числовую, сравнения значения зарплаты с указанным пороговым значением А и печать требуемых строк ведомости (программа 5.21).

Программа 5.21

```
10 REM Строки 20-40-ввод строк платежной ведомости
20 INPUT "Укажите количество строк в ведомости";N
30 DIM A$(N): PRINT "Укажите фамилию и инициалы
   сотрудника и его зарплату. Помните, под запись
   значения зарплаты отводится 6 правых позиций
   строки!"
40 FOR I=1 TO N: INPUT A$(I):NEXT I
50 INPUT "Укажите значение порога A"; A
60 PRINT: PRINT: PRINT "Список сотрудников,
   зарплата которых менее"; A; " руб.":PRINT
70 REM В строке 60 программы обеспечивается
   пропуск 2 строк на экране,печать заголовка
   выходного списка и пропуск еще одной
   строки"
80 FOR I=1 TO N
90 IF VAL (RIGHT$(A$(I),6))<A THEN PRINT A$'
   функция VAL преобразует вырезанные из A$(I)
   шесть правых символов в число
100 NEXT I
110 END
```

Пример 5.3. Результаты вступительных экзаменов представлены в виде списка из N строк, в каждой строке которого записаны фамилия студента и отметки по каждому из *M* экзаменов. Определить количество абитуриентов, сдавших вступительные экзамены только на «отлично».

Алгоритм решения задачи заключается в следующем. Организуется доступ ко всем строкам списка (с помощью цикла). В каждой строке поочередно «вырезается» символ, относящийся к результату того или иного экзамена и преобразуется в числовую форму. Сумма чисел по всем экзаменам для одного студента дает сумму баллов этого студента. Остается только сравнить эту сумму с максимальной суммой баллов по всем экзаменам и, если требуется, учесть этого студента в общем количестве отличников (программа 5.22).

Программа 5.22

```
...
100 REM Считаем, что массив A$(N), т.е. список
    из N строк, уже имеется в памяти компьютера
110 REM Каждая запись A$(I) имеет длину L симво-
    лов, из них последние M символов (позиций)
    заняты результатами экзаменов.
120 K=0' K - количество абитуриентов-отличников
130 FOR I=1 TO N:S=0'S-сумма баллов абитуриента
140  FOR Y=1TO M
150 S=S+VAL(MID$(A$(I),L-M+Y,1))
160 REM Функция MID "вырезает"  из  переменной
    A$(I)  один, (L-M+Y)- й символ.
    Функция VAL преобразует этот символ в число-
    вую  форму.
170 NEXT Y
180 IF S = 5*M THEN K=K+1
190 NEXT I
200 PRINT "Количество абитуриентов, сдавших
    сессию  на  отлично, равно";K
210 END
```

5.9. ВЗАИМОДЕЙСТВИЕ ПОЛЬЗОВАТЕЛЯ С КОМПЬЮТЕРОМ

Включение компьютера. Взаимодействие пользователя с компьютером начинается с его включения. Для этого возможно придется выполнить ряд действий, которые зависят от типа компьютера. Рассмотрим порядок включения компьютера с одним терминалом (персональный компьютер), когда интерпретатор с языка БЕЙСИК находится в постоянном запоминающем устройстве (ПЗУ) компьютера*. ПЗУ — обычная область памяти, которую можно только читать. При изготовлении компьютера в нее заносятся программы. В этом случае подготовка компьютера состоит из одного действия: включения электропитания.

При благополучном включении на дисплее должен появиться специальный маркер (курсор, вид которого зависит от компьютера). Курсор на экране всегда

---

* Порядок загрузки дискового варианта языка БЕЙСИК описан в гл. 9. Там же можно прочитать об особенностях работы компьютера, входящего в локальную сеть.

указывает знакоместо, на которое будет выведен символ, набранный на клавиатуре.

Набор программы. Набор программы представляет собой обычное печатание текстов. При печатании текста программы необходимо помнить лишь, что понятие строки программы отличается от обычного понятия строки текста наличием номера у строки и длиной.

Длина строки программы не более 255 символов. Длина же строки на экране обычно не больше 40 символов, т. е. строка программы может занимать несколько экранных строк. Поэтому набор длинных программных строк вызывает у пользователей некоторые затруднения. На самом деле эти затруднения надуманные. Они исчезнут сами собой, если на проблему переноса текста строки программы с одной экранной строки на другую вообще не обращать никакого внимания, а доверить компьютеру самому решить эту задачу.

Набор строки программы всегда должен завершаться нажатием на клавишу перевода строки (возврата каретки). Только эта клавиша заставляет компьютер прочитать текст программы с экрана и занести ее в память. Как правило, при занесении строки программы в память компьютер модифицирует ее (заменяет строчные латинские буквы на прописные, в именах переменных удаляет русские буквы, в числах убирает незначащие нули, определяет тип числовых констант и т. д.). Таким образом, запомненная строка может отличаться от строки, которую пользователь видит на экране. Поэтому необходимо периодически проверять содержание находящихся в памяти строк. Для контроля строк программы используется команда LIST (LIST N, LIST—N, LIST N—, LIST N—M), вызывающая на экран текст всей программы (или соответственно N-й строки, с первой по N-ю, с N-й до последней, с N-й по M-ю строки).

Выполнение программы. Выполнение программы с оператора, имеющего наименьший номер, осуществляется по команде RUN. Возможен также запуск программы с любой строки программы. Для этого достаточно набрать команду RUN <номер строки>, с которой должно начаться выполнение программы, и нажать на клавишу возврата каретки.

Если ошибок в программе нет, то она будет выполняться до конца. В противном случае интерпретатор прервет выполнение программы и сообщит об ошибке. Выполнение программы может быть прервано или приостановлено пользователем: одновременное нажатие клавиш CTRL и STOP приводит к прекращению выполнения программы и переходу компьютера в режим немедленной обработки; нажатие только клавиши STOP приостанавливает выполнение программы. Для возобновления выполнения программы необходимо повторное нажатие этой же клавиши STOP.

Редактирование программы. Ошибки при наборе программы неизбежны. Процесс исправления ошибок в программе называют *редактированием.* Существует несколько способов редактирования строк программы. Наиболее распространено редактирование путем перенабора или удаления строк программы (более подробно редактирование программ описано в гл. 10).

Неверно набранные строки программы или группы строк могут быть удалены из программы:

командой NEW (стирается вся программа);

набором номера строки программы и нажатием клавиши возврата каретки (удаляется строка программы с этим номером);

командой DELETE N—M с последующим нажатием клавиши возврата каретки удаляются из программы все строки, номера которых находятся в интервале от N до M.

УПРАЖНЕНИЯ

5.1. Напишите программы к примерам гл. 4.

5.2. Напишите программы к упражнениям гл. 4.

5.3. Разработайте и напишите программу, анализирующую длину слов в предложении. Программа должна воспринимать предложения разного вида, в которых допускаются все обычные знаки препинания. Она должна также определять максимальное, минимальное и среднее количества букв в словах предложения.

5.4. Разработайте и напишите программу анализа текста. Она должна воспринимать фрагмент текста как одно предложение и вычислять частоту появления входящих в него слов. При обработке текста программа должна заполнять таблицу, содержащую слова и значение частоты появления этих слов в тексте.

Отсортируйте строки таблицы: а) в порядке алфавита; б) в порядке убывания значений частоты появления слов в тексте.

Глава 6

ЭЛЕМЕНТЫ МАШИННОЙ ГРАФИКИ

В этой главе рассматриваются принципы ввода, хранения и обработки графической информации в компьютере, рассказывается о различных способах организации памяти компьютера, предназначенной для хранения графической информации, приводятся операторы языка БЕЙСИК, позволяющие строить на экране дисплея компьютера простейшие геометрические примитивы, статические и динамические изображения. Дается большое количество примеров, иллюстрирующих действие описанных в главе операторов языка БЕЙСИК.

6.1. ПРЕДСТАВЛЕНИЕ ГРАФИЧЕСКОЙ ИНФОРМАЦИИ В КОМПЬЮТЕРЕ

В предыдущих главах экран дисплея компьютера рассматривался лишь для отображения информации, представленной в алфавитно-цифровом виде: вывода на экран чисел, текстов, таблиц и даже графиков функций, построенных при помощи обычных текстовых символов. Возможность построения таких изображений (обычно их называют псевдографическими) предоставляется посредством оператора PRINT. Графическое качество изображения графика в основном зависит от наличия среди символов специальных графических символов (отрезков, уголков, квадратов и т. д.).

Существует другой способ построения графических изображений, в основе которого лежит новый тип «строительных» элементов, более мелких, чем символы. Эти элементы называются *точками*. Допустимое число точек на экране во много раз превышает допустимое число символов. Если, например, на экране дисплея компьютера «ЯМАХА» допустимое число символов равно $24 \times 40 = 960$, то допустимое число точек будет $192 * 256 = 49\,152$, т. е. в 51 раз больше.

Точки можно расположить на экране так близко друг к другу, что на некотором расстоянии от экрана изображаемая с их помощью линия кажется сплошной и более или менее гладкой. При этом чем больше

допустимое число точек на экране, т. е. чем выше разрешающая способность экрана, тем меньше искажается линия и тем она плавней. Точное построение изображений называется *графическим построением изображений.*

Принцип хранения изображения в компьютере. Отметим, что вся информация, которая отображается на экран дисплея, хранится в специальной области памяти, называемой *видеопамятью.* Специфика этой памяти состоит в том, что компьютер автоматически с определенной частотой регенерации экрана отображает все содержимое видеопамяти на экран дисплея. Отображение видеопамяти происходит быстро и повторяется достаточно часто, тем самым создается впечатление устойчивой картинки.

Существует несколько способов запоминания цветного графического изображения в видеопамяти. Рассмотрим два из них.

1. Каждой точке экрана дисплея однозначно соответствует однобайтная ячейка памяти. В этом случае для запоминания изображения, нарисованного на экране дисплея, имеющего, например, 49 152 точки, потребуется не менее 48 Кбайт памяти. Вспомним, что в одном байте памяти можно хранить лишь целое число из интервала от 0 до 255. Используем это число для кодировки цвета точки. Тогда каждую точку экрана можно «закрасить» (занося в соответствующий байт памяти код цвета закраски) в один из 256 цветов.

2. Уменьшение объема видеопамяти без изменения количества точек на экране. Все точки экрана сгруппируем в горизонтальные отрезки, длина которых равна восьми точкам, и установим каждому такому отрезку (их в 8 раз меньше, чем точек) в однозначное соответствие однобайтную ячейку памяти. Пока точки в пределах отрезка не различимы. Воспользуемся содержимым байта памяти. За каждым битом однобайтной ячейки памяти закрепим одну из точек отрезка. Теперь число, хранимое в одном байте видеопамяти, будет определять уже не цвет точки, а местоположение точек в горизонтальном отрезке. Например, десятичное число 17 (двоичная запись 99919991), занесенное в байт памяти, укажет, что в отрезке две точки — четвертая и восьмая.

Цвет точек пока не определен. Поэтому за каждым из отрезков закрепим еще один байт памяти. Поделим каждый такой байт памяти на два полубайта. Пусть первый полубайт хранит информацию о цвете точек на отрезке, второй — о цвете фона отрезка. Теперь палитра цветов состоит лишь из 16 различных цветов. Одновременно на отрезке из 8 точек может присутствовать только два цвета. Таким образом, сократив объем памяти (в данном случае в 4 раза), мы вынужденно ограничили себя в палитре цветов.

Описанные способы запоминания графического изображения используются, например, в компьютере «ЯМАХА».

Кодирование графических изображений. Итак, чтобы получить на экране дисплея какое-либо графическое изображение, необходимо в видеопамять занести соответствующие данные. Автоматически эти данные отобразятся компьютером на экран в виде графического изображения. Ввести данные в видеопамять можно «вручную», непосредственно указывая адрес памяти и данные, которые мы хотим занести в ячейку памяти. Эта работа очень кропотливая и сложная. Поэтому для того, чтобы упростить процесс кодирования изображения, в язык БЕЙСИК введены специальные *графические операторы.* Эти операторы позволяют строить изображение из готовых конструкций, называемых *графическими примитивами.* К ним относятся точка, отрезок, прямоугольник, окружность, эллипс, дуга. Процесс кодирования изображения еще более упрощается при использовании программ, называемых *графическими редакторами.* О работе с ними будет рассказано в гл. 9.

6.2. ОСОБЕННОСТИ ИЗОБРАЖЕНИЯ ГРАФИЧЕСКИХ ОБЪЕКТОВ НА ДИСПЛЕЕ

Рассмотрим процесс построения графических объектов на экране дисплея. Как уже отмечалось, на экране может быть высвечено лишь конечное число точек. Множество точек образует матрицу, состоящую из фиксированного числа строк (192) и столбцов (256). Для того чтобы выделить точку на экране, необходимо задать номера строки и столбцов матрицы. Обозначим номер строки через Y, номер столбца че-

рез X. Тогда пара чисел (X, Y) однозначно задает местоположение точки на экране. Будем далее трактовать эту пару чисел как координаты точки плоскости, в которой введена система координат X и Y. Эта плоскость и введенная на ней система координат имеют ряд особенностей. На координатных осях можно использовать лишь конечное (ограниченное сверху и снизу) число значений. Все значения должны быть обязательно целыми и положительными. Назовем ука-

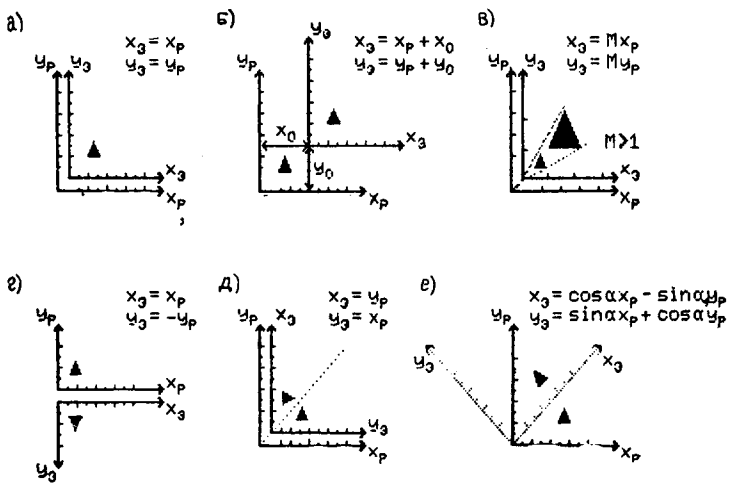


Рис. 6.1. Преобразования координат:

а — тождественное; *б* — перенос; *в* — масштабирование; *г* — поворот на 180° (симметричное отображение относительно координатной оси); *д* — поворот на 90° (смена осей); *е* — поворот на произвольный угол

занную плоскость *экранной,* а плоскость с обычной системой координат (на осях координат которой можно выбирать бесконечное число значений как целых, так и дробных) — *реальной.*

Графические объекты, построенные в реальной плоскости, перед отображением их на экран необходимо предварительно преобразовать. Существует несколько типов преобразований. Рассмотрим лишь три из них: перенос, поворот и масштабирование.

Пусть $X_р$, $Y_р$ — координаты реальной плоскости, а $X_э$, $Y_э$ — экранной. Суть преобразования состоит в пе-

ресчете значений координат объекта, заданных в системе координат $X_р$, $Y_р$, в значения координат системы $X_э$, $Y_э$. Для рассматриваемых преобразований формулы для пересчета приведены на рис. 6.1.

Аналогично можно построить формулы для пересчета координат более сложных преобразований в виде одного преобразования или в виде последовательного действия более простых преобразований.

Например, преобразование «движение» можно совершить, используя формулы

$$X_э = \cos\alpha x_р - \sin\alpha y_р + x_0;$$
$$Y_э = \sin\alpha x_р + \cos\alpha y_р + y_0$$

или последовательно преобразуя исходную систему путем поворота и переноса.

Преобразования координат можно использовать не только для перевода графического объекта из одной плоскости в другую, но и в рамках одной плоскости. При этом цели могут быть разные, а именно: расположение типовых и повторяющихся частей объекта в разных местах плоскости; получение симметричных частей объекта; деформация (масштабирование) фигур и т. д.

6.3. ГРАФИЧЕСКИЕ ОПЕРАТОРЫ ЯЗЫКА БЕЙСИК

Перейдем непосредственно к кодированию изображения графического объекта, удовлетворяющего требованиям экранной плоскости. Сразу откажемся от ручного способа кодирования изображения из-за его громоздкости с помощью операторов, работающих с видеопамятью. (Для справки — в языке БЕЙСИК MSX — это оператор VPOKE и функция VPEEK.) Рассмотрим только операторы языка БЕЙСИК, позволяющие автоматизировать процесс кодирования изображения (графические операторы).

Построение графических объектов посредством графических операторов. Оно во многом схоже с обычным рисованием линий на бумаге: сначала отмечается (определяется) положение точек, а затем они соединяются необходимой линией. Поэтому способ кодирования изображения с помощью графических опера-

торов не только менее трудоемкий, но и более понятный. Однако выигрывая в понятности и простоте, мы проигрываем в быстродействии. Графический оператор исполняется компьютером в несколько этапов. Кроме присущего всем операторам синтаксического разбора компьютер совершает следующие операции: вычисление положения всех точек, образующих линию; определение соответствующих адресов ячеек видеопамяти; ввод необходимых данных в найденные ячейки памяти. Суммарное время, затраченное компьютером на выполнение этих операций, превышает время отображения содержимого видеопамяти на экран дисплея. Возникает иллюзия рисования линий. Компьютер на экране как бы вычерчивает каждую линию, входящую в графический объект. Поэтому построение изображений посредством графических операций получило название медленной графики.

Состав графических операторов языка БЕЙСИК существенно зависит от модели компьютера. На разных компьютерах для построения одних и тех же графических объектов используются разные наборы графических операторов. Рассмотрим набор графических операторов языка БЕЙСИК стандарта MSX.

Управление режимом работы экрана. MSX БЕЙСИК представляет пользователю 6 различных режимов работы экрана дисплея. Режимы экрана характеризуются типом выводимой на него информации, многообразием цветов, объемом видеопамяти, допустимым количеством точек на экране.

Режим работы экрана задается оператором SCREEN, имеющим следующую форму записи:

SCREEN <режим>

Рассмотрим только четыре значения параметра <режим>:

0 — задает текстовый экран размером 24* 40 (24 строки по 40 позиций в строке);

1 — текстовый экран размером 24* 32;

2 — графический экран высокой разрешающей способности размером 256* 192 точек;

3 — графический экран низкой разрешающей способности размером 64* 48 точек.

Текстовый экран предназначен только для вывода текстовых и специальных символов. Символы выводятся на экран с помощью оператора PRINT. При

этом под каждый символ отводится одна позиция (прямоугольник размером 8 * 6 точек в SCREENO или 8 * 8 точек в SCREEN1).

Графический экран представляет собой *рабочее поле* (зона построений) и *кромку* (бордюр). Допускается на рабочее поле выводить как графические, так и текстовые изображения. Кромка для построений не используется. Рабочее поле (экранная плоскость) имеет левую декартовую систему координат с началом координат в верхнем левом углу (рис. 6.2).

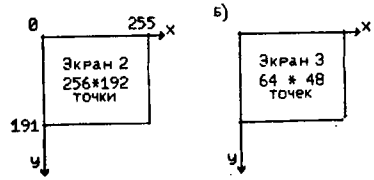


Рис. 6.2. Координатные оси графического экрана компьютера «ЯМАХА»

Конечно, тип экрана полностью определяется типом выводимой на него информации. И тем не менее необходимо также учитывать, что операторы ввода-вывода INPUT и PRINT работают только с текстовым экраном, а графические операторы — только на втором и третьем экранах.

Выбор цветов на графическом экране (табл. 6.1) Задание цвета линий (изображения), рабочего поля и

Таблица 6.1. **Определение кода цвета**

Код	Цвет	Код	Цвет
0	Прозрачный (невидимый)	8	Красный
		9	Светло-красный
1	Черный	10	Темно-желтый
2	Зеленый	11	Светло-желтый
3	Светло-зеленый	12	Темно-зеленый
4	Темно-синий	13	Сиреневый
5	Светло-синий	14	Серый
6	Темно-красный	15	Белый
7	Небесно-синий		

кромки осуществляет оператор COLOR. Общая форма записи COLOR:

COLOR <код цвета изображения>, <код цвета рабочего поля>, <код цвета кромки>

Таким образом, палитра цветов составляет 16 цветов. Как отмечалось ранее (см. § 6.1), ограничение на цвета вызвано объемом видеопамяти. Если объем видеопамяти невелик, то и палитра цветов скудная. В режимах SCREEN2 и SCREEN3 для запоминания изображения отводится 12 Кбайт видеопамяти. Указанного объема памяти хватает для запоминания изображения, составленного из 256 * 192 точек, только в том случае, если палитра цветов состоит всего из 16 цветов, а в каждом горизонтальном отрезке из восьми клеток используется только два цвета. Поэтому при работе с различными цветами необходимо следить не только за тем, чтобы код цвета принимал значения лишь от 0 до 15, но и за количеством цветов на горизонтальных отрезках. Появление в таком отрезке третьего цвета приводит к искажению изображения: весь отрезок закрашивается третьим цветом (компьютер таким своеобразным образом указывает на ошибку).

Особенности исполнения оператора на разных экранах. Оператор COLOR допустимо использовать в текстовом и графическом режимах работы. В текстовом режиме экрана оператор COLOR изменяет цвета рабочего поля и всех символов, выведенных к этому моменту на экран. Текстовый экран не имеет кромки (бордюра). Поэтому в данном случае значение третьего параметра оператора COLOR всегда игнорируется. Оператор COLOR приобретает вид:

COLOR <код цвета символов>, <код цвета экрана>

В графическом режиме оператор COLOR имеет еще более короткий вид: COLOR <код цвета изображения>. Он меняет цвет линий и символов, которые будут выводиться на экран после его исполнения.

В полном виде оператор COLOR используется только перед выходом в графический режим, т. е. непосредственно перед оператором SCREEQ2 или SCREEN3.

При включении компьютера автоматически срабатывают операторы SCREEN0 и COLOR15,4.

Точка. Построение точки на экране дисплея производит оператор

PSET <координаты точки>, <код цвета точки>

Параметр <координаты точки> служит для задания положения точки на экране, а параметр <код цвета точки> — ее цвета. Координаты точки могут быть заданы в абсолютной или относительной форме. Абсолютная форма записи координат имеет вид (x, y), где x и y — арифметические выражения (т. е. числа, переменные, стандартные функции и т. д.). Так как экран имеет конечные размеры по осям X и Y, то значения x и y должны находиться в интервалах $0<x<256$; $0<y<192$. В противном случае точка либо будет размещена за пределами экрана [если числовые значения x и y не на много больше (меньше) чисел 255 (0), 191 (0)], либо появится сообщение об ошибке.

Пример 6.1. Построить точку на экране дисплея (программа 6.1).

Программа 6.1

```
10 COLOR 6,7,4 'задает цвета: рабочего поля -
    небесно - синий, изображения - темно-красный,
              бордюра - темно-синий.
20 SCREEN 2 'выход в графический экран
30 PSET (100,50),6 'рисует темно-красную точку  с
                   координатами по X=100, по Y=50
40 PSET (80,0),9 'светло-красная точка с координа-
                 тами (80,0)
50 GOTO 50 'зацикливает программу
```

Использование оператора 50 (вместо обычно оператора END) вызвано необходимостью приостановить нормальное завершение исполнения программы. При нормальном завершении программы (при выполнении оператора END) компьютер инициализирует графический экран и автоматически переходит в режим работы с текстовым экраном.

Пример 6.2. Положение точки в относительной форме записывается STEP(x, y). Здесь числовые значения арифметических выражений x и y задают приращения координат точки относительно предыдущей точки (точки 0, 0), если построения к этому моменту отсутствовали.

Программа 6.2

```
10 COLOR 6,7,4
20 SCREEN 2
30 PSET (100,50),6
40 PSET STEP (-20,-50),9 'точка с координатами (80,0)
50 GOTO 50
```

Параметр <код цвета точки> — необязательный параметр, т. е. допустим следующий вид записи оператора PSET: PSET <координаты точки>. В этом случае цвет точки будет совпадать с цветом изображения, заданного оператором COLOR В программах 6.1 и 6.2 действие программы не изменится, если вместо оператора под номером 30 записать оператор PSET (100, 50).

При построении точки оператором PSET (аналогично и во всех графических операторах) используются лишь целые части числовых значений выражений x и y, т. е. INT(x) и INT(y). Это значит, что любые два дробных числа, имеющие одинаковые целые части, графическими операторами не различаются.

Располагая оператором PSET, можно строить уже любые кривые. Для этого, как правило, достаточно определить математическую формулу, описывающую кривую и, многократно обращаясь к этой формуле, найти положение всех точек на данной кривой.

Пример 6.3. Программа 6.3. **Построение окружности радиуса 50 с центром в точке (120, 90).**

Программа 6.3

```
10 COLOR 1,15,15
20 SCREEN
30 FOR FI=0 TO 6.28 STEP 6.28/100
40 PSET (120+50*COS(FI),90+50*SIN(FI))
50 NEXT
60 GOTO 60
```

В программе 6.3 используется уравнение окружности, записанное в полярной системе координат (R, φ):

$$x = x_0 + R\cos\varphi,$$

$$y = y_0 + R\sin\varphi.$$

Точка (x_0, y_0) задает центр окружности; R — радиус. При этом полярный угол изменяется в пределах от 0 до 2π.

Отрезок прямой, прямоугольник. Изображение отрезков прямых и прямоугольников выполняется оператором LINE.

Общая форма записи оператора: LINE<координаты точки 1>—<координаты точки 2>, <код цвета>, <признак прямоугольника>. Допустимы также следующие *формы записи оператора* LINE:

1. LINE <координаты точки 1>—<координаты точки 2>, <код цвета>
2. LINE<координаты точки 1>—<координаты точки 2>
3. LINE—<координаты точки 2>
4. LINE<координаты точки 1>—<координаты точки 2>, <признак прямоугольника>
5. LINE—<координаты точки 2>„<признак прямоугольника>.

Первые три формы записи оператора LINE используются для построения отрезков. В этом случае параметры <координаты точки 1> (расшифровку параметра см. выше) и <координаты точки 2> определяют положение точки начала и конца отрезка. Параметр <код цвета> задает цвет отрезка. Если он опущен (формы оператора 2 и 3), то отрезок строится цветом изображения оператора COLOR. В случае умолчания параметра <координаты точки 1> (форма 3) за точку начала отрезка принимается точка окончания предыдущих построений.

Пример 6.4. Построить квадрат с разноцветными сторонами

Программа 6.4

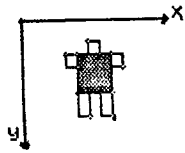
```
10 COLOR 1,15,15
20 SCREEN 2
30 LINE (10,10)-(10,60),8 'красный отрезок,
                    параллельный оси Y
40 LINE -(60,60),2 'зеленый отрезок,  параллельный
                     оси X
50 LINE -(60,10)   'черный отрезок
60 LINE -(10,10),14'серый отрезок
70 GOTO 70
```

Оператор LINE (программа 6.4), записанный в формах 4 и 5, а также в общем виде, выполняет построение прямоугольников, стороны которых параллельны осям координат. Параметр <признак прямоугольника> может принимать только два символьных значения: B — признак контурного прямоугольника и BF — признак закрашенного прямоугольника. При этом параметры <координата точки 1> и <координата точки 2> рассмат-

риваются как начало и конец отрезка диагонали прямоугольника, а параметр <код цвета> — как цвет прямоугольника. Параметр <код цвета> может отсутствовать (формы 4 и 5), тогда его место указывается запятой (первая запятая в четвертой и пятой записях оператора LINE заменяет пропущенный параметр, вторая запятая отделяет следующий параметр от пропущенного).

Пример 6.5. Построить робот. Ниже приведена программа 6.5 и рис. 6.3 построения робота.

Рис. 6.3. Робот



Программа 6.5

```
10 COLOR 1,15,15
20 SCREEN 2
30 LINE (40,10)-(45,15),,B  'голова
40 LINE (35,15)-(50,35),,BF 'туловище
50 LINE (25,15)-(35,20),,B  'правая рука
60 LINE (50,15)-(60,20),,B  'левая рука
70 LINE (35,35)-(40,45),,B  'правая нога
80 LINE (45,35)-(50,45),,B  'левая нога
90 GOTO 90
```

Окружность, эллипс, дуга. Построить окружность, эллипс, дугу можно при помощи оператора CIRCLE. Общая форма записи оператора:

CIRCLE <координаты центра>, <радиус>, <код цвета>, <начальный угол дуги>, <конечный угол дуги>, <отношение осей эллипса>.

Частные формы записи:

1. CIRCLE <координаты центра>, <радиус>, <код цвета>
2. CIRCLE <координаты центра>, <радиус>
3. CIRCLE <координаты центра>, <радиус>, <код цвета>, <начальный угол дуги>, <конечный угол дуги>
4. CIRCLE <координаты центра>, <радиус>, <начальный угол дуги>, <конечный угол дуги>
5. CIRCLE <координаты центра>, <радиус>, <код цвета> ,,, <отношение осей эллипса>

6. CIRCLE <координаты центра>, <радиус>,,,,<отношение осей эллипса>

Первые две формы оператора CIRCLE используются для построения окружностей путем задания <координат центра> и <радиуса>. Значение радиуса окружности может быть определено арифметическим выражением. Если числовое значение арифметического выражения меньше 1, то будет построена точка, являющаяся центром окружности. При очень больших радиусах будет построена только часть окружности, попадающая на экран.

Третья и четвертая формы записи оператора CIRCLE служат для построения дуг окружностей. Положение дуги на окружности определяется <начальным углом дуги> и <конечным углом дуги>. При этом значения параметров могут быть заданы арифметическими выражениями. Дуга строится против часовой стрелки от точки окружности, зафиксированной начальным углом, до точки, зафиксированной конечным углом. Значения углов считаются заданными в радианах.

Для построения эллипсов применяются формы 5 и 6. Эллипс строится из окружности путем ее деформации. Можно построить лишь два типа эллипсов: горизонтальный (окружность, сжатая по оси Y) и вертикальный (окружность, сжатая по оси X). Тип эллипса определяет параметр отношения осей эллипса. Если значение параметра меньше 1, то строится горизонтальный эллипс; в противном случае — вертикальный эллипс. Степень деформации окружности заэллипса определяет параметр отношения осей эллипса>. Чем больше это значение отличается от 1, тем больше сжимается окружность.

Оператор CIRCLE, записанный в общем виде, используется для построения дуг эллипсов.

Пример 6.6. Построить чайник. Ниже приведена программа 6.6 и рис. 6.4 построения чайника.

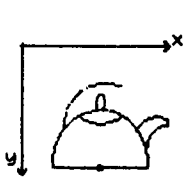


Рис. 6.4. Чайник

Программа 6.6

```
10 COLOR 1,15,15
20 SCREEN 2
30 CIRCLE (100,100),50,6.28,3.14   'корпус
40 LINE (50,100)-(150,100)         'дно
50 CIRCLE (100,44),5               'ручка крышки
60 CIRCLE (100,55),20,,3.14,6.28,.1      'крышка
70 CIRCLE (110,70),50,,3.14/3,3.14,.9    'ручка
80 CIRCLE (160,70),20,,3.14/2,3.14,.5    'верх носа
90 CIRCLE (177,90),30,,3.14/1.5,3.14/1.1,.8   'низ
                                              носа
100 LINE (160,60)-step(4,7)        'горло
110 GOTO 110
```

Закраска фигур. Закрашивание любых замкнутых областей цветом контура выполняет оператор.

Общая форма записи оператора:

PAINT <координата точки>, <код цвета>

Параметр <координата точки> определяет область закраски по отношению к контуру. Если точка содержится внутри области, охваченной контуром, то будет закрашена ее внутренняя часть, в противном случае — внешняя. Код цвета закраски должен совпадать с кодом цвета контура. Контур, составленный из разных цветов, считается незамкнутым. Закрашивание фигур с незамкнутым контуром или цветом, отличным от цвета контура, приводит к закраске всего рабочего поля.

Пример 6.7. Построить круг (закрашенная окружность) (программа 6.7).

Программа 6.7

```
10 COLOR 1,15,15
20 SCREEN 2
30 A$="C1R50D40L50U40BM+55,+4H30G30BM+25,+16
   C8R10D20L10U20BD6R10BL5D14"
40 DRAW"BM25,60XA$;"
50 GOTO 50
```

6.4. ПОСТРОЕНИЕ И ПРЕОБРАЗОВАНИЕ СЛОЖНЫХ РИСУНКОВ

Рассмотренных выше графических операторов достаточно для построения любого графического объекта. Однако в ряде случаев оказывается весьма трудоемким особенно кодирование рисунков, составленных из большого числа отрезков, кодирование изображений объектов с помощью указанных операторов. Для облегчения решения этой задачи в языке БЕЙСИК MSX введен специальный оператор DRAW. С его помощью сложные рисунки на экране можно создавать из отдельных, заранее заготовленных элементов, производя над ними простейшие преобразования: масштабирование, поворот и перенос. Заготовленные элементы хранятся в памяти компьютера. Каждому такому элементу присваивается имя. В процессе создания сложного изображения пользователь оперирует уже не с отдельными графическими примитивами (точками, отрезками, дугами окружностей), а с более крупными «конструкциями» — с заготовленными элементами. Общая форма записи оператора DRAW:

DRAW <строка символов>,

где <строка символов> — символьный код рисунка.

Параметр <строка символов> может быть записан в одной из следующих форм:

«<символьная константа>» — код рисунка;

<имя символьной переменной>, содержащей символьный код элемента рисунка;

X <имя символьной переменной>; «<символьная константа>» (X — буквенный символ — композиция кодов элементов).

Как видим, основой для составления параметра <строка символов> служит символьный код рисунка. Рассмотрим принципы его формирования.

Символьный код рисунка собирается в виде последовательности буквенных и цифровых символов, управляющих движением рисующего карандаша (графического курсора). Буквенные символы обозначают имена управляющих команд. Цифровые символы (целые числа) задают значения этих команд.

Типы управляющих команд. Возможны следующие типы команд:

перемещение в заданном направлении: U <число> — перемещение графического курсора вверх на <число> точек; D <число> — вниз; L <число> — влево; R <число> — вправо; E <число> — вправо вверх; F <число> — вправо вниз; G <число> — влево вниз; H <число> —влево вверх;

перемещение в произвольном направлении: M±±<число 1>,±<число 2>). <Число 1> и <число 2> задают величину перемещения по осям X и Y в положительном (знак+перед числами) или отрицательном (знак —) к осям направлении;

перемещение без рисования: B. Имя этой команды записывается только перед командами перемещения;

перемещение с возвратом: N. Команда N, записанная перед командой перемещения, предписывает вернуть графический курсор в точку, с которой началось данное перемещение;

назначение цвета: C <число>. Параметр <число> может принимать значения от 0 до 15. Команда C задает цвет рисующего карандаша.

Типы преобразующих команд. Кроме управляющих (рисующих) команд оператор GRAW имеет преобразующие команды. К ним относятся:

Перенос рисунка: BM <число 1>, <число 2> (не путать с M+<число 1>, +<число 2>!)

Параметры <число 1>, <число 2> задают координаты точки, которая послужит началом построения рисунка.

Поворот: A <число>. Параметр <число> может принимать одно из следующих четырех значений: 0 — поворот по часовой стрелке на 0°; 1 — на 90°; 2 — на 180°; 3 — на 270°.

Масштабирование: S <число>, где значение параметра <число> изменяется от 0 до 255. Команда S производит масштабирование элемента рисунка в <число>/4 раза.

Действие команд A и S отменяется только новой командой A и S соответственно.

Команды перемещения допускают задание числовых данных через переменные величины. В этом случае они записываются в виде

P=<имя переменной>;

где P — одна из букв U, D, L, R, E, F, G, H; M+= =<имя переменной>;,+=<имя переменной>; и M=<имя переменной 1>;,=<имя переменной 2>.

Пример 6.8. Построить дом (рис. 6.5).

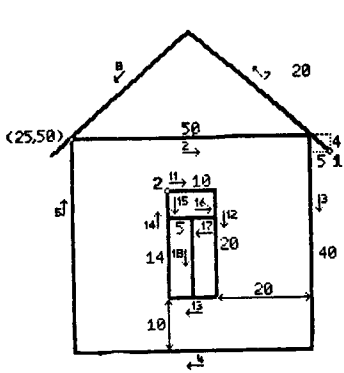


Рис. 6.5. Последовательность построения рисунка «Дом»:

1 — выбор черного карандаша; *2...5* — рисование; *6* — перемещение без рисования в точку *1*; *7*, *8* — рисование; 9 — перемещение без рисования в точку *2*; *10* — выбор красного карандаша

```
10 COLOR 1,15,15
20 SCREEN 2
30 A$="... из примера .8"
40 DRAW"BM10,50XA$;BM60,130A2XA$;A0"
50 GOTO 50
```

Пример 6.9. Симметричное отображение (рис. 6.6).

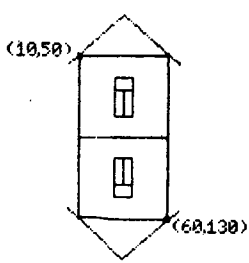


Рис. 6.6. Симметричное отображение рисунка

```
10 COLOR 1,15,15
20 SCREEN 2
30 CIRCLE (100,100),50,8
40 PAINT (100,100),8
50 GOTO 50
```

Пример 6.10. Масштабирование (рис. 6.7).

Рис. 6.7. Масштабирование рисунка


```
10 COLOR 1,15,15
20 SCREEN 2
30 A$="... из примера  .8"
40 DRAW"BM10,50XA$;BM100,70S6XA$;S0"
50 GOTO 50
```

6.5. ПРОСТЕЙШИЕ ЭЛЕМЕНТЫ МУЛЬТИПЛИКАЦИИ

Роль мультипликатора компьютер исполняет посредством функции SPRITE$ (функция для кодирования (прорисовки) мультипликационных картинок) и оператора PUT SPRITE (для отображения нарисованных картинок на экран).

Специфика построения изображения при помощи указанных операторов состоит в том, что процесс занесения данных в видеопамять в данном случае оказывается разделенным по времени на два этапа: на этап кодирования и записи кода изображения в специальную область памяти (назовем эту область — областью спрайтов) и на этап пересылки кода изображения из области спрайтов в видеопамять. Так как операции пересылки данных из одной области памяти в другую компьютер выполняет очень быстро, то заготовленная заранее картинка отображается на экране практически мгновенно. Поэтому такой способ построения изображений называют *быстрой графикой.*

Функция SPRITE$. Общая форма записи функции—

SPRITE$ (<номер картинки>),

где <номер картинки> — целочисленная константа. Значение этой константы может быть задано арифметическим выражением. Компьютер сам «следит» за целочисленностью результата арифметического выражения и в случае необходимости округляет его до целого путем отбрасывания дробной части числа.

Количество <номеров картинок> зависит от раз-

мера картинки. Допускается четыре типоразмера картинок: 0 — картинка размером 8 * 8 клеток (клетка — точка экрана); 1 — картинка размером 8* 8 увеличенных клеток (увеличенная клетка — четыре точки экрана); 2 — размером 16* 16 клеток; 3 — размером 16 * 16 увеличенных клеток.

Размер картинки задается оператором SCREEN, который в данном случае приобретает вид:

SCREEN <тип экрана>, <размер картинки>

Для картинок размером 8 * 8 можно использовать 256 номеров (от 0 до 255), для картинок 16 * 16 — 64 номера (от 0 до 63).

Код картинки (значение функции SPRITE $) задается путем присваивания этой функции значения строковой константы или переменной.

Пример 6.11. Кодирование картинки размером 8* 8 клеток (программа 6.8).

Программа 6.8

```
10 COLOR 1,15: SCREEN 2,0 : B$=""
20 FOR I = 1 TO 8
30 READ A$
40 B$=B$+CHR$(VAL("&B"+A$));
50 NEXT I
60 SPRITE$(1)=B$
100 DATA 11101101
110 DATA 11111111
120 DATA 11111111
130 DATA 11010010
140 DATA 11000000
150 DATA 11000000
160 DATA 11000000
170 DATA 11000000
```

Здесь код картинки формируется операторами 20...50 путем накопления этого кода в символьной переменной B$. Сама картинка представлена в виде «квадрата» из восьми операторов DATA, содержащих восьмисимвольные последовательности нулей и единиц. Единицы квадрата задают вид картинки. В приведенном примере нарисован флажок.

Кодирование картинки размером 16 * 16 несколько сложнее. Так как квадрат размером 16* 16 приходится разрезать на две полосы размером 16 * 8 (шестнадцатесимвольной двоичной строке — двоичному числу

соответствуют десятичные числа от 0 до 511, а аргумент функции CHR$ может принимать лишь значения от 0 до 255).

Пример 6.12. Кодирование картинки размером 16* 16 (программа 6.9).

Программа 6.9

```
 10 COLOR 1,15 : SCREEN 2 : B$="":C$=""
 20 FOR I = 1 TO 16
 30 READ A$
 40 B$=B$+CHR$(VAL("&B"+LEFT$(A$,8)))
 50 C$=C$+CHR$(VAL("&B"+RIGHT$(A$,8)))
 60 NEXT I
 70 SPRITE$(1)=B$+C$
100 DATA  0000011100000100
    ...     .   .   .
250 DATA  0000000001100010
```

При кодировании большого числа картинок стандартного объема памяти 20 байт, отводимого компьютером под символьные переменные, не хватает. Объем этой памяти можно увеличить при помощи оператора CLEAR.

CLEAR <размер области памяти символьных переменных>, где <размер области памяти символьных переменных> — целочисленная константа. Ее значение не должно превышать числа 2040.

Оператор CLEAR необходимо записывать всегда в начале программы, так как его исполнение приводит к инициализации (обнулению) памяти, отводимой под переменные.

Оператор PUT SPRITE. Общая форма записи оператора:

SPRITE <номер плана>, <координата>, <цвет картинки>, <номер картинки>

Воспроизвести картинку на графическом экране можно в одной из 32 специальных плоскостей (рис. 6.8). Номер плоскости задает параметр <номер плана>. В выбранной плоскости, имеющей размер рабочего поля экрана (192* 256), положение картинки (ее верхнего левого угла) задает параметр <координата> (расшифровку параметра см. в пункте «точка»).

Два оставшихся параметра определяют цвет и номер картинки.

На экране дисплея одновременно можно воспроизвести 32 картинки, по одной картинке в плоскости. При этом число картинок с одинаковой координатой y не должно превышать восьми для картинки 8* 8 (не больше 8 в ряд) и четырех для 16* 16.

Отображение нескольких картинок на одну и ту же плоскость приводит к стиранию предыдущих картинок

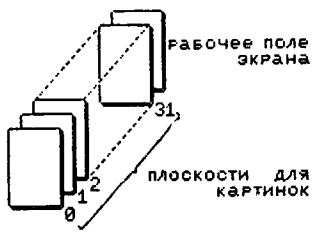


Рис. 6.8. Схема размещения плоскостей оператора PUT SPRITE по отношению к рабочему полю

последующими. Стирание всего экрана (в том числе и картинок) можно выполнить также при помощи оператора SCREEN.

Пример 6.13. Движение флажка (программа 6.10).

Программа 6.10

```
10
:          из примера 6.11
170
180 FOR X=0 TO 12.56 STEP 12.56/200
190 PUT SPRITE 1, (X*20, 100+50*SIN(X)),8,1'
    красный флажок движется в одной плоскости по
    синусоиде
200 NEXT
```

Комбинируя разными картинками, можно построить многоцветные картинки или картинки, содержащие подвижные элементы, например шагающих чело-

вечков, разных птиц и т. д. В первом случае монотонные картинки разных цветов достаточно расположить в разных плоскостях, во втором случае, наоборот, картинки с изображениями разных элементов необходимо отображать в одну плоскость.

6.6. ОСОБЕННОСТИ ВВОДА-ВЫВОДА ЧИСЛОВОЙ И СИМВОЛЬНОЙ ИНФОРМАЦИИ

Рассмотренные в гл. 5 операторы ввода-вывода числовой и символьной информации INPUT и PRINT ориентированы на работу только с текстовым экраном, графический экран им не доступен.

Для ввода-вывода информации в графическом режиме используются операторы INPUT$ и PRINT#, действия которых аналогичны действию INPUT и PRINT.

Оператор PRINT#. Общая форма записи оператора:

PRINT # <номер файла>, <список переменных>,

где <номер файла> — целочисленная константа от 1 до 15. Параметр <список переменных> расшифровывается подобно параметру <список переменных> оператора PRINT (см. гл. 5).

Оператор PRINT# предназначен для вывода информации, оформленной в виде файла (поименованного набора данных), на различные технические устройства, например на принтер, дисковод, графический экран и т. д. Поэтому перед выводом данных необходимо заранее оговорить устройство, на которое будет отображаться информация. Так, например, назначить графический экран в качестве устройства, служащего для вывода данных, можно при помощи оператора

OPEN «GRP:» FOROUTPUT AS # <номер файла>

Более подробно оператор OPEN описан в гл. 8. Здесь приведем лишь перевод ключевых слов оператора: OPEN — открыть, GRP — графический экран, FOROUTPUT — для вывода, AS — служебные символы.

Графический экран в отличие от текстового не имеет ни формата, ни позиционного деления на строки и

столбцы. Поэтому символы можно вывести в любом месте графического экрана, что достигается путем предварительного задания точки вывода при помощи оператора PSET.

Пример 6.14. Вывод текста на графический экран.

Программа 6.11

```
10 COLOR 1,15,15
20 OPEN "GRP: "FOROUTPUT AS#1
30 SCREEN2
40 PSET (170,10),15:PRINT#1, "ТЕКСТ"
50 PSET (140,20),15:COLOR 8:PRINT#1,"на графическом
   экране"
60 GOTO 60
```

В программе 6.11 первые три оператора подготовительные: назначают цвета, делают доступным для вывода текстов графический экран и назначают этот экран. Операторы 40 и 50 непосредственно выводят символы на экран, начиная с точки (170, 10) и (140, 20) соответственно.

Под каждый символ на графическом экране отводится квадрат со стороной из восьми точек. Сам же квадрат с помещенным в него символом фиксируется на экране верхним левым углом.

В операторе PSET цвет точки можно выбирать произвольным, он не влияет на цвет выводимых символов. Поэтому рекомендуем ставить точку цветом, невидимым для человеческого глаза (но не для оператора PRINT#), т. е. цветом рабочего поля. Цвет же символов можно изменять посредством оператора COLOR (см. оператор 50 в примере 6.14).

Функция INPUT $. Общая форма записи функции:

INPUT$ (<количество вводимых символов>)

Функция INPUT$ организует ввод любого набора символов заданной длины. В число вводимых символов могут входить и непечатаемые символы, такие, как символ возврат каретки, символы перемещения курсора и т. д.

Функция INPUT$ дает возможность полного управления вводом путем конструирования собственных процедур ввода.

Пример 6.15. Ввод числовых данных.

Программа 6.12

```
10 READ X,Y,A$
20 COLOR 1,15,15
30 OPEN "GRP:"FOR OUTPUT AS#1
40 SCREEN 2
50 PSET (X,Y),15:PRINT #1,A$
55 S$=""
60 X$=INPUT$(1)
70 IF X$=CHR$(13) THEN 110
80 S$=S$+X$
90 PSET (X+(LEN(A$)+1)*8, Y), 15:PRINT #1,S$
100 GOTO 60
110 A=VAL(S$)
120 DATA 10,50, Введите число
```

Приведенную программу 6.12 можно рассматривать как программу ввода числовых данных с клавиатуры в графическом режиме. Здесь оператор 10 присваивает значения переменным x, y и A$, которые используются в программе для сообщения пользователю о моменте ввода данных. Это сообщение выводится на экран оператором 50. Изменяя значения констант в операторе DATA, можно организовать вывод сообщений в разные места экрана. В операторе 60 предусмотрен ввод одного символа с клавиатуры и его запоминание в переменной с именем X$. Если введенный символ отличен от клавиши «возврат каретки» (ее значение задано в программе функцией CHR$ (13), число 13 — код ASCII символа ↵), то ввод будет продолжен. Предварительно значение нажатой клавиши добавится к символьной переменной S$ (оператор 80) и содержимое S$ распечатается на экране (оператор 90). Здесь координата X в операторе PSET вычислена с учетом вывода сообщения A$ (длина A$, умноженная на 8, определяет число точек, занятых сообщением A$), При нажатии клавиши «возврат каретки» ввод прекращается, а строка символов S$ преобразуется в число (действие функций VAL(S$) в операторе 110).

Возможности программы ввода в примере 6.15 примитивные. В ней полностью отсутствует, например, возможность редактирования вводимой информации. Поэтому предлагаем читателю в качестве упражнения дополнить эту программу необходимыми функциями ввода.

6.7. ПОСТРОЕНИЕ СТАТИЧЕСКИХ И ДИНАМИЧЕСКИХ ИЗОБРАЖЕНИЙ

Статическое изображение. Под статическим изображением будем понимать изображение, собранное из конечного числа геометрических примитивов, положение которых на экране не меняется и заранее известно.

Программа, рисующая статическое изображение, представляет собой линейную цепочку графических операторов, параметрами которых являются числовые константы. При написании такой программы разработчику требуется выбрать цвета, включить графический экран и правильно записать графические операторы. Поэтому при программировании статического изображения можно сымитировать экран дисплея у себя в тетради. Для этого, например, можно на листе бумаги построить рамку размером 34×24 клетки, положив сторону клетки равной восьмиэкранным точкам.

Нарисовав изображение на бумаге, его затем значительно проще воссоздать при помощи графических операторов на экране дисплея. При программировании нарисованного изображения (при вычислении координат) не забывайте, что экранная система координат может отличаться от реальной, т. е. может возникнуть необходимость пересчета координат точек. Например, для компьютера «Ямаха» этот пересчет нужен и может быть осуществлен по формулам

$$x_э = x_р; \quad y_э = 191 - y_р,$$

где $x_р$, $y_р$ — реальные координаты точки, а $x_э$, $y_э$ — экранные.

Пример 6.16. Статическое изображение «Лук». Построение данного изображения приведено на рис. 6.9 и в программе 6.13.

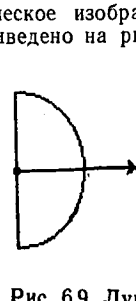


Рис. 6.9. Лук

Программа 6.13

```
10 COLOR 1,15:SCREEN 2
20 CIRCLE (10,50), 20,1,3*3.14/2, 3.14/2'лук
30 LINE ( 10,30)-(10,70),1' тетива
40 LINE (10,50)-(50,50),1' стрела
50 LINE - (47,47),1' наконечник
60 LINE - (47,53),1
70 LINE -(50,50),1
80 GOTO 80
```

Динамическое изображение. Под *динамическим изображением* будем понимать изображение, положение или вид которого может меняться с течением времени.

Обычно изменение изображений основано на использовании набора статических изображений, каждое последующее из которых лишь немногим отлича-

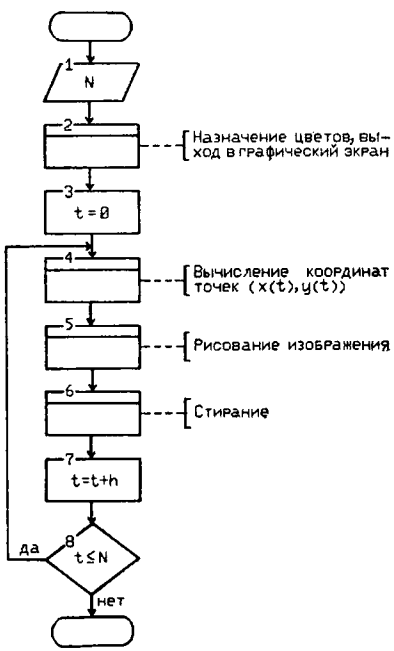


Рис. 6.10. Блок-схема построения динамического изображения

ется от предыдущего. Если просмотреть эти изображения, быстро сменяя одно другим, то возникает впечатление непрерывного плавного изменения изображения.

Реализуем этот способ на компьютере. Для этого сначала нарисуем изображение. Затем сотрем его и нарисуем снова, изменив при этом предварительно местоположение или вид изображения. Если при каждом новом построении местоположение изображения меняется незначительно, то получится плавное изменение изображения; в противном случае — скачкообразное. Программно это можно реализовать с помощью цикла (рис. 6.10).

По приведенной блок-схеме изображение изменяется $N/(h+1)$ раз. Заметим, что все построение динамического изображения основано на многократном преобразовании осей координат (блок 4). Только благодаря постоянному изменению положения осей координат и удается создать динамику.

Пример 6.17. Изменим условие предыдущего примера. Заставим стрелу вылететь из лука (см. рис. 6.8). Для этого введем движущуюся систему координат, переменные значения (X, Y). Пусть в начальный момент эта система координат находится в точке (10, 50) (конец стрелы).

Запишем координаты всех точек стрелы в новой системе координат:

точка (50,50) = (X+40, Y);
точка (47,47) = (X+37, Y −3);
точка (47,53) = (X+37, Y +3).

Задавая различные законы изменения X и Y от времени t, можно получить различные полеты стрелы.

Горизонтальный полет стрелы. Положим Y=const; в нашем случае Y=50, а X будем считать функцией от t. Например, если X(t) принять равным aX+b, где a и b — константы, то получим равномерный полет стрелы, а для $X=ct^2+bt+a$ ($c<0$) — равнозамедленный полет.

Полет стрелы с падением. Будем считать, что X и Y — функции от t. Например, X=vt, Y=h−gt, где v — скорость стрелы; h — высота, с которой стрела начала падать; g — ускорение падения.

Пример 6.18. Полет стрелы с падением.

Программа 6.14

```
10 INPUT"Введите время полета стрелы";N
20 COLOR 1,15: SCREEN 2
30 CIRCLE (10,50),20,1,3*3.14/2,3.14/2
40 LINE (10,30)-(10,70),1
50 FOR T=0 TO N STEP 1/60
60 X=10+80*T: Y=50+9.8*T^2
70 LINE (X,Y)-(X+40,Y)
80 LINE-(X+37,Y-3)
90 LINE-(X+37,Y+3)
100 LINE-(X+40,Y)
110 LINE (X,Y)-(X+40,Y),15
120 LINE-(X+37,Y-3),15
130 LINE-(X-37,Y+3),15
140 LINE-(X+40,Y),15
150 NEXT
```

В строке 60 программы 6.14 задан закон полета стрелы. Стрела выпущена со скоростью 80 единиц из точки с координатами (10,50). Операторы 70...100 рисуют стрелу, а 110...140 стирают ее (рисуя стрелу заново цветом рабочего поля).

Приведенный способ стирания стрелы не единственный. Можно, например, стирать изображение при помощи оператора CLS (оператор стирания рабочего поля экрана) или стирать при помощи закрашенных цветом рабочего поля прямоугольников.

За время полета стрела изменит свое положение N *60+1 раз. В приведенной программе стрела нарисована при помощи оператора LINE. Это медленно исполняемый оператор. Поэтому полет стрелы будет сопровождаться миганием (рисованием и стиранием стрелы). Если стрелу закодировать через функции SPRITE, а затем отобразить ее на экран при помощи оператора PUT SPRITE, то полет стрелы примет более динамичный характер.

Ниже приведем программы 6.18 и 6.19, рисующие на экране красивые узоры. Машинные узоры используют обычно для рекламы возможностей компьютера, в качестве заставок в книгах, журналах, на телевидении и т. д. В некоторых странах организуются даже выставки компьютерного искусства.

Программы, строящие узоры, несложные. Здесь важны выдумка и декоративный эффект.

Пример 6.19. Муаровый эффект (рис. 6.11).

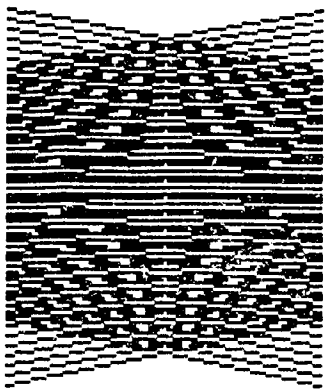


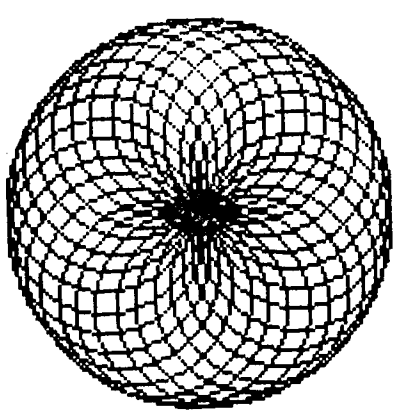
Рис. 6.11. Муаровый эффект

Программа 6.15

```
10 COLOR 1,15: SCREEN 2
20 FOR I=1 TO 49
30 Y=(I-26)*3
40 Y1=(I-26)*4
50 LINE (0,100+Y)-(255,100+Y1)
60 LINE (255,100+Y)-(0,100+Y1)
70 NEXT
80 GOTO 80
```

Пример 6.20. Кольцо (рис. 6.12).

Рис. 6.12. Кольцо



Программа 6.16

```
10 COLOR 1,15:SCREEN 2
20 READ R,N
30 FOR I=1 TO N
40 AL=6.28*I/N+1.57
50 CIRCLE (100+R*COS(AL),100+R*SIN(AL)),R
60 NEXT
70 GOTO 70
80 DATA 30,30
```

Обычно программы, рисующие узоры, строят по принципу многократного исполнения одного или нескольких графических операторов, имеющих в качестве параметров переменные величины. Значения этих величин подчиняются какому-либо закону. Например, в программе 6.16 многократно выполняется один оператор CIRCLE, рисующий семейство окружностей, центры которых перемещаются по другой окружности.

УПРАЖНЕНИЯ

Составить программы к рис. 6.13:

а) при помощи операторов LINE и CIRCLE (подсказка: программа будет короче, если изображение акробата оформить

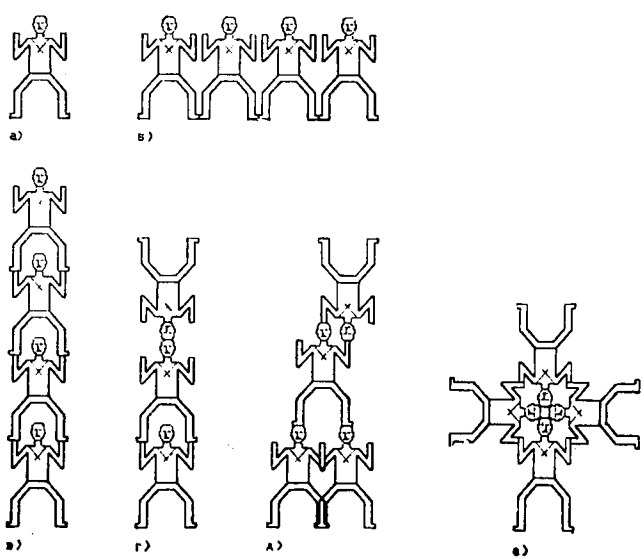


Рис. 6.13. Рисунок к управлению

в виде подпрограммы, входными параметрами которой являются координаты точки — начала рисования акробата);

б) при помощи оператора DRAW;

в) при помощи операторов SPRITE$ и PUT SPRITE$.

Глава 7

ОТЛАДКА И ТЕСТИРОВАНИЕ ПРОГРАММ

В гл. 7 описаны некоторые приемы отладки и тестирования программ, которые позволяют поэтапно устранить ошибки, допущенные не только на этапе программирования, но и при разработке алгоритма (или даже при постановке задачи).

7.1. КЛАССИФИКАЦИЯ ОШИБОК В ПРОГРАММАХ

Во время разработки сложных программ для компьютеров неизбежны различные производственные ошибки. Поэтому решение задач с помощью компьютеров включает особые этапы тестирования и отладки программ, где происходит обнаружение и исправление допущенных ошибок.

Тестирование и отладка зачастую поглощают больше времени, чем первоначальное написание программы. Эти этапы занимают, по оценкам специалистов, от 40 до 80 % общего времени разработки программы. Сам процесс тестирования и отладки во многом зависит от используемого языка программирования, компьютера, специфики самой программы, опыта и интуиции разработчика программы.

Тестирование и отладка программ различаются тем, что при тестировании устанавливается факт ошибки, а отладка выявляет причину и предлагает способы ее устранения. Эти этапы создания правильных программ чередуются и перекрывают друг друга.

Рассматривая отдельные приемы тестирования и отладки, мы будем опираться на возможности интерпретатора языка БЕЙСИК MSX и технику работы с персональным компьютером.

Источники ошибок в программах могут быть различны. В табл. 7.1 ошибки сгруппированы в соответствии с этапами общей схемы решения задач.

Таблица 7.1. **Виды ошибок при решении задач с помощью компьютера**

Вид ошибки	Основные причины появления ошибок
Получаемые результаты не соответствуют сформулированной задаче	Неправильная постановка задачи Выбор неадекватной математической модели
Программа выдает неточные решения или неэффективна	Разработан или выбран неправильный или неэффективный алгоритм
Логические ошибки	Неверно запрограммирован алгоритм, пренебрежение малыми параметрами модели и пр.
Синтаксические ошибки	Нарушение правил синтаксиса при программировании
Ошибки при выполнении операций	Деление на ноль, вычисление логарифма отрицательного числа, извлечение квадратного корня из отрицательного числа, переполнение.
Ошибки в данных	Ввод или обработка данных, не предусмотренных программой, неверно определены тип, диапазон изменения данных

Указанные в таблице ошибки могут быть обнаружены при тестировании и затем устранены в результате отладки.

Выявление интерпретатором синтаксических ошибок представляет собой начальный этап тестирования программы на компьютере. При выполнении каждого оператора программы интерпретатор предварительно анализирует правильность конструкции оператора и возможность его выполнения.

Каждая группа синтаксических ошибок имеет свой код и соответствующий комментирующий текст для выдачи пользователю. Так, при пропуске в операторе необходимого знака пунктуации, несогласованности скобок, неправильном формировании оператора или вызове системных функций выдается сообщение SINTAX ERROR («синтаксическая ошибка») с указанием номера строки, в которой обнаружена ошибка.

Осуществляется и диагностика ошибок, связанных с неверным взаимодействием операторов — отсутствием оператора окончания цикла, отсутствием описа-

ния массива, противоречивыми операторами и т. п. Сообщения о таких ошибках дифференцированы.

В рассматриваемой версии языка БЕЙСИК сообщение об ошибке указывает только номер ошибочной строки программы, и задача разработчика — отыскать ее. Следует иметь в виду, что иногда номера строк и типы ошибок в сообщениях могут быть ложными, поскольку порождены «незамеченными» ошибками в предшествующих операторах или исходных данных.

Существуют ошибки, обычно логические, которые интерпретатор языка не выявляет, так как с формальной точки зрения синтаксис программы корректен. Примеры такой ситуации: пропуск части программы, переход не на ту ветвь обработки информации после оператора условного перехода, неправильные параметры циклов, неправильная индексация массивов, неопределенные переменные. В этом случае программа будет выдавать какие-то результаты, но они будут неверными, что выявится в ходе тестирования программы.

7.2. КОНТРОЛЬ ПРАВИЛЬНОСТИ АЛГОРИТМОВ И ПРОГРАММ НА ЭТАПЕ РАЗРАБОТКИ

Различают три способа контроля правильности разработки алгоритма или программы без применения компьютера: просмотр, проверка и прокрутка.

Просмотр. Текст составленного алгоритма или программы внимательно читается с целью обнаружения описок и смысловых расхождений программы и алгоритма. Обычно осуществляется выборочный просмотр отдельных логически сложных фрагментов алгоритма.

Проверка. При проверке алгоритма или программы мысленно восстанавливается по блок-схеме или программе тот вычислительный процесс, который они определяют. В ходе этой процедуры осуществляется сверка с требуемым процессом. На время проверки следует как бы «забыть» о том, что должен делать проверяемый участок программы и «узнавать» об этом в ходе его проверки.

Приемы проверки программ сугубо индивидуальны. У каждого разработчика могут быть собственные приемы.

Прокрутка. Основой прокрутки является имитация выполнения алгоритма или программы разработчиком с целью более конкретного и наглядного представления процесса обработки информации, которую они определяют.

Прокрутка позволяет проверить программу как бы в ходе ее выполнения компьютером.

Пример 7.1. Допустим, имеется упорядоченный по убыванию массив X(i), i=1,2, ..., N, и число C. Требуется образовать массив Z(i), i=1,2, ..., N+1, объединив массив X и число C так, чтобы не нарушить упорядоченность.

Предположим, что в результате разработки получен алгоритм, изображенный на рис. 7.1.

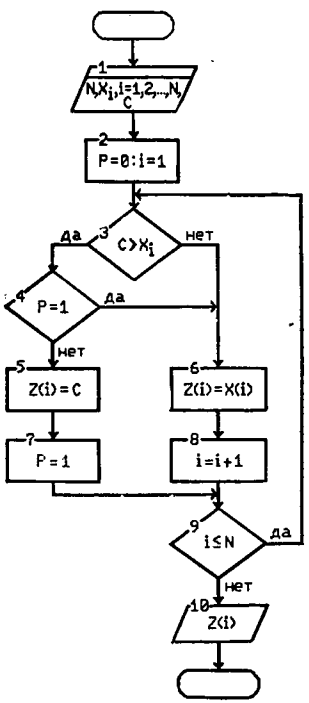
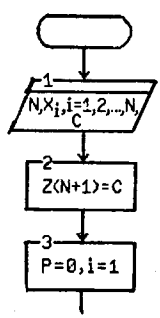


Рис. 7.1. Алгоритм получения упорядоченного массива

Рис. 7.2. Коррекция алгоритма получения упорядоченного массива



Отладку лучше начинать с начальных этапов разработки алгоритма и программы решения задачи, еще до выхода к компьютеру.

Для прокрутки зададимся конкретным набором величин N,

X(i), C. Процедуру прокрутки алгоритма представим в виде специальной табл. 7.2.

Анализируя полученные результаты, можно было бы утверждать, что алгоритм работает правильно, и на этом ручной контроль закончить. Однако для случая C>X(N) алгоритм работает неверно, в чем можно убедиться, составив таблицу прокрутки для такого набора данных.

Для исправления алгоритма можно вначале, например, под номером 2 (соответственно перенумеровав остальные блоки) поставить блок Z(N+1)=C (рис. 7.2).

Таблица 7.2. **Таблица прокрутки (исполнения) алгоритма**

Шаг алгоритма	Исходные данные				Промежуточная величина		Результаты			Проверка логических условий
	N	X (1)	X (2)	C	i	p	Z (1)	Z (2)	Z (3)	
1	2	5	2	3						
2					1	0				
3										3>5? Нет
4							5			
5					2					
6										2<=2? Да
7										3>2? Да
8										p = 1? Нет
9										
10								3		2<=2? Да
11										3>2? Да
12										p = 1? Да
13									2	
14					3					
15										3<=2? Нет
16					Печать		5	3	2	

Прокрутка позволяет разработчику алгоритма обнаружить более завуалированные ошибки, чем при простой проверке, хотя требует значительного объема ручной работы.

Результат прокрутки алгоритма в виде таблицы исполнения (табл. 7.2) является еще и удобным тестом для выполнения главной задачи — проверки правильности программы на компьютере, при которой в текст программы могут быть временно включены операторы вывода значений промежуточных переменных. Это делается для последующего сопоставления ре-

зультатов тестирования программы на компьютере и эталонных значений из таблицы.

С методической точки зрения прокрутка алгоритма особенно полезна для начинающих. В последующем это позволяет хорошо понимать работу программ и успешно их отлаживать. При наборе текста программы с клавиатуры возможно внесение различных ошибок (опечаток), поэтому желательно перед выполнением программы сверить ее с исходным текстом.

7.3. ТЕСТИРОВАНИЕ ПРОГРАММ

Решающим этапом, устанавливающим правильность программы для работы, является контроль по результатам выполнения на компьютере. Этот контроль осуществляется с помощью специальных тестов. *Тестом* будем называть информацию, состоящую из исходных данных, специально подобранных для отлаживаемой программы, и из сответствующих им эталонных результатов.

Для реализации тестов должны быть известны эталонные результаты, которые можно получить, например, выполняя вычисления вручную (как результат прокрутки).

Способы тестирования. Различают три способа тестирования: алгоритмическое, аналитическое и содержательное.

Алгоритмическое тестирование применяют для контроля этапов алгоритмизации и программирования. Тесты направлены на проверку правильности логики работы алгоритма и выполнения арифметических действий.

Аналитическое тестирование служит для проверки правильности метода решения задачи и возможности его работы на различных наборах данных. Это тестирование осуществляют на последних этапах отладки при анализе результатов пробного счета.

Содержательное тестирование служит для проверки правильности постановки задачи. Здесь используются качественные или статические оценки работы программы, анализируется физический смысл полученных результатов.

Содержательные и аналитические тесты проверя-

ют правильность работы программы в целом или крупных ее частей; алгоритмические тесты — в первую очередь работу отдельных операторов программы. Алгоритмические тесты помимо установления наличия ошибок в программе должны по возможности давать представление о местонахождении ошибок в программе для их дальнейшей локализации при отладке.

Разрабатывая тесты, следует стремиться к тому, чтобы они доказывали отсутствие ошибок. Но для этого требуется прогон такого большого числа тестов, что этапы тестирования и отладки могут растянуться на необозримое время. Поэтому при создании тестов стоит задача минимизации числа тестов и усилий разработчика.

Приемы формирования приемлемых тестов. Каждый линейный участок программы должен быть выполнен компьютером по крайней мере для одного теста, обеспечивающего контроль всех операций. Когда управлять прохождением определенных участков программы с помощью исходных данных трудно, можно создавать тесты для поблочного контроля программы. Тестирование операторов программы целесообразно осуществлять в случае их слабой зависимости, иначе приходится многократно проверять отдельные участки программы.

Арифметические выражения контролируют путем сверки с эталонными результатами. Здесь важно обратить внимание на точность такой сверки. Дело в том, что величины, входящие в арифметическое выражение, вносят различный вклад в результат вычислений. Поэтому может оказаться, что неправильно запрограммированное выражение для некоторых тестов будет давать правильные результаты.

Например, из того, что величина D совпала с эталоном для оператора D=A+C, не означает его правильность, так как для случая, когда A существенно больше C, случайная ошибка в знаке не будет обнаружена, если эталон определен не очень точно.

При создании тестовых наборов данных естественно задать такие величины, которые упрощали бы вычисления, обработку символьной и графической информации. Простые тестовые наборы облегчают ручной контроль результатов, но опасны из-за возможной неполноты контроля. Например, когда при вычислении

величины B=(A+C)/D в программе будут опущены скобки, но будет определен набор величин A=0, C=2, D=3 или любые A, C и D=1, ошибка в операторе может быть не обнаружена.

Основой эффективного тестирования является его плановость и систематичность.

Вначале определяется общий подход к тестированию программы, выделяя наиболее сложные логические конструкции, отдельные подпрограммы, требующие более основательного тестирования. Далее подбираются исходные данные, которые обеспечивают проверку работоспособности этих блоков. Таким образом заранее устанавливается, что необходимо проконтролировать и как это лучше выполнить. Планы тестирования могут включать как проверку всей программы, так и отдельных ее частей.

При решении сложных задач программа чаще всего состоит из основного модуля и нескольких подпрограмм. В этом случае тестирование лучше начинать с основного модуля. Работа подпрограмм заменяется так называемыми имитаторами или «заглушками». В задачу имитатора входит моделирование работы соответствующей подпрограммы, например, в виде передачи некоторой константы или сообщения о факте своего участия в работе. Такой нисходящий способ тестирования сложных программ сочетается с независимой отладкой отдельных подпрограмм.

Различают четыре вида тестов.

Основной, или *нормальный, тест,* с помощью которого анализ программы осуществляется по данным из области их допустимых значений.

Вырожденный тест, затрагивающий работу основных функций программы на множестве допустимых значений и обеспечивающий получение очевидного упрощенного результата.

Экстремальный тест, с помощью которого проверяется работа программы для граничных значений задачи. Работа программы носит особый характер, требующий и особого контроля.

Аварийный тест, с помощью которого проверяется реакция программы на возникновение различного рода исходных данных, данных, не имеющих физического смысла для конкретной задачи.

Пример 7.2. Рассмотрим набор тестов для проверки решения системы двух уравнений с двумя неизвестными:

$$ax + by = c;\ dx + ey = f. \qquad (7.1)$$

Эту систему можно решить по формулам:

$$x = (c - by)/a; \qquad (7.2)$$

$$y = (fa - ds)/(ea - db). \qquad (7.3)$$

Возможный набор тестов представлен в табл. 7.3.

Таблица 7.3. **Возможный набор тестов для решения системы двух уравнений с двумя неизвестными**

Номер теста	Коэффициенты						Примечания
	a	b	c	d	e	f	
1	2	3	8	2	1	7	Нормальный тест (x= =1, 75 y=—0,5)
2	2	1	1	4	3	4	Вырожденный тест. Один корень равен нулю (x=1, y=0)
3	1	3	6	1	2	4	Вырожденный тест. Один корень равен нулю (x=0, y=2)
4	4	6	8 или 7	2	3	4 или 5	Экстремальный и аварийный тест (ea—db)= =0 (бесконечное множество решений или их отсутствие)

Для каждой задачи набор тестов зависит от ее специфики. Однако даже сложную программу можно целенаправленно протестировать, предварительно разделив ее на отдельные программные модули. При этом эффективно проектировать программу сразу с учетом необходимости тестирования.

Начинать тестирование следует с ручной проверки, на ранних этапах разработки алгоритма и программы. Необходимо проверять правильность алгоритма и программы на упрощенном варианте.

Тестирование повторяется после каждого внесения изменений в программу.

7.4. ОТЛАДКА ПРОГРАММ. ЛОКАЛИЗАЦИЯ ОШИБОК

Когда в ходе тестирования программы установлено, что в некоторой ее части имеется ошибка, переходят к отладке программы. Возникает задача локализации ошибки — установление ее точного местоположения в программе.

В ходе поиска ошибок разработчик программы, анализируя полученные в компьютере результаты, проверяет различные предположения о характере и месте ошибки. Выдвигаемые гипотезы могут проверяться и отвергаться снова путем подбора специальных тестов. Таким образом, процесс локализации ошибок носит в значительной степени творческий, индивидуальный характер.

Однако если успех в быстром поиске ошибок программы зависит от способностей и опыта разработчика программы, то изучение и использование средств, помогающих локализовать ошибки, доступно каждому разработчику. Имеются в виду средства, которые существуют в используемом языке программирования.

Одним из первых шагов отладки должно быть распечатывание или вывод на экран дисплея всех введенных в компьютер данных. Множество программных ошибок связано с неверным вводом данных. Бесполезно тратится масса времени на поиск не существующей ошибки в программе, в то время как истинной причиной являются неправильные данные.

Данные следует выводить сразу же после их ввода (эхо-проверка данных).

Программа 7.1

```
10 INPUT "A,B"; AB
20 READ C1, D2, F$
30 FOR I=1 TO 5
40 READ A(I)
50 NEXT I
60 DATA 12.5, 1.41, METP, 2.1, 4.6, 7.8, 9.2, 6.4
70 PRINT "A="; A; "B="; B; "C1="; C1; "D2="; D2;  "F$="; F$
80 FOR I=1 TO 5
90 PRINT "A(";I;")="; A(I)
100 NEXT I
```

Операторы 70...100 в программе 7.1 используются для эхо-проверки вводимых данных.

В ряде случаев проверку исходных данных из принятого диалога можно организовать программным путем, предусмотрев соотвествующие реакции при вводе-выводе информации.

Удобным средством обнаружения ошибки в строке программы является последовательное разбиение этой строки на несколько операторов-строк, из проверки которых можно сделать однозначное заключение о местонахождении ошибок в данных.

Когда после пуска программы нет уверенности в том, что она начала выполняться или произошел преждевременный останов без выдачи сообщения об ошибке, или программа слишком долго работает, требуется информация о выполняемых интерпретатором операторах программы. Эта информация доставляется в течение так называемой *трассировки* программы, которая осуществляется с помощью специальных операторов TRON и TROFF языка БЕЙСИК, временно вставляемых в программу.

Оператор TRON включает трассировку, а TROFF ее выключает. Трассировке может подвергаться как вся программа, так и ее отдельные участки. В момент трассировки на экран выдаются номера выполняемых операторов.

Программа 7.2

```
100 TRON
110 X=Y^2+1
120 A=Y^2+SQR(X)
130 . . . . . .
140 . . . . . .
150 TROFF
```

После выполнения фрагмента программы 7.2 получим информацию о выполнении следующих строк:

110 120 130 140

Процесс обнаружения ошибок характеризуется выявлением двух мест в программе: точки обнаружения и точки происхождения. *Точка обнаружения ошибки* — это место в программе, где ошибка себя проявляет. *Точка происхождения ошибки* — это место, где

возникают условия для появления ошибки. Например, в операторе C=B/A затруднения с его выполнением появляются при A=0. В этом случае оператор будет точкой обнаружения ошибки, а точкой происхождения ошибки — какой-то предшествующий оператор, где A присваивается это нулевое значение.

Для обнаружения точек происхождения ошибок в текст программы могут вводиться отладочные операторы, сигнализирующие о прохождении важных участков программы и контролирующие значения отдельных меняющихся параметров.

Программа 7.3

```
100 F=X^2-X+1
110 PRINT "ЗНАЧЕНИЕ F ВЫЧИСЛЕНО"
. . . . . .
150 A=SIN(F)+0.5
160 IF A=0 THEN PRINT "В ОПЕРАТОРЕ 150 A РАВНО 0"
    ELSE C=B/A
```

Оператор 110 в программе 7.3 сообщает о вычислении параметра F, а оператор 160 контролирует значение величины A.

Выборочные сообщения можно использовать для обнаружения зацикливания, размещая их до и после цикла.

Выборочные распечатки параметров применяются для контроля числа итераций, значений индексов в массивах и т. п.

Отладочная печать вводится и для слежения за логикой работы программы. Переходы по различным логическим ветвям вычислений или выходы в подпрограммы и возврат из них могут сопровождаться соответствующими сообщениями, например:

```
150 IF X<0 THEN PRINT "ПЕРЕХОД В ВЕТВЬ X<0";Y=ABS(X+1)
    ELSE PRINT "ВЕТВЬ X>=0":Y=X+1
160 PRINT "ПЕРЕХОД В ПОДПРОГРАММУ"
170 GOSUB 400
180 PRINT "ВОЗВРАТ ИЗ ПОДПРОГРАММЫ"
. . . . . .
400 A=Y*SIN(Y):RETURN
```

Выдаваемые сообщения помогают ориентироваться в последовательности и логике работы программы.

Простым, но эффективным способом отслеживания за нормальным завершением работы программы является соответствующее сообщение, выдаваемое по ее окончании, например:

```
.....................
200 PRINT "КОНЕЦ":END
```

Если такого сообщения не выводить, можно не заметить ситуацию, когда программа не завершается (зацикливается, «зависает»).

Хорошим стилем написания программ является такой, при котором появляющиеся ошибки легко обнаруживаются и исправляются. Такой стиль называют еще *защитным программированием.*

Встраивание отладочных средств в программу есть не что иное, как защитное программирование.

Средства, отладки, предусмотренные в программе, называют стопорами ошибок. Их назначение — установить местонахождение ошибки и при возможности устранить. Для этого можно использовать оператор обработки ошибок:

ON ERROR GOTO <номер оператора>

Известно, что в средствах языка БЕЙСИК имеется перечень ошибок, встречающихся при работе программы. Коды ошибок обозначаются числами 1...255. Перечень стандартных синтаксических ошибок обозначается кодами 1...71, а начиная с 72 этот перечень может дополняться ошибками пользователя. При возникновении любой такой ошибки можно предусмотреть необходимую реакцию программы (информирование, устранение или переход в другую часть). Для этого создаются специальные подпрограммы обработки ошибок. Завершаются такие подпрограммы оператором

RESUME <параметр>.

В качестве <параметра> может использоваться ноль — для возврата на оператор, в котором произошла ошибка, слово NEXT — для передачи управления оператору, следующему за ошибочным; наконец, номер оператора, которому передается управление.

Например, в программе

```
10 ON ERROR GOTO 100
20 INPUT X
30 Y=SIN(X)/X
40 PRINT "Y=";Y
50 END
100 PRINT "ЭТО ЗНАЧЕНИЕ X ПРИВОДИТ К НЕОПРЕДЕЛЕННОСТИ"
110 Y=1
120 RESUME NEXT
```

подпрограмма обработки ошибки состоит в замене недопустимого значения Y при X=0 его предельным значением

$$Y = \lim_{x \to 0} \frac{\sin x}{x} = 1$$

и сообщением об ошибке.

Ввод кодов ошибок пользователя осуществляется оператором ERROR <целочисленное выражение>.

Значением этого выражения могут быть числа 0...255.

Оператор ERROR имитирует появление ошибки.

Например, в фрагменте программы

```
10 ON ERROR GOTO 200
. . . . . .
40 X=A^2-6*A+1
50 IF X<=0 THEN ERROR 150
60 Y=LOG(X)+8
70 Z=(Y-2)/2
80 IF Z>14.5 THEN ERROR 160
90 D=EXP(10*Z)
. . . . . .
200 IF ERROR=150 THEN PRINT "ЗНАЧЕНИЕ X<=0":
    X=1E-30:RESUME 60
210 IF ERROR=160 THEN PRINT "Z>14.5, ЗНАЧЕНИЕ D ВЫЧИСЛЕНО
    ПРИ Z>14.5":Z=14.5:RESUME 90
```

контролируются и корректируются критические значения переменных X и Z.

Операторы 50 и 80 указывают на наличие ошибок, кодируемых, например, числами 150 и 160. Первый из них объявляет ошибку при значении X<=0 (предотвращает вычисление при таком аргументе), а второй при Z>14.5 — вычисление EXP(X) возможно при X<=14.5. Если такие ситуации в программе возникают, то осуществляется переход к подпрограммам обработки ошибок, в которых операторы 200 или 210 сообщают о соответствующих ошибках. В этих же подпрограммах ошибки можно исправить и перейти на продолжение выполнения программы.

Для обработки ошибок можно использовать специальные функции ERR и ERL.

Функция ERR возвращает номер последней зафиксированной ошибки (например, X=ERR), а функция ERL — номер строки, где произошла такая ошибка (например, Y=ERL).

УПРАЖНЕНИЯ

7.1. Возьмите какую-нибудь простую программу и подсчитайте в ней количество различных логических путей. Возможно ли для этой программы исчерпывающее тестирование.

7.2. Для программы поиска наибольшего и наименьшего элементов некоторого массива составьте подробную таблицу исполнения и протестируйте с ее помощью программу.

7.3. Запрограммируйте вычисление, для которого

$$y = \sum_{N=1}^{100} \frac{1}{xN^3}.$$

Предусмотрите сообщение об ошибке при $x=0$.

7.4. Задана формула

$$y = \frac{a}{bc} - \sqrt{c+d}$$

Подготовьте примеры для тестирования этой формулы в нормальных, исключительных и экстремальных ситуациях.

7.5. Напишите программу вычисления величины

$$S = \left(\sum_{i=1}^{10} \frac{x(i) - a}{b} \right)^{1/2}$$

Массив $X(i)$, $i=1,2,\ldots,10$, известен. Параметры a и b также определены. Организуйте проверку и обработку возможных ошибок в данных.

Глава 8

БАЗОВОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

Программное обеспечение необходимо для решения любых задач по обработке информации с помощью компьютера. В данной главе рассматриваются основные программы, без которых немыслима работа персонального компьютера (ПК). Именно такие программы и называют базовым *программным обеспечением* (ПО).

8.1. ОПЕРАЦИОННЫЕ СИСТЕМЫ ПЕРСОНАЛЬНЫХ КОМПЬЮТЕРОВ

Операционные системы (ОС) обеспечивают взаимодействие всех программ пользователя с внешними устройствами (дисплеями, магнитофонами, дисководами и др.), распределяют оперативную память между программами, реагируют на различные события при обработке информации. В различных моделях компьютеров используются ОС разной конфигурации и с разными возможностями.

Основные компоненты ОС. Полная конфигурация ОС для персонального компьютера должна содержать следующие основные компоненты: файловую систему; драйверы внешних устройств; процессор командного языка.

Файл — это порция определенным образом организованной информации. Файлом может являться программа, текст, закодированное изображение и пр. Файлы хранятся в специально отведенных для них операционной системой участках памяти на внешних магнитных носителях, на гибких или жестких магнитных дисках, а также на магнитных лентах. Каждый файл имеет имя, присвоенное ему пользователем. Имена всех файлов, хранимых на данном диске, регистрируются системой в каталоге, который хранится на этом же диске. С помощью командного языка ОС пользователь может читать содержимое каталога, т. е. узнавать, какие файлы хранятся на данном диске, переименовывать зарегистрированные в каталоге файлы или вовсе удалять их. Каталог может иметь собственное имя и может быть зарегистрирован в другом каталоге более высокого уровня наравне с именами обычных файлов; в таком случае файловая система становится иерархической: в ней хранятся каталоги разных уровней.

Для обеспечения работы пользователя с ОС оказалось удобным распространить понятие файла на любой источник или потребитель информации в компьютере, будь то принтер, дисплей, клавиатура и др. Такая трактовка понятия файла удобна для взаимодействия программ пользователя с внешними устройствами, так как позволяет работать с внешними устройствами, как с обыкновенными файлами. Эта же

трактовка использована, например, в языке БЕЙСИК MSX для взаимодействия программы пользователя с графическим экраном (см. гл. 6) как с файлом «GRP:».

Файловая система — наиболее крупная составная часть ОС персонального компьютера. Именно от файловой системы зависит удобство работы пользователя, скорость доступа к информации, возможность создания хороших баз данных и т. д.

Персональный компьютер может иметь большой набор внешних устройств (ВУ). Помимо стандартных ВУ — дисплея, клавиатуры, гибких дисков, жестких дисков и принтера, к ПК могут подключаться дополнительные устройства ввода-вывода: графопостроители, планшеты, манипуляторы, а так же устройства для связи с телефонными линиями, аналого-цифровые и цифро-аналоговые преобразователи и другое оборудование. Обеспечивать работу широкого набора ВУ — одна из важнейших функций ОС. Для выполнения этой функции существуют *драйверы* — специальные программы, управляющие внешними устройствами. Каждому ВУ соответствует свой драйвер.

Драйверы стандартных ВУ образуют в совокупности *базовую систему ввода-вывода*, которая обычно заносится в постоянное запоминающее устройство (ПЗУ) компьютера и загружается в память при его включении независимо от того, подключены ли в данный момент стандартные ВУ к компьютеру. Драйверы дополнительных ВУ хранятся в их собственных ПЗУ и стыкуются с ОС при включении этих устройств.

Во всякой ОС имеется *командный язык*, с помощью которого пользователь описывает те или иные действия (например, обращение к каталогу, запуск программ). Анализ и исполнение составленных на этом языке команд пользователя осуществляются входящей в состав ОС специальной программой — *процессором командного языка*.

Подчеркнем, что процессор командного языка — это не электронный блок ПК, а специальная программа, обеспечивающая взаимодействие ПК с пользователем. Процессор командного языка анализурует и выполняет не только отдельные команды, но и составленные пользователем заранее довольно сложные последовательности таких команд. Таким образом,

пользователь имеет возможность писать целые программы на командном языке ОС.

Основные типы ОС для ПК. В настоящее время создано несколько десятков различных типов ОС для ПК. В простейших учебных и домашних ПК функции ОС реализуются с помощью команд, включенных прямо в БЕЙСИК (вспомните операторы LIST, RUN, NEW из § 5.9 — по своему смыслу это команды управления компьютером).

На более мощных ПК операционная система отделена от языков программирования. В настоящее время выделились три наиболее распространенных ОС:

1) операционная система CP/M для ПК с 8-разрядными микропроцессорами;

2) операционные системы MS-DOS, CP/M-86, ОС M86 для ПК с 16-разрядными микропроцессорами;

3) операционная система UNIX для ПК с 16- и 32-разрядными микропроцессорами.

Все остальное программное обеспечение для ПК также делится на три обычно несовместимые между собой группы, связанные с тремя перечисленными ОС. Ниже рассматриваются характеристики этих трех ОС, а также ОС MSX-DOS, примыкающей к MS-DOS и ориентированной на учебные ПК.

Операционная система CP/M положила начало созданию ОС для ПК. Она была разработана в 1974 г. и установлена на многих ПК с 8-разрядными микропроцессорами. Успех CP/M был обусловлен ее предельной простотой, компактностью и возможностью быстрой настройки на разные конфигурации ПК. Настройка на конкретную конфигурацию осуществляется с помощью специального файла, в котором описываются подключенные ВУ.

Система CP/M занимает всего несколько килобайт памяти, хранится на диске и состоит из трех файлов:

BIOS — базовая система ввода-вывода (BASIC INPUT/ONTPUT SYSTEM);

BDOS — базовая дисковая операционная система (BASIC DISK OPERATING SYSTEM);

CCP — процессор командного языка (CONSOLE COMMAND PROCESSOR).

Файл BIOS содержит драйверы внешних устройств и требует настройки на конкретные ВУ. BDOS пред-

ставляет собой файловую систему, реализующую около 40 различных функций. Эта файловая система довольно ограничена и не допускает иерархической организации каталогов. Системой CCP выполняются следующие команды: USER (от англ. user — пользователь) — настройка на нового пользователя; DIR (от англ. directory — каталог) — выдача каталога диска; TYPE (от англ. type — печатать) — вывод содержимого файла на экран дисплея; REN (от англ. rename — переименовать) — переименование файла; ERA (от англ. erase — стирать) — удаление файла. Другие команды ОС реализованы с помощью служебных программ, работающих под управлением ОС CP/M, но не входящих в ее состав.

Операционная система CP/M заметно влияла на техническую политику многих разработчиков ПК. Эта система считается одной из лучших ОС для ПК с 8-разрядными микропроцессорами.

Операционная система MS-DOS превратилась в настоящее время в стандарт ОС для ПК с 16-разрядными микропроцессорами. В состав MS — DOS входят примерно такие же компоненты, как и в CP/M, но она имеет более широкие возможности. К основным достоинствам MS-DOS относятся: развитый командный язык; возможность организации иерархической файловой системы; возможность работы с внешними устройствами как с файлами; возможность подсоединения драйверов дополнительных ВУ; возможность выполнения так называемых фоновых заданий, не прерывающих диалог пользователя с ПК.

Для работы MS-DOS требуется около 60 Кбайт памяти, что существенно больше, чем для CP/M.

Операционная система UNIX широко распространилась в 70-е годы. Она была разработана первоначально для ПК фирмы DEC, но благодаря хорошо продуманным возможностям быстро нашла применение для других типов ПК с 16- и 32-разрядными микропроцессорами. Ее распространению способствовали, во-первых, заранее предусмотренная при проектировании простота переноса на другие ПК (мобильность), во-вторых, упрощенный диалог пользователя с ПК.

Операционная система UNIX располагает развитой файловой системой и мощным командным язы-

ком. Кроме того, в состав UNIX входит множество служебных программ — утилит, обеспечивающих выполнение разнообразных функций при разработке программного обеспечения.

При разработке MSX-DOS была учтена необходимость поддержки обширного программного обеспечения, работающего под управлением ОС CP/M, и предусмотрена совместимость с файлами, созданными под управлением MSX-DOS. В отличие от MS — DOS, допускающей подсоединение драйверов дополнительных внешних устройств, система MSX-DOS ориентирована на стандартный набор аппаратных средств учебного компьютера, включающий оперативную память объемом не менее 16 К, постоянную память объемом 32 К с встроенным интерпретатором языка БЕЙСИК, цветной графический дисплей разрешающей способностью 256 на 192 точек и 16 цветами, трехголосный звуковой генератор на восемь октав, принтер и дисковод.

8.2. КОМПИЛЯТОРЫ И ИНТЕРПРЕТАТОРЫ

Компиляторы и интерпретаторы являются важной составляющей базового программного обеспечения, которая обеспечивает выполнение программ пользователя, написанных на языках высокого уровня. Об этих компонентах программного обеспечения уже упоминалось в гл. 1 и 5. Здесь мы рассмотрим их более подробно.

Компиляторы. Как известно, каждый тип компьютера имеет свою систему команд, т. е. тот набор машинных команд, который он может выполнить. Поэтому программа, под действием которой обрабатываются данные, должна состоять из этого набора команд. Однако пользователь составляет программу на языках более высокого уровня, что значительно облегчает труд и делает его более производительным.

Программа, написанная на языке высокого уровня (исходная программа), не может быть непосредственно исполнена компьютером. Прежде ее необходимо преобразовать в объектную программу, представленную в двоичных кодах машинных команд. Эта задача решается с помощью специальной программы, называемой *транслятором*. По способу преобразова-

ния транслирующие программы подразделяются на компиляторы и интерпретаторы.

Компилятор в процессе работы осуществляет полный перевод исходной программы в объектную. Объектная программа при отсутствии в ней ошибок может быть загружена в память компьютера и исполнена.

Каждый язык программирования высокого уровня имеет свою грамматику, определяющую допустимую форму (синтаксис) предложений языка. Поэтому функции компилятора заключаются в установлении соответствия написанных программистом предложений структуре данного языка и генерации соответствующих машинных кодов.

Предложение исходной программы рассматривается не просто как строка символов, а как последовательность, состоящая из некоторых более крупных образований — *лексем*. Лексемами могут быть: ключевое слово, имя переменной, арифметический оператор и т. д. Просмотр исходного текста с целью распознания и классификации различных лексем называется лексическим анализом.

После того как лексемы выделены, каждое предложение программы просматривается как некоторая конструкция языка, т. е. происходит его синтаксический анализ.

Последним шагом процесса трансляции является генерация объектной программы (объектного кода). При этом одному оператору (строке) исходной программы будет соответствовать несколько машинных (ассемблерных) команд. Проиллюстрируем это на примере компиляции небольшого фрагмента программы на языке ПАСКАЛЬ для ПК «Ямаха»:

```
S:=0; FOR I:=1TONDOS:=S+1;
```

Результатом компиляции будет следующий объектный код и ассемблерная программа:

Адрес	Машинный код	Команды Ассемблера	
202D:	21 00 00	LD	HL, 0000
2030:	22 B6 CB	LD	(C8B6),HL
2033:	21 01 00	LD	HL, 0001
2036:	E5	PUSH	HL
2037:	2A B8 C8	LD	HL, (C8 B8)
203A:	D1	POP	DE
203B:	CD 35 06	CALL	0635
203E:	7A	LD	A, D

```
203F:   B3          OR    E
2040:   CA 5C 20    JP    Z, 205C
2043:   D5          PUSH  DE
2044:   22 BA C8    LD    (C8 BA), HL
2047:   2A B6 C8    LD    HL, (C8 B6)
204A:   E5          PUSH  HL
204B:   2A BA C8    LD    HL, (C8 BA)
204E:   D1          POP   DE
204F:   19          ADD   HL, DE
2050:   22 B6 C8    LD    (C8 B6), HL
2053:   2A BA CB    LD    HL, (C8 BA)
2056:   23          INC   HL
2057:   D1          POP   DE
2058:   1B          DEC   DE
2059:   C3 3E 20    JP    203E
205C:   CD 52 16    CALL  1652
```

Таким образом, компиляция включает в себя три основных шага — лексический анализ, синтаксический анализ и генерацию машинных кодов. Однако это совсем не значит, что компилятор должен делать три просмотра транслирующей программы. Для некоторых языков программирования компиляция производится за один просмотр.

В процессе лексического и синтаксического анализов выявляются некоторые синтаксические ошибки, допущенные пользователем при написании программы. Этими ошибками могут быть: неправильная запись оператора, имен переменных, параметров, меток и др. После компиляции эти ошибки могут быть распечатаны и в дальнейшем исправлены пользователем.

Интерпретаторы. Интерпретатор, так же как и компилятор, обрабрабатывает исходную программу, написанную на языке высокого уровня. Основное различие состоит в том, что интерпретатор непосредственно исполняет некоторое внутреннее представление исходной программы, а не транслирует ее в машинные коды.

Интерпретатор также осуществляет лексический и ситаксический анализы и трансляцию исходной программы. При этом возможно использование различных внутренних представлений программы пользователя. Под управлением внутреннего представления каждого оператора исходной программы производится вызов соответствующей подпрограммы в машинных кодах, обеспечивающей выполнение предписываемых операций. Результаты интерпретации

фиксируются в памяти по частям и только на период, необходимый для их использования. Таким образом, интерпретаторы читают, расшифровывают и выполняют операторы исходной программы.

Интерпретация исходной программы проще и быстрее, чем компиляция в машинные коды. Однако выполнение интерпретатором внутреннего представления программ за счет многократного обращения к подпрограммам осуществляется гораздо медленнее, чем выполнение машинных кодов, сгенерированных компилятором. Поэтому интерпретатор нецелесообразно использовать в тех случаях, когда требуется высокая скорость решения задач. Кроме того, интерпретирующая программа вместе с исходной программой должна постоянно находиться в памяти компьютера, занимая достаточно большой ее объем. Этот недостаток интерпретаторов особенно проявляется при наличии дефицита памяти.

К преимуществу интерпретатора по сравнению с компилятором следует отнести реализацию в нем средств отладки программ, обеспечивающих во время выполнения программ автоматическую распечатку поименованных данных, трассировку операторов исходной программы с указанием номеров строк, в которых они содержатся, и т. д. Интерпретация исходной программы осуществляется в диалоговом режиме с пользователем. Поэтому пользователь имеет возможность следить за выполнением введенной программы на экране дисплея. По сообщениям, выдаваемым интерпретатором об ошибках, пользователь исправляет ошибочные операторы или строки программы, а после исправления снова вводит их в компьютер для исполнения. Благодаря этому процесс отладки программ в интерпретирующих системах занимает меньше времени, чем в компилирующих.

8.3. ОБЕСПЕЧЕНИЕ РАБОТЫ С ФАЙЛАМИ

Структура файловой системы. Для идентификации (обозначения) файла используют имя и тип файла. Имя файла может состоять из 1...8 букв, цифр и некоторых знаков, расположенных в произвольном порядке. Тип файла отображает характер хранимой информации. Задание типа производит сам пользова-

тель или программа, порождающая файл (например, при компиляции исходная программа преобразуется в объектный файл типа OBJ). Тип может состоять из 1..3 букв или цифр, а также других символов или же вообще отсутствовать. Имя и тип файла образуют полное имя файла. При наличии типа он отделяется от имени файла символом «.» Например, TEXT.DAT, NEW.TXT, ACOUNT.BAS и т. д. Иногда тип файла называют расширителем имени файла.

При создании файла или изменении его содержания автоматически регистрируются дата и время, снятые с текущих показаний календаря и часов системы. Имя, тип, дата и время являются атрибутами файла, которые фиксируются в оглавлении файлов — ката-

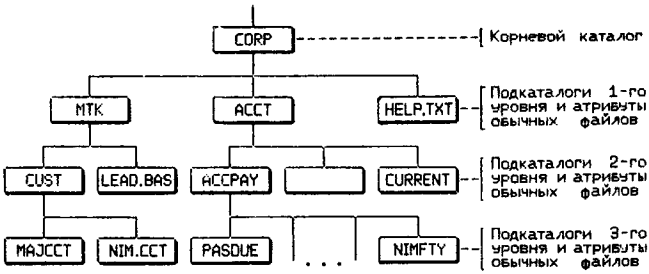


Рис. 8.1. Пример древовидного каталога «Корреспонденты»

логе. *Каталог* — это группа атрибутов файлов, размещенных на одном носителе (дискете, диске).

Каталоги могут иметь простую линейную структуру (в MSX-DOS). Тогда на дисковом носителе содержатся только обычные файлы. В развитой файловой системе используются сложные структуры каталогов, имеющие вид дерева (в MS-DOS). Корень этого дерева соответствует системному (корневому) каталогу. Исходящие от корневого каталога неразветвляющиеся ветви соответствуют обычным файлам, а разветвляющиеся ветви — другим, подчиненным каталогам (подкаталогам). Системный каталог всегда присутствует на диске (рис. 8.1). В нем регистрируются обычные файлы и подкаталоги 1-го уровня. В свою очередь в подкаталоге 1-го уровня могут также регистрироваться обычные файлы и подкаталоги 2-го уровня и т. д. В результате образуется иерархиче-

ская многоуровневая система, в которой месторасположение файла определяется по цепочке подчиненных каталогов путем описания маршрута имен этих каталогов. Для каталогов действует то же правило образования имен, что и для файлов, так как каталоги — это файлы.

Когда ОС делает опрос с целью поиска или создания файла, ей необходима такая последовательность данных: имя накопителя, каталог с файлом, имя файла. Для обозначения накопителей, используемых в системе, употребяются буквы A, B, C и т. д., после которых ставится символ «:» (например, A:). Имя накопителя в совокупности с именем файла образуют спецификацию файла. Примеры спецификаций: C: CUST.BAS, A: TXT.DAT, B: PROGRAM3. Каталог, содержащий файл, задается маршрутом имен подчиненных каталогов в порядке, начиная с имени подкаталога 1-го уровня или текущего каталога и кончая именем подкаталога, в котором зафиксирован данный файл. Если маршрут начинается с имени подкаталога 1-го уровня, т. е. от корня системного каталога, то перед ним ставится символ «\»; если с имени текущего каталога, то этот символ опускается. Имена подкаталогов в маршруте отделяются друг от друга этим же символом. Текущим каталогом является последний каталог, к которому обращалась операционная система. Отметим, что в MSX — DOS имеется только текущий каталог. Примеры описания маршрутов: \MTK\CUST\MAJCCT, ACCT\ACCTPAY\PASTDUE.

В первом случае описание маршрута идет от корневого каталога; подкаталог MTK является подкаталогом 1-го уровня. Во втором случае маршрут описан в текущем каталоге ACCT.

При использовании в системе нескольких дисковых накопителей в маршрут может быть включено имя накопителя, если он не является текущим (рабочим) накопителем. Имя накопителя должно предшествовать маршруту имен каталогов. Например,

B:\ACCT\ACCTPAY\ PASTDUE.

Данный маршрут указывает на элементы подкаталога 3-го уровня PASTDUE накопителя B.

Маршрут может быть использован как префикс к

имени файла. Он отделяется от имени файла тем же разделителем «\». Например:

A:ACCT\CURRENT

Здесь файл CURRENT располагается в текущем каталоге ACCT накопителя A.

Итак, для полного задания файла необходимы следующие элементы: имя накопителя, маршрут имен каталогов и собственное имя файла, содержащиеся в последнем каталоге маршрута. Имя накопителя и маршрут могут отсутствовать. В MSX-DOS маршрут имен каталогов всегда отсутствует.

Команды ОС для работы с файловой системой. Для работы с файловой системой в ОС используется группа команд, предоставляющих пользователю такие возможности, как вывод (распечатка) содержимого каталога, создание нового (пустого) каталога, смена текущего каталога, удаление каталога; копирование, переименование, удаление и распечатку содержимого файла любого подкаталога и др. Рассмотрим некоторые, наиболее часто употребляемые команды в операционной системе MSX-DOS.

Программы ОС MSX-DOS оформлены в виде двух файлов: MSXDOS. SYS и COMMAND. COM, находящихся на дискете. При включении компьютера или при нажатии кнопки RESET (сброс) начинает работать специальная программа самозагрузки, всегда находящаяся в постоянном запоминающем устройстве компьютера. Она обращается к накопителю A для пересылки операционной системы в оперативную память компьютера, т. е. реализует процесс, называемый загрузкой операционной системы. Если накопитель A выключен или в нем нет дискеты с двумя указанными выше файлами операционной системы, то автоматически произойдет загрузка интерпретатора языка БЕЙСИК. В случае успешной загрузки ОС на экране возникает подсказка a>, стоящая перед курсором и напоминающая о том, что накопитель A является текущим (рабочим) накопителем и в дальнейшем имя A этого накопителя можно опускать во всех командах ОС.

Команда DATE. Теперь после подсказки a> пользователь может вводить команду. Для этого ему следует набрать команду на клавиатуре и ввести ее в

машину клавишей «возврат каретки» (ВК). Обычно первой командой пользователя является команда установки даты: DATE mm — dd — уууу, где mm — месяц, dd — день, уууу — год. Например, команда DATE 01—25—1989 устанавливает дату 25 января 1989 г. Операционная система записывает дату на дискету.

Спросить установленную дату пользователь может командой DATE. В ответ на эту команду ОС отображает на экране установленную дату с указанием дня недели, который определяется по специальной программе. Первоначально, как уже говорилось, ОС обращается по умолчанию к накопителю А. Если необходимо переназначить текущий накопитель, то нужно ввести обозначающую его букву, затем двоеточие и нажать клавишу ВК. Например, на команду B: (ВК) операционная система ответит подсказкой b>, говорящей о том, что теперь текущим накопителем будет накопитель В.

Команда DIR. Чтобы вывести на экран перечень всех файлов, записанных на дискете, пользователю следует ввести команду DIR (от англ. directory — справочник). На экране это выглядит так:

```
a>DIR
```

В данном случае каталог диска, находящегося в накопителе А, выводится на экран после нажатия клавиши ВК. Каталог может содержать, однако, так много имен файлов, что его невозможно вывести на экран целиком. В подобной ситуации для построчного вывода каталога на экран команда DIR снабжается суффиксом/P, т. е. на экране получается

```
a>DIR/P.
```

Иногда пользователю необходимо обратиться сразу к некоторой группе файлов, а не работать с каждым файлом в отдельности. Для этого используются специальные символы (шаблоны) «?» и « * ».

Вопросительный знак является единственным неоднозначно интерпретируемым символом в имени файла. Например, имена PROGRAM 1. BAS, PROGRAM 2. BAS, PROGRAM 3. BAS соответствуют групповому имени PROGRAM ?. BAS.

Звездочка обозначает любое количество неоднозн-

начно интерпретируемых символов. Например, групповое имя PROG*.BAS. будет обозначать любое имя файла, которое начинается на PROG, при условии, что его расширение начинается с буквы B. Комбинация же символов *.* соответствует вообще любому имени файла.

Все файлы на одной дискете должны иметь разные имена, но на разных дискетах могут использоваться файлы с одним и тем же именем. Следовательно, компьютер с несколькими накопителями может одновременно работать с несколькими дискетами, содержащими файлы с одинаковыми именами. Чтобы устранить здесь неопределенность, перед именем файла ставят двухсимвольный префикс, обозначающий накопитель. Первый символ — это буква, обозначающая накопитель, а второй символ — двоеточие. Например, имя файла b: ADDRESS. DAT указывает на то, что файл с именем ADRESS. DAT находится на накопителе B. Как отмечалось выше, имя текущего накопителя всегда опускается. Например, команда DIR b: *.COM выдает на экран все имена файлов с расширением .COM, находящихся на дискете накопителя B, а команда DIR *.COM выдает на экран имена всех аналогичных файлов, находящихся на дискете в текущем накопителе.

Команда COPY. Чтобы скопировать файл с одной дискеты на другую, пользователю следует ввести команду COPY (от англ. copy — копировать) с двумя именами файлов. Первое имя указывает файл, содержимое которого нужно копировать, а второе имя — файл, в который нужно копировать, но который не обязан существовать до начала копирования. Например, команда COPY CONTRACT. DOC CONTRACT. BAK копирует содержание файла CONTRACT. DOC в файл CONTRACT. BAK на том же накопителе, а команда COPY a: CONTRACT. DOS b: CONTRACT. BAK копирует содержимое файла CONTRACT. DOC с накопителя A в файл CONTRACT. BAK на накопитель B. При этом, если файл CONTRACT. BAK не существовал до начала копирования, то он будет создан, а если существовал, то старая информация в нем уничтожится.

Если во втором имени файла присутствует только префикс b: (как, например, в команде COPY a: CON-

TRACT. DOC b:), то содержимое файла CONTRACT. DOC с накопителя A копируется в файл с тем же именем CONTRACT. DOC на накопитель B. В команде COPY могут использоваться и групповые имена. Например, команда COPY a: * . * b: вызовет копирование всех файлов накопителя A на накопитель B. Если в качестве второго имени файла указать имя prn:, стандартно закрепленное за принтером, то с помощью команды COPY можно распечатывать файлы. Например, команда COPY CONTRACT. DOC prn: копирует файл CONTRACT. DOC на принтер, т. е. печатает этот файл.

Команда REN. Чтобы переименовать файл, пользователю следует ввести команду REN (от англ. rename — переименовать) с двумя именами файла — старым и новым. Например, команда REN CONTRACT. DOC CONTRACT. TXT переименует в файл CONTRACT. DOC, дав ему новое имя CONTRACT. TXT. В качестве старого и нового имен могут использоваться групповые имена. Например, команда REN * .DOC * . TXT переименует все файлы, имеющие расширение .DOC, путем замены этого расширения на новое расширение TXT.

Команда DEL. Чтобы удалить файл, пользователю следует ввести команду DEL (от англ. delete — удалить), указав затем имя удаляемого файла, которое может быть и групповым. Например, команда DEL b: CONTRACT. DOC удалит файл CONTRACT. DOC на накопителе B, а команда DEL a: * . DOC удалит все файлы на накопителе A, имеющие расширение .DOC.

Команда TYPE. Чтобы вывести содержимое файла на экран, пользователю следует ввести команду TYPE (от англ. type — печать), указав затем имя выводимого файла, которое не может быть групповым. Например, команда TYPE b: CONTRACT. DOC выводит на экран содержимое файла CONTRACT. DOC с накопителя B. Команда TYPE предполагает, что в файле содержится информация, закодированная в соответствии со стандартом ASCII; при этом каждый байт хранимой информации обозначает букву, цифру или знак, представленный на экране. В противном случае на экране будет возникать мешанина знаков, сопровождаемая выдачей звуковых сигналов (некото-

рые коды вместо печати на экране генерируют звуковые сигналы). Как правило, файлы, имена которых имеют расширение .TXT, .DOC, .BAT, .PAS. и некоторые другие, могут быть выведены на экран командой TYPE. С другой стороны, заведомо известно, что имена с расширениями .COM, .EXE, OBJ обозначают файлы, не выводимые на экран командой TYPE.

Команда FORMAT. Если пользователь начинает работу с новой дискетой, не содержащей никакой информации, то ему следует после загрузки ОС первым делом выполнить команду FORMAT. Эта команда разметит (отформатирует) должным образом поверхность дискеты. Перед выполнением команды FORMAT следует убедиться, что в накопителе установлена именно та дискета, которая подлежит форматированию, поскольку при форматировании ранее использованных дискет вся информация на них уничтожается. Команда FORMAT отмечает все поврежденные участки дискеты, предотвращая их выделение в будущем под какой-либо файл.

Команды языка БЕЙСИК для работы с файлами. К файловой системе имеет доступ также любая прикладная программа, для чего в языках программирования имеются специальные процедуры. В качестве примера рассмотрим некоторые команды, обеспечивающие работу с файлами в интерпретирующей системе языка БЕЙСИК MSX.

Команды CSAVE и CLOAD. Для сохранения (записи) программ, написанных на MSX БЕЙСИКе, на магнитной ленте используется команда CSAVE.

Формат команды:

CSAVE "имя программы"; <скорость>

В имени программы значащими являются только первые шесть символов. Второй параметр задает скорость записи данных на носитель. Если он равен 1, то скорость равна 1200 бод (1200 бит/с), если 2, то 2400 бод. Он также может быть опущен, тогда этот параметр по умолчанию равен 1.

Команда CSAVE сохраняет программу на магнитной ленте во внутреннем формате, т. е. в том виде, в каком она хранится в памяти компьютера.

Пример:

CSAVE "PROG1"

По этой команде на ленте будет записана программа с именем PROG1 во внутреннем представлении. Для проверки правильности записи программ на ленте можно использовать команду CLOAD?

Перемотав ленту назад и введя команду

```
CLOAD? "PROG1"
```

сверяем (верифицируем) текст записанной на ленте программы с текстом программы в памяти компьютера. При обнаружении несовпадения эта процедура прекращается и на экране появляется сообщение "verify error" — ошибка верификации.

Команда SAVE. Команда SAVE позволяет записать программу как на ленте, так и на гибком магнитном диске. Формат команды:

```
SAVE "имя устройства: имя программы", A
```

В качестве имени устройства может быть указана буква A — идентификатор основного накопителя или буква B — идентификатор дополнительного накопителя, либо идентификатор магнитофона CAS. По умолчанию, когда имя устройства не указано, запись программы осуществляется на дискету основного накопителя.

Без указания в командной строке параметра A программа на дискете записывается во внутреннем коде, а на ленте — в формате ASCII, т. е. в том виде, в котором она вводилась с клавиатуры, без преобразования ее во внутреннее представление. Если ее необходимо сохранить на дискете в формате ASCII, то для этого в командной строке нужно указать параметр A. Например,

```
SAVE "PROG"
SAVE "B:PROG1", A
SAVE "CAS:PROG1"
```

При выполнении первой команды программа записывается на дискете накопителя A во внутреннем формате. По второй команде запись производится на дискету накопителя B в формате ASCII. Третья команда осуществляет запись программы на ленту в этом же формате.

Команда FORMAT. Форматирование дискеты может быть произведено с помощью команды CALL FORMAT или FORMAT. При форматировании разру-

шается (стирается) любое прежнее содержание дискеты, если оно туда записывалось.

Команды FILES, LFILES и DSKF. Наличие тех или иных программ либо файлов на дискете можно определить с помощью команд FILES или LFILES. По команде FILES имена программ и файлов, хранящихся на дискете, выводятся на экран дисплея. А команда LFILES выполняет ту же процедуру для вывода этой информации на принтер.

При записи программ и файлов для хранения необходимо знать наличие свободной памяти на дискете. Для этого предусмотрена команда:

DSKF <номер накопителя>

номер накопителя может быть 0, 1, и 2: 0 — соответствует текущему дискетному накопителю; 1 — накопителю A; 2 — накопителю B.

Команда KILL. Для удаления программы (файла) с дискеты вводится команда

KILL <имя программы>

Команды CLOAD и LOAD. Программы и файлы, сохраненные на ленте и дискете, загружаются в память компьютера с помощью команд CLOAD и LOAD. Формат команды CLOAD:

CLOAD "имя программы" <скорость>

Эта команда позволяет загрузить программу, сохраненную на магнитной ленте во внутреннем формате. Загрузка выполняется на этой же скорости, на которой программа была записана на ленту.

По команде без имени будет загружена первая программа, записанная на ленте. Для загрузки требуемой программы в команде должно быть указано имя программы в виде, в котором оно было представлено при записи программы.

Оператор LOAD производит загрузку программы, сохраненной на ленте в формате ASCII либо на дискете в этом же формате, либо во внутреннем формате, т. е. в том формате, в котором она была записана на дискете. Команду можно использовать с параметром R(RUN), что обеспечивает автоматический запуск программы ее загрузки в память компьютера.

Например,

```
LOAD "CAS:PROG1", R
LOAD "CAS:"
```

```
LOAD "PROG1"
LOAD "B:PROG1", R
```

По первой команде производится загрузка программы PROG1 с ленты, а затем осуществляется ее запуск. По второй команде загружается первая, найденная на ленте программа. По третьей команде программа PROG1 загружается с накопителя A, по четвертой — с накопителя B с последующим запуском.

Команда RUN. Загрузку и запуск программы можно также производить с помощью команды RUN. В ней должны быть указаны имя устройства и имя программы. Она похожа на команду LOAD с параметром A, за тем исключением, что загружает программу только в формате ASCII и закрывает все открытые файлы. В обоих случаях перед загрузкой программы автоматически выполняется команда NEW, очищающая память от ранее введенных программ. Например, по команде

```
RUN "PROG1"
```

происходит загрузка и запуск программы PROG1, сохраненной на дискете в формате ASCII.

Команды MERGE и RENUM. Для объединения текста программы, сохраненной ранее в формате ASCII, с текстом программы, находящейся в памяти компьютера, используется команда MERGE. Наиболее целесообразно употреблять эту команду для работы с библиотеками подпрограмм на ленте или дискете.

Предположим, что программа, находящаяся в памяти, занимает строки с 10-й до 1200-й. Для добавления подпрограммы, названной для сохранения, например ROUT1, и занимающей строки с 10-й до 200-й, она должна быть загружена в память и перенумерована с помощью команды RENUM. При этом начальная строка подпрограммы должна начинаться с номера большего, чем имеет последняя строка основной программы. После этого подпрограмма записывается на ленту или дискету, но уже с новым именем, например ROUT0. Загрузив повторно основную программу и введя команду

```
MERGE "ROUT 0"
```

произведем присоединение к тексту основной программы текста подпрограммы ROUT0.

Команда OPEN. Перед использованием файла его необходимо объявить открытым. Для этого используется команда OPEN, параметрами которой являются: имя устройства (CAS:, A: B:), имя файла, направление потока данных и номер, присваиваемый файлу для операций передачи данных. Например:

```
OPEN "CAS:ADDRESS" FOR INPUT AS #1
OPEN "ADDRESS" FOR OUTPUT AS #1
OPEN "B:ADDRESS" FOR APPEND AS # 2
```

По первой команде осуществляется ввод файла, хранящегося на ленте; по второй команде создается файл на дискете текущего накопителя; по третьей команде файл добавляется к уже существующему файлу на дискете накопителя B.

Команда MAXFILES. Максимальное количество файлов, которые могут быть открыты одновременно, устанавливается командой:

MAXFILES=<количество>

При включении компьютера этот параметр равен 1, что соответствует команде MAXFILES-1. Таким образом, если нужно открыть один файл, то использование этого оператора необязательно. Максимальное количество одновременно открытых файлов равно 15.

Операторы PRINT# и INPUT # . Для записи данных в открытый файл используется оператор PRINT в формате:

PRINT <номер>, <данные>

Данные могут быть числового или символьного типа. Использование расширителя USING позволяет определить собственный формат данных.

Программа 8.1 создает на дискете A: файл TESTF и записывает в него числа от 0 до 10.

Программа 8.1

```
10 OPEN "TESTF" FOR OUTPUT AS#1
20 FOR I=0 TO 10
30 PRINT#1,I
40 NEXT
50 CLOSE
```

Даные из созданного файла могут быть использованы для организации массива переменных в памяти и его вывода на экран.

Программа 8.2

```
10 OPEN "TESTF" FOR INPUT AS#1
20 DIM X(10)
30 FOR I=0 TO 10
40 INPUT#1,X(I)
60 NEXT
70 CLOSE
```

Оператор INPUT# (программа 8.2) используется для ввода данных из файла. Команда CLOSE — закрытие ранее открытого файла осуществляется посредством команды

CLOSE <номер>

Если номер файла в команде CLOSE отсутствует, то происходит закрытие всех открытых файлов. Могут быть указаны несколько номеров закрываемых файлов:

```
100 CLOSE #1,#2
```

Номер закрытого файла может использоваться для открытия другого файла; этот прием особенно удобен, когда в программе необходим только один открытый файл (MAXFILES-1).

Команды NAME и COPY. Для изменения имени файлов на дискете используется команда NAME. Формат команды:

NAME <старое имя> AS <новое имя>

Новое имя файла может быть любым, отличным от существующих на этой дискете. Копирование файлов на дисках осуществляется с помощью команды COPY, имеющей следующий формат:

COPY <имя файла-источника> TO <имя файла-приемника>

Пример:

COPY "TXT1" TO"B:TXT2"

Здесь файл TXT1 копируется с дискеты текущего накопителя в файл TXT2 накопителя B.

Рассмотренные команды не исчерпывают всего набора команд, используемых в интерпретирующей си-

стеме языка БЕЙСИК MSX для работы с файлами. Существуют другие команды, предоставляющие пользователю ряд дополнительных возможностей при работе с файлами.

8.4. ПОДДЕРЖКА ЛОКАЛЬНОЙ СЕТИ

Ранее описывалась локальная сеть, позволяющая производить обмен информацией между компьютерами (см. гл. 1). Теперь рассмотрим программное обеспечение для такого обмена.

Методы доступа. Как правило, для обмена информацией между абонентами локальной вычислительной сети (ЛВС) используется общий канал. Доступ к общему каналу абонентов может быть организован различными методами. Выделяют случайный, детерминированный и комбинированный методы доступа.

В ЛВС со случайным доступом все абоненты равноправны и могут выходить на передачу в любое время. Такая свобода выхода абонента в канал может приводить к конфликтным ситуациям, когда несколько абонентов обращаются к общему каналу. Этот недостаток особенно проявлялся при увеличении интенсивности потока информации в канале. Разработан ряд практических способов и алгоритмов минимизации числа конфликтных ситуаций, что позволяет использовать ЛВС с таким методом доступа.

От этого недостатка свободны сети с детерминированным доступом, где общий канал между абонентами распределяется в определенные моменты времени. Это распределение может быть основано на методе временного разделения работы канала на отдельные интервалы времени, приписываемые отдельным абонентам. В каждый временной интервал канал предоставляется только паре обменивающихся информацией абонентов.

К детерминированному методу доступа также относится широко распространенный метод последовательных передач, получивший название «эстафетного» метода. В этом случае канал предоставляется только тому абоненту, у которого находится «эстафетная палочка». После того как абонент — держатель «эстафетной палочки» посылает сообщение, «эстафетная

палочка» передается следующему абоненту. Таким образом, «эстафетная палочка» ходит по кругу, предоставляя последовательно канал всем абонентам сети. «Эстафетной палочкой» служит код символа, называемый *маркером*.

В качестве передающей среды (физических каналов связи) в ЛВС используются витые пары телефонных проводов, коаксиальные и волоконно-оптические кабели. В зависимости от этого сети имеют различные скорости передачи данных.

В вычислительной сети связь в действительности устанавливается не между отдельными компьютерами, а между прикладными программами, или, более точно, между прикладными вычислительными процессами, протекающими в компьютерах. При этом под прикладным вычислительным процессом понимается прикладная программа вместе с ее набором данных и выделенными ей ресурсами компьютера.

Программное обеспечение ЛВС. Программное обеспечение, поддерживающее ЛВС, имеет многоуровневую структуру. Каждый программный уровень выполняет определенные функции сети. Он может быть при необходимости модифицирован и тем самым приспособлен к особенностям той или иной локальной сети.

Различают семь уровней программного обеспечения сети:

1) прикладной уровень обеспечивает выполнение прикладных программ и управление терминальными устройствами;

2) представительный уровень обеспечивает представление данных, которыми обмениваются устройства оконечных пользователей в принятой для них форме. Это позволяет использовать в различных комплектах оборудования различные способы представления данных без ущерба для взаимодействия пользователей;

3) сеансный уровень обеспечивает организацию, поддержку и окончание сеансов связи между прикладными процессами;

4) транспортный уровень обеспечивает вышестоящие уровни некоторой совершенной системой транспортировки данных. Средства управления транспортного уровня позволяют сделать для пользователя не-

заметными различия, существующие в реальных системах передачи данных;

5) сетевой уровень обеспечивает необходимый маршрут передачи данных, следующих от одного конечного устройства к другому, предоставляя средства управления потоками данных и коммутацией каналов. Кроме того, этот уровень может включать в себя функции организации связи между двумя отдельными сетями;

6) канальный уровень обеспечивает управление каналом передачи данных, установление, поддержание и разъединение каналов;

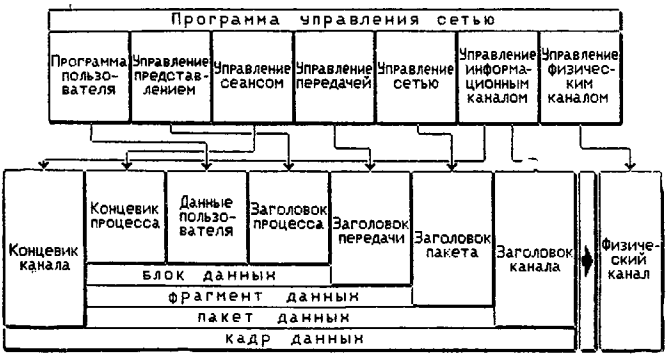


Рис. 8.2. Процедура образования данных в канале

7) физический уровень обеспечивает сопряжение вышестоящих уровней с передающей средой путем предоставления средств подключения (отключения) к физическому каналу связи и преобразования сигналов.

Процедура образования данных в канале под действием программ различных уровней (рис. 8.2) состоит в следующем. Сначала программа абонента-отправителя, представляющая прикладной уровень, выдает данные пользователя, которые следует передать по каналу. Для того чтобы устройство абонента-получателя могло различить эти данные, необходимо указать их адрес в заголовке прикладного процесса, который добавляется к данным посредством программы

представительного уровня. Для выявления ошибок в данных после их передачи другому пользователю программой сетевого уровня в конце к данным добавляется концевик процесса, содержащий необходимую информацию для проверки передачи данных. Программа пользователя, программы управления представлением и сеансом связи образуют прикладной процесс, в результате которого образуется массив данных, называемый блоком данных.

В ЛВС прикладные процессы абонента-отправителя и абонента-получателя находятся в различных компьютерах и взаимодействуют между собой через каналы связи. Поэтому каждому блоку данных нужно придать адреса исходного и конечного процессов, а также идентификатор и тип передаваемого массива данных. Эти сведения содержатся в заголовке передачи, который формируется программой транспортного уровня. Блок данных с заголовком передачи образуют *фрагмент данных.*

Каждый прикладной процесс может быть связан с другими прикладными процессами, выполняемыми в различных компьютерах сети. В результате возникает необходимость задания маршрута передачи. Сведения о маршруте указываются в заголовке пакета, который добавляется к фрагменту данных программой сетевого уровня. Фрагмент данных с заголовком пакета образуют *пакет данных.*

Пакет данных передается по информационному каналу. Информация, необходимая для управления каналом, содержится в заголовке канала, который добавляется к пакету данных программой канального уровня. Этой же программой формируется концевик канала, содержащий информацию, необходимую для проверки правильности приема пакета данных. Пакет данных с заголовком и концевиком канала образуют *кадр данных.* Для отделения кадров друг от друга, идущих в информационном канале непрерывным потоком, используются специальные символы, называемые *флажками.*

Последний уровень программного обеспечения сети представляет программа управления физическим каналом, обеспечивающая необходимую коммутацию и поддержание связи во время передачи.

Таким образом, под действием управляющих про-

грамм сети осуществляется последовательный процесс упаковки данных пользователя на передающей стороне сети. На приемной стороне сети выполняется обратная операция — распаковка, которая осуществляется под действием программ соответствующего уровня.

Взаимодействие одноименных уровней программного обеспечения сети осуществляется по определенным правилам, зафиксированным в так называемых *протоколах*. Локальная сеть имеет такую же многоуровневую иерархию протоколов, как и взаимодействующие друг с другом программные уровни. Протоколы сети могут быть реализованы программными либо аппаратными средствами.

Команды управления локальными сетями. Для управления локальными сетями используются системы команд. Рассмотрим в качестве примера наиболее часто употребляемые команды ЛВС «ЯМАХА MSX2». Данная сеть разработана для связи компьютеров в учебном классе. Она позволяет соединить рабочее место преподавателя (РМП) с 15 рабочими местами учащихся (РМУ), давая при этом возможность обмена программами, данными и сообщениями как между преподавателем и учеником, так и между учениками.

Для начала работы с сетью используется команда NETINIT, если она не была запущена при включении компьютеров. Для отключения сети применяется команда NETEND. Ввод этой команды бывает необходим для работы с прикладной программой, которая не может быть вызвана при работающей сети.

Разрешение на передачу с РМУ осуществляется с помощью команды ENACOM. Формат команды:

CALL ENACOM ([<номер РМУ>]).*

Пример:

CALL ENACOM (12). Эта команда дает разрешение на передачу в сеть РМУ N12. Если номер РМУ в команде не указан, то передача в сеть разрешается всем РМУ.

Для запрета передачи РМУ используется команда DISCOM с форматом

CALL DISCOM ([<номер РМУ>])

Действие параметра в этой команде аналогично

* Параметры в квадратных скобках в команде могут отсутствовать.

команде ENACOM. Передача сообщений РМП в РМУ осуществляется посредством команды MESSAGE. Формат команды

CALL MESSAGE (<сообщение> [,[<номер РМУ>]])

Пример:

CALL MESSAGE («Подготовьтесь к работе!», 10)

По данной команде на экран РМУ N10 будет выдано сообщение «Подготовьтесь к работе!». Длина сообщения ограничивается 36 символами. Если в команде номер РМУ не указан, то сообщение передается всем РМУ.

Для пересылки БЕЙСИК-программы с РМП в РМУ используется команда SEND с форматом

CALL SEND [([[<имя файла>] [,[<номер РМУ>]])]

Пример:

CALL SEND ("A:TEST.BAS", 0)

По этой команде программа TEST. BAS считывается с дискового накопителя A: и передается в память всех РМУ. При этом старая БЕЙСИК-программа стирается.

Пример:

CALL SEND (, 10)

Эта команда передает БЕЙСИК-программу, находящуюся в памяти РМП, в память РМУ N10.

Команда SNDRUN позволяет переслать БЕЙСИК-программу в РМУ и запустить ее. Формат команды SNDRUN и действие параметров аналогичны команде SEND.

Прием БЕЙСИК-программы с РМУ в РМП осуществляется с помощью команды RECEIVE. Формат команды

CALL RECEIVE ([<имя файла>], <номер РМУ>)

Пример:

CALL RECEIVE (B:TEST.BAS, 14)

По этой команде РМП принимает БЕЙСИК-программу с РМУ № 14 и записывает ее на диск накопителя B: в файл с именем TEST. BAS.

Пример:

CALL RECEIVE (, 1)

В этом случае БЕЙСИК-программа принимается

из РМУ № 1 и записывается в память РМП, стирая при этом старую программу.

Для запуска БЕЙСИК-программы РМУ используется команда RUN с форматом:

CALL RUN[[([[<номер РМУ>][,[номер строки<]]]])]

Пример:

CALL RUN (1,100)

Эта команда запускает программу РМУ № 1 с 100-й строки. Если программа была уже запущена, то она останавливается и ее выполнение начинается с заданной строки. При отсутствии в команде параметров программа запускается со своей первой строки.

Рассмотренные выше команды относятся к командам преподавателя, т. е. к командам, вводимым только с его рабочего места. Однако в системе команд также имеются команды учащегося и команды учащегося/преподавателя. К последним принадлежат, например, команда BSEND и BRECEIVE. С помощью этих команд осуществляются передача и прием программ в машинных кодах и изображений с экрана дисплея. При этом ввод команд может производиться как с рабочего места преподавателя, так и с рабочего места учащегося.

8.5. ТЕСТ-ДИАГНОСТИЧЕСКИЕ ПРОГРАММЫ

Проверка состояния (правильности функционирования) компьютера и обнаружение неисправных устройств в нем осуществляются с помощью набора тест-диагностических программ, входящих в состав базового программного обеспечения.

Данные программы позволяют проверить правильность функционирования всей системы, отдельных ее компонентов, определить неверно выполняемые операции и неправильные компоненты.

Существуют тест-программы команд, прерываний, памяти, арифметики и периферийных устройств ввода-вывода.

Тест-программа команд является основной. С ее помощью проверяется правильность выполнения команд из базового подмножества системы команд компьютера. В процессе выполнения тест-программы проверяется правильность исполнения каждой коман-

ды из этого подмножества. Если обнаружено, что команда выполняется неправильно, осуществляется останов и на дисплей выдается необходимая информация.

Тест-программа памяти предназначена для проверки оперативной памяти. Она задает определенный режим работы памяти и проверяет правильность ее функционирования с помощью специально сформатированных кодовых массивов. В процессе выполнения тест-программы проверяется правильность выбора адресов ячеек памяти и правильность записи-чтения информации при упорядоченном перемещении в ячейках всего массива памяти нулей и единиц. Если при проверке памяти обнаруживается ошибка, выдается соответствующее сообщение.

Тест-программа прерываний используется для проверки правильности аппаратных средств компьютера, с помощью которых выполняются внутренние и внешние прерывания программы. Эта тест-программа проверяет правильность выполнения последовательности операций и заполнения служебных ячеек памяти при различных видах прерываний программы (командные прерывания, прерывания от внешних устройств и т. д.).

Тест-программа арифметики для проверки дополнительных групп команд, предназначенных для выполнения арифметических операций, в том числе умножения и деления чисел. С помощью этой программы проверяется выполнение команд с различными данными и способами адресации путем сравнения ожидаемых результатов с полученными при тестировании.

Тест-программы внешних устройств ввода-вывода предназначены для проверки клавиатуры, печатающих устройств, внешних запоминающих устройств и другого периферийного оборудования.

Все тест-программы исполняются отдельно или под управлением специальной *тест-мониторной операционной системы* (ТМОС), объединяющей тестовые программные средства в единую систему. Она может применяться при проверке и поиске неисправностей компьютера в процессе наладки, испытаний, эксплуатации и ремонта.

ТМОС выполняет следующие функции:

1) организацию справочника из тестовых программ и документов;

2) загрузку и запуск тестовых программ;

3) корректировку тестовых программ;

4) создание и корректировку тестовых файлов;

5) формирование последовательности команд, задающих последовательность выполнения отдельных тест-программ и число повторений для каждой из них;

6) копирование ТМОС с одного носителя (дискеты, диска, магнитной ленты) на другой носитель;

7) генерацию версии ТМОС в соответствии с составом аппаратной части компьютерной системы.

Проверка компьютера или компьютерного комплекса по тест-программам, вводимым в ОП с внешнего запоминающего устройства, является сравнительно длительным и трудоемким процессом. Поэтому для проверки основных модулей системы, составляющих ее функциональное ядро, часто вводится дополнительный, менее глубокий, но более оперативный уровень тестирования — тестирование с помощью так называемых встроенных программ, размещенных в ПЗУ и автоматически выполняемых при включении питания или повторном запуске системы.

В случае возникновения неисправностей, не локализуемых встроенными тестовыми программами, необходимо обратиться к ТМОС и осуществить более глубокую и детальную проверку функционирования и диагностику неисправностей компьютера или компьютерного комплекса.

Глава 9

ПРИКЛАДНОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

Одним из возможных путей повышения производительности труда при обработке информации с помощью компьютера является создание специальным образом организованных программных комплексов, называемых пакетами прикладных программ (ППП), обеспечивающих потребности большого числа пользователей.

9.1. ПАКЕТЫ ПРИКЛАДНЫХ ПРОГРАММ

Особенности организации ППП. Можно выделить следующие особенности организации ППП.

1. Пакет ориентирован на определенный класс задач и конкретную предметную область.

2. Каждый пакет обладает соответствующим набором возможностей по методам обработки и формам представления данных.

3. Для выбора варианта обработки и представления данных пакет имеет средства настройки. Исключение составляют пакеты, имеющие простую структуру в виде библиотеки стандартных программ БСП. Они состоят из совокупности программ, составленных на одном из языков программирования и удовлетворяющих определенным единым требованиям к структуре организации их входов и выходов. Стандартные программы имеют единую форму обращения, что обеспечивает возможность автоматизации их включения в вызывающую программу и настройки их параметров.

4. Задание параметров и управляющей информации для настройки пакета осуществляется с помощью входного языка. Пакет имеет достаточно простой входной язык и может эксплуатироваться пользователем без участия программиста. Как правило, это относится к пакетам, ориентированным на решение экономических, инженерных, научных и специальных задач. Такие пакеты часто называют пакетами общего назначения.

Классификация ППП. Пакеты прикладных программ принято классифицировать по реализуемым функциям, по типу операционной системы, под управлением которой работает пакет, и по способу управления пакетом.

По реализуемым функциям ППП делятся на: расширяющие возможности операционных систем; обеспечивающие решение задач пользователя.

Классификация по типу операционной системы определяет возможность применения пакета в конкретной операционной обстановке. Как правило, пакет, работающий под управлением одной ОС, не может работать под управлением другой.

По способу управления ППП делят на два

класса: пакеты простой структуры и сложной структуры. *Пакет простой структуры* представляет собой набор программных модулей, обеспечивающих решение различных задач из предметной области, на которую ориентирован пакет. Обращение к модулям может осуществляться из прикладной программы пользователя на том же самом языке, который использован для написания программ пользователя. При этом программа пользователя и модули должны быть размещены в оперативной памяти. В некоторых случаях это ограничивает размер решаемой задачи. Такие пакеты целесообразно использовать для расширения системных и личных библиотек. Их называют пакетами, расширяющими библиотеки.

В *пакетах сложной структуры* функции управления передаются специальным модулям, образующим управляющую программу. Обращение к ним осуществляется с помощью функций пользователя, для формирования которых используется входной язык пакета. Функции пользователя обеспечивают задание требуемой последовательности обращения к обрабатывающим модулям. Эта последовательность может быть реализована в виде программы на входном языке пакета.

По способу организации управления пакетом различают три класса: 1) пакеты, работающие в режиме пакетной обработки, 2) пакеты, работающие в диалоговом режиме; 3) пакеты, обеспечивающие по выбору любой из указанных режимов.

Режим пакетной обработки предусматривает предварительную подготовку на перфокартах, перфоленте или магнитной ленте набора (пакета) заданий на выполнение одной или нескольких связанных между собой программ. При выполнении каждого задания проверяется правильность задания, осуществляется выделение заданию необходимых ресурсов (памяти, устройств ввода-вывода), определяется программа, которая должна быть выполнена в данный момент, контролируется ход ее выполнения. Для описания реально используемых программ и устройств ввода-вывода служит язык управления заданиями.

В диалоговом режиме ввод управляющих команд на выполнение заданий осуществляется в процессе взаимодействия пользователя с вычислительной си-

стемой. Команды диалоговой системы носят характер директив, в которых широко используются правила умолчания и подстановки. Это дает возможность значительно сократить количество набираемой на клавиатуре информации.

Основным принципом построения пакетов прикладных программ является модульность структуры. Один из возможных вариантов организации пакетов сложной структуры представлен на рис. 9.1.

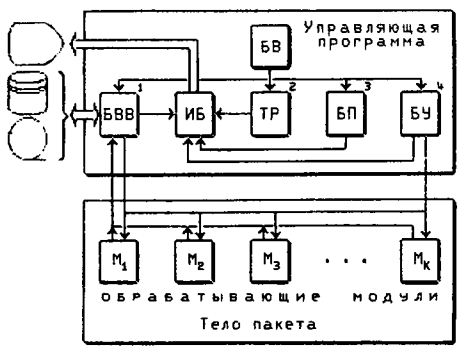


Рис. 9.1. Организация пакета сложной структуры

Пакет включает в себя управляющую программу (УП), реализующую функции настройки пакета и управления вычислительным процессом, и совокупность обрабатывающих модулей — тело пакета (ТП).

Ведущий блок (ВБ) предназначен для управления ходом работы программ (блоков) пакета. Он определяет порядок их работы, формирует данные, необходимые операционной системе для выполнения того или иного этапа работы пакета.

Блок ввода-вывода (ББВ) выполняет функции обмена с внешними устройствами: ввод программы на входном языке пакета, ввод исходных данных, вывод промежуточных и конечных результатов обработки на заданные устройства внешней памяти.

Трансляцию входной программы во внутреннее представление производит транслятор (ТР). Результаты трансляции могут иметь различную форму. Это

зависит от уровня входного языка и используемых методов и средств трансляции. Если для управления пакетом используется язык высокого уровня, то результатом трансляции может быть управляющая информация, используемая для реализации заданной последовательности обращений к обрабатывающим модулям.

Блок памяти (БП) обеспечивает распределение основной памяти компьютера для решения задачи. Он также может осуществлять функции обслуживания внешней памяти, например библиотеки пакета.

Обращение к телу пакета осуществляется посредством блока управления (БУ). В соответствии с управляющей информацией, поступающей от транслятора, он реализует заданную последовательность обращений к обрабатывающим модулям. Этот блок также может осуществлять функции загрузчика модулей в основную память в тех случаях, когда структура пакета предусматривает замещение одного обрабатывающего модуля другим.

Выдача пользователю сообщений, отражающих ход выполнения этапов работы пакета, производится с помощью информационного блока (ИБ).

Кроме рассмотренных блоков пакеты могут содержать наборы сервисных блоков, обеспечивающих диагностику ошибок, возникающих при невозможности корректного выполнения системных (управляющих) функций пакета, и ошибок, обнаруженных в тексте программы на входном языке пакета. Для исправления этих ошибок пакет включает в себя блок, предоставляющий пользователю средства редактирования.

Основными командами входного языка пакета являются команды ввода-вывода, сохранения данных во временном файле, записи данных из временного файла в рабочую область пакета и условного продолжения программы по коду возврата модуля. Смысл последней команды состоит в том, что если код возврата обрабатывающего модуля, непосредственно предшествующего команде продолжения, меньше или равен коду, указанному в команде продолжения, то выполнение программы продолжается, иначе программа прерывается и на устройство вывода выдается соответствующее сообщение.

Пакеты, в которых предусмотрен диалоговый ре-

жим работы, включают в себя блоки, обеспечивающие ведение диалога. Диалог может использоваться как для инициализации пакета, так и для его сопровождения. Инициализация заключается в формировании и передаче пакету программы на входном языке для ее последующей трансляции и внесения исправлений. Сопровождение обеспечивает получение нужных промежуточных результатов решения после завершения заданных этапов работы пакета. Эта информация дает возможность пользователю оперативно изменять стратегию управления пакетом путем введения соответствующих инструкций. Режим сопровождения расширяет возможности оперативного управления ходом решения и требует постоянного взаимодействия пользователя с пакетом.

9.2. ТЕКСТОВЫЕ РЕДАКТОРЫ

В настоящее время программное обеспечение включает в себя большое количество программ сервисного назначения, облегчающих обработку информации. К этим сервисным средствам относятся и различного типа текстовые редакторы.

Текстовый редактор представляет собой программу, с помощью которой пользователь может создавать и изменять программы, тексты, формулы, таблицы, диаграммы. Редактирование осуществляется в процессе диалога пользователя с компьютером, во время которого решаются следующие задачи:

1) выбор части документа для его отображения и редактирования;

2) создание визуального представления объекта редактирования;

3) задание и выполнение операций, изменяющих документ;

4) изменение визуального представления объекта редактирования в соответствии с выполненными операциями.

Поиск нужной части документа для его отображения и редактирования производится путем просмотра всего документа. Для этого могут быть, например, использованы операции построчного листания документа или поиска по заданному образцу. После определения местоположения нужной области с помощью операции

фильтрации осуществляется выбор объекта для редактирования. Этим объектом может быть страница, абзац, оператор, группа операторов и т. д. Затем посредством операции *форматирования* создается визуальное представление редактируемого объекта, которое отображается на экране дисплея.

Создание и изменение документа в процессе редактирования производятся под действием таких операций, как «вставить», «уничтожить», «заменить», «передвинуть» и «копировать». Функции редактора зависят от его целевого назначения. Так, например, редактор, ориентированный на работу с текстовой информацией, может оперировать с отдельными символами, словами, строками, предложениями и абзацами; программно-ориентировочный редактор — с идентификаторами, ключевыми словами, метками и операторами.

Редакторы, в которых представление документа осуществляется на экране монитора и предусмотрена возможность выполнения операций редактирования над его частями без ограничения справа и снизу, называют экранными редакторами. В более развитых экранных редакторах пользователь с помощью передвигаемого по экрану окна или нескольких таких «окон» может одновременно визуализировать различные части документа и производить их редактирование.

Общение пользователя с редактором. Оно осуществляется посредством диалоговых языков редактирования. Существуют разные способы общения.

Наиболее распространенный способ — это набор и ввод командных строк редактора с клавиатуры. Здесь предполагается, что пользователь должен знать формат всех команд редактора. В противном случае он должен обращаться к руководству по описанию команд или запрашивать помощь у компьютера, если такая помощь в программе редактора реализована. Очевидно, что такой способ общения не эффективен, так как требует больших затрат времени.

В настоящее время широкое распространение получил способ общения, в котором определенные действия редактора производятся с помощью функциональных клавиш. Для расширения возможностей клавиатуры часто используются комби-

нации функциональных клавиш с другими. Комбинации клавиш предполагают их одновременное нажатие.

При использовании стандартной клавиатуры преимущественно применяют способ общения, основанный на меню. Меню представляет собой набор текстовых строк или образов, выраженных графическими символами, объектов или операций. Когда имеется много возможных действий и несколько путей выполнения каждого из них, меню имеет иерархическую структуру, в которой ряд вспомогательных меню подчинен головному. Для выбора нужной команды или функции из меню часто используется клавишный координатный манипулятор. На каждой клавише манипулятора изображена соответствующая стрелка, направленная вверх, вниз, влево и вправо. С помощью этих клавиш путем установки курсора на позицию с номером меню осуществляется выбор того или иного действия редактора.

Экранный редактор. Он ориентирован на работу с текстовой информацией и может оперировать с отдельными символами, словами, строками, предложениями и образцами. В функции этих редакторов могут входить:

1) построчное редактирование, позволяющее изменение только одной строки на экране телевизионного монитора;

2) поэтапное редактирование, представляющее возможность изменения сразу нескольких или всех строк на экране путем передвижения курсора в любую позицию экрана;

3) вывод на экран первой и последующих строк файла; перемещение по файлу в обоих направлениях, сверху вниз и снизу вверх;

4) управление несколькими логическими экранами — окнами, позволяющими совмещать на одном экране тексты различных файлов;

5) поиск заданного текста путем сканирования (построчного просмотра) редактируемого файла от его начала к концу и наоборот до совпадения с заданным текстом — последовательностью символов;

6) поэкранное листание файла в прямом и обратном направлениях, обеспечивающее просмотр и поиск листа в файле, где необходимо произвести редактирование;

7) удаление строк в редактируемом файле, восстановление ранее удаленных строк, вставка строк из другого файла, копирование строк из одного места файла в другое, перенумерация, сортировки строк и другие функции, предоставляющие пользователю возможность быстро вносить необходимые изменения и дополнения в тексты редактируемых файлов;

8) повторное выполнение последней команды редактора, позволяющее многократно повторять последнюю введенную команду;

9) выполнение макрокоманд редактирования, расширяющее возможности редактора за счет выполнения заранее введенной последовательности команд редактирования в виде макрокоманды (путем вызова макрокоманды можно многократно повторять запрограммированные действия);

10) завершение работы редактора без сохранения и с сохранением отредактированного файла на диске (при сохранении отредактированного файла некоторые редакторы удаляют предыдущую версию, заменяя ее новой, другие — сохраняют ее на диске, но под другим именем);

11) получение справочной информации о командах редактора, предоставляющее пользователю возможность при необходимости получить помощь (в виде подсказки) по всем или отдельным командам редактора.

Ознакомимся с некоторыми возможностями текстового экранного редактора в интерпретирующей системе БЕЙСИК MSX. Для этого произведем редактирование БЕЙСИК-программы вычисления суммы первых N членов натурального ряда, введенной в память компьютера с ошибками.

После ввода команды LIST на экране отобразится текст программы 9.1.

Программа 9.1

```
10 INP T"N";N
20 S=0
30 FOR I=1 TO M
40 S=SS+I
60 PRINT"S=";S
70 END
```

Нетрудно заметить в программе ошибки. В слове INPUT (строка 10) пропущена буква U, в строке 30 вместо буквы N введена буква M, в строке 40 ошибочно введено имя переменной SS и, наконец, в программе опущен оператор конца цикла NEXT.

Редактирование строк программы осуществляется в следующем порядке. Сначала курсор, указывающий на место следующего вводимого символа, устанавливается в нужную позицию строки, и с помощью команд редактирования, реализуемых нажатием соответствующих клавиш, устраняются ошибки. Затем, не производя перемещение курсора из редактируемой строки, нажимается клавиша ВК. При этом правильная строка замещает в памяти компьютера строку с ошибками.

Для устранения ошибки в строке 10 необходимо курсор установить в позицию буквы T, нажать клавишу «INS» (от англ. insert — вставить) и ввести пропущенную букву U.

Замена в строке 30 буквы M на букву N производится установкой курсора в позицию буквы M и вводом на это место буквы N.

Для удаления лишнего символа в имени переменной SS (срока 40) нужно курсор установить в позицию первой буквы и нажать на клавишу «DEL» (от англ. delet — удалить). При этом произойдет удаление символа и сдвиг оставшейся части строки на одну позицию влево.

Ввод пропущенной строки (оператор NEXT) осуществляется обычным способом. Набирается номер строки (например, 50), текст строки и нажимается клавиша «ВК».

После редактирования программы ее вновь можно распечатать на экране с помощью команды LIST.

9.3. ГРАФИЧЕСКИЙ РЕДАКТОР

В этом параграфе рассмотрим специальные программы — *графические редакторы*, позволяющие формировать изображение при помощи курсора. Качество изображений, построенных с помощью графических редакторов, в основном зависит от художественных (а не программистских) способностей пользователя. При этом экран дисплея напоминает обычный

холст художника, на котором путем манипулирования различными клавишами и строится рисунок (точнее «игра» клавишами приводит к изменению (редактированию) данных видеопамяти).

В настоящее время существует много различных вариантов графических редакторов. Возможности редакторов в основном зависят от типа компьютера (от объема его оперативной памяти, от разрешающих способностей дисплея и т. д.). Поэтому наряду с небольшими программами имеются целые системы, позволяющие комбинировать видеоизображение, полученное с видеомагнитофона и видеоканала телевизора, с компьютерной графикой; вводить в компьютер фотографические изображения и над ними совершать различные преобразования, сохранять их в памяти или выводить на бумагу.

Рассмотрим возможности среднего графического редактора, например программы «PAINTER», разработанной для персонального компьютера «ЯМАХА MSX2». Персональный компьютер «ЯМАХА MSX2» обладает большим объемом видеопамяти (128 Кбайт). Указанного объема памяти хватает на создание «холста» из 432 * 576 точек с палитрой из 14 цветов (каждый цвет палитры может принимать один из 512 цветовых оттенков). Размер холста в 432 * 576 точек при выводе на печать соответствует размеру обычного листа писчей бумаги (188 * 203 мм). Процесс создания рисунка в графическом редакторе представляет собой диалог пользователя с компьютером. Диалог ведется под управлением «выпадающих» и позиционных меню (рис. 9.2). «Выпадающие» меню несут справочную информацию о режимах работы редактора (рис. 9.3). Позиционные же меню (рис. 9.4) содержат наборы условных символов (пиктограмм).

В основе работы любого графического редактора лежит принцип использования курсора, которым можно управлять либо клавишами, либо с помощью специального манипулятора «Мышь» (этот манипулятор получил название «Мышь» исключительно из-за его внешнего сходства). Курсор можно заставить не только перемещаться по экрану, но и менять свою форму. Перемещаемый курсор, оставляющий на экране дисплея след, соответствующий форме кисти, является основным рисующим элементом графического редакто-

Рис. 9.2. Экран графического редактора

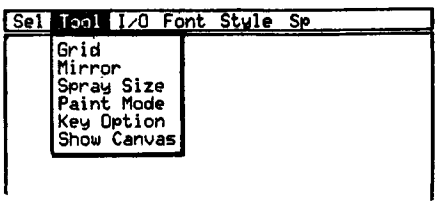


Рис. 9.3. Выпадающее меню

Рис. 9.4. Позиционное меню:

а — карта кистей; *б* — карта шаблонов; *в* — карта инструментов; *г* — карта координат; *д* — карта цветов

холст художника, на котором путем манипулирования различными клавишами и строится рисунок (точнее «игра» клавишами приводит к изменению (редактированию) данных видеопамяти).

В настоящее время существует много различных вариантов графических редакторов. Возможности редакторов в основном зависят от типа компьютера (от объема его оперативной памяти, от разрешающих способностей дисплея и т. д.). Поэтому наряду с небольшими программами имеются целые системы, позволяющие комбинировать видеоизображение, полученное с видеомагнитофона и видеоканала телевизора, с компьютерной графикой; вводить в компьютер фотографические изображения и над ними совершать различные преобразования, сохранять их в памяти или выводить на бумагу.

Рассмотрим возможности среднего графического редактора, например программы «PAINTER», разработанной для персонального компьютера «ЯМАХА MSX2». Персональный компьютер «ЯМАХА MSX2» обладает большим объемом видеопамяти (128 Кбайт). Указанного объема памяти хватает на создание «холста» из 432 * 576 точек с палитрой из 14 цветов (каждый цвет палитры может принимать один из 512 цветовых оттенков). Размер холста в 432 * 576 точек при выводе на печать соответствует размеру обычного листа писчей бумаги (188 * 203 мм). Процесс создания рисунка в графическом редакторе представляет собой диалог пользователя с компьютером. Диалог ведется под управлением «выпадающих» и позиционных меню (рис. 9.2). «Выпадающие» меню несут справочную информацию о режимах работы редактора (рис. 9.3). Позиционные же меню (рис. 9.4) содержат наборы условных символов (пиктограмм).

В основе работы любого графического редактора лежит принцип использования курсора, которым можно управлять либо клавишами, либо с помощью специального манипулятора «Мышь» (этот манипулятор получил название «Мышь» исключительно из-за его внешнего сходства). Курсор можно заставить не только перемещаться по экрану, но и менять свою форму. Перемещаемый курсор, оставляющий на экране дисплея след, соответствующий форме кисти, является основным рисующим элементом графического редакто-

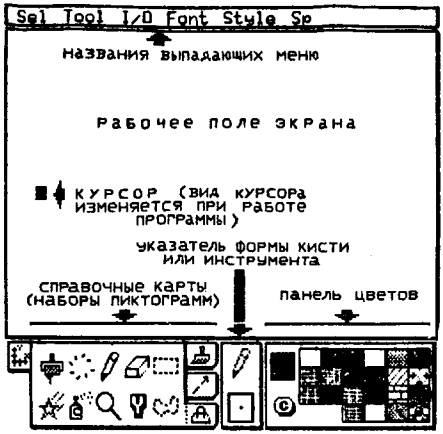


Рис. 9.2. Экран графического редактора

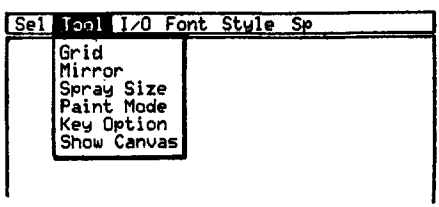


Рис. 9.3. Выпадающее меню

Рис. 9.4. Позиционное меню:

а — карта кистей; *б* — карта шаблонов; *в* — карта инструментов; *г* — карта координат; *д* — карта цветов

ра. Но кисти — не единственный инструмент редактора.

В программе «PAINTER», например, кроме кистей существуют такие инструменты, как шаблонные линейки, валик для наката краски, аэрограф, карандаш, лупа, ластик, шрифтовые трафареты, ножницы и т. д.

Шаблонные линейки в редакторе «умеют растягиваться», а шрифтовые трафареты — «менять» стиль начертания букв (рис. 9.5). Кроме того, в редакторе имеются инструменты, подобных которым нет ни в одной мастерской художника. Это инструменты для рас-

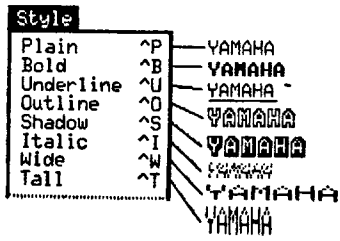


Рис. 9.5. Стили начертания букв

тягивания или сжатия фрагментов рисунков, изображения симметричных фигур, разметки холста сеткой, перемещения фрагментов рисунка и, наконец, инструмент для создания мультфильмов.

Но, пожалуй, самыми важными режимами работы редактора являются режимы записи данных видеопамяти (рисунков) на диск и вывода рисунков на лист бумаги. Первая возможность позволяет нарисованные в редакторе изображения использовать в программах, написанных на одном из языков программирования. Вывод же рисунков на бумагу позволяет использовать редактор в оформительской деятельности художника, например, как это было сделано при разработке всех рисунков данного пособия.

9.4. СИСТЕМЫ УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ

Большое число различных задач связано с хранением значительного объема информации в виде единообразных записей и сравнительно простой их обра-

боткой для извлечения новой информации в удобной для пользователя форме. Предназначенные для этого программные средства обычно объединяются под общим названием *системы управления базами данных* (СУБД), в котором под *базой данных* понимается совокупность взаимосвязанных и хранящихся в удобном для использования виде данных. Наилучшая форма организации данных зависит как от их содержания, так и от того, как эти данные должны использоваться. Например, существует немало СУБД, предназначенных для обработки списков, содержащих разные сведения о перечисленных в этих списках людях. Эти СУБД существенно различаются в зависимости от того, какие сведения о каждом лице имеются в списке (т. е. в зависимости от логической структуры данных), и от того, как осуществляется поиск фамилий того или иного лица.

Вообще, *логической структурой* данных принято называть конкретные логические взаимосвязи между данными, обусловленные содержанием этих данных. Например, логической структурой стипендиальной ведомости студенческой группы является взаимосвязь каждой фамилии студента с величиной его стипендии и взаимосвязь всех перечисленных в ведомости фамилий с номером группы. Совокупность логически взаимосвязанных данных (или, как их еще называют, *полей*) принято называть *логической записью*, а совокупность логических записей данного вида — *файлом* (это то же самое понятие, которое было введено ранее).

Таким образом, каждая строчка стипендиальной ведомости образует отдельную логическую запись, состоящую из двух полей — фамилии и величины стипендии (например, ИВАНОВ 50), а вся ведомость образует файл.

В базе данных может одновременно храниться много различных файлов, поэтому пользователь присваивает каждому файлу свое имя.

Всякая база данных должна храниться в запоминающих устройствах компьютера. Поэтому форма организации данных зависит не только от логической структуры, но и от ее физического представления, т. е. от способа хранения данных в памяти компьютера. Порцию данных, которую СУБД считывает из па-

мяти или записывает в память компьютера как единое целое, принято называть *физической записью.*

Максимальный размер физической записи зависит от технических характеристик запоминающих устройств компьютера. Так, современные запоминающие устройства на магнитных дисках позволяют объединить несколько десятков строчек (т. е. логических записей) стипендиальной ведомости в одну физическую запись, ускорив тем самым считывание из памяти и запись этой ведомости в память компьютера.

Одной из самых важных функций СУБД является поиск логической записи в файле. Поиск производится по ключу, который может состоять из поля, из части поля, из комбинаций полей. Например, при поиске величины стипендии студента по фамилии ИВАНОВ ключом является фамилия ИВАНОВ.

Категории баз данных. По характеру связи между логическими записями, полями и ключами все базы данных делятся на три категории: иерархические, сетевые, реляционные.

Иерархическая база данных. Она называется так потому, что в ней существует упорядочность полей внутри каждой логической записи: одно поле считается главным, а другие поля ему подчинены. Группы логических записей упорядочиваются в определенную последовательность как ступеньки лестницы, и поиск данных может осуществляться прохождением по этой лестнице в соответствии с порядком, определенным последовательностью главных полей.

В качестве примера иерархической базы данных рассмотрим данные о размерах стипендий всех студентов института. Первый уровень в организации логической структуры такой базы может включать таблицу, содержащую названия факультетов, численность студентов этих факультетов, получающих стипендию, и стипендиальный фонд факультетов (рис. 9.6). Название факультета выступает в качестве главного поля. Только осуществив поиск в таблице первого уровня и выбрав нужный факультет (например, машиностроительный), можно получить доступ к таблице второго уровня.

Второй уровень иерархии может включать таблицы, содержащие перечни специализаций каждого фа-

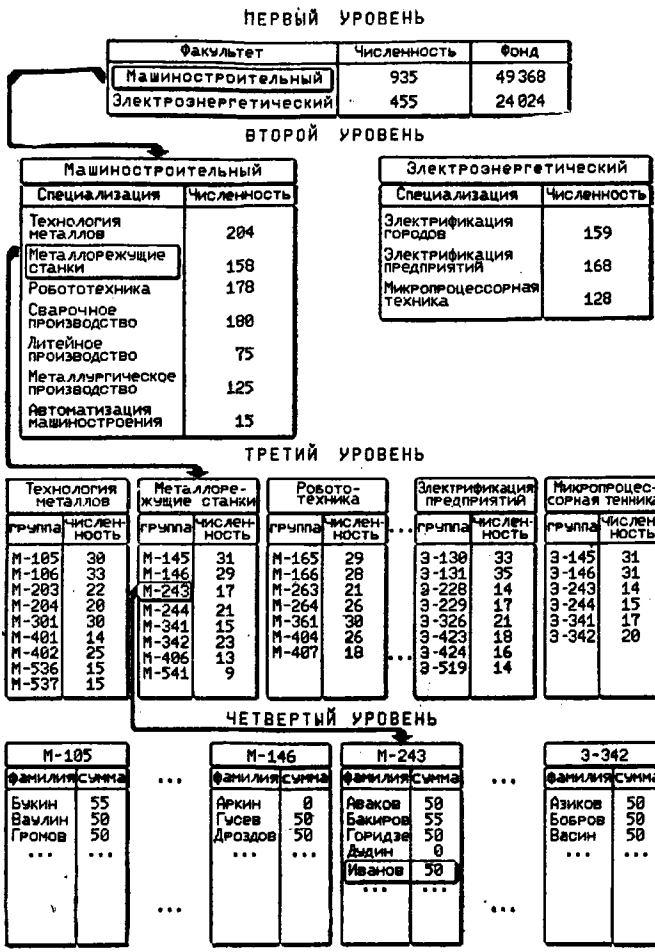


Рис. 9.6. Пример иерархической базы данных

культета. При этом в качестве главного поля выступает название специализации. В таблицах следующих уровней может содержаться информация о номере студенческих групп (главное поле) по каждой специализации, их численности и о размерах стипендий, получаемых студентами каждой группы. Последовательно пройдя через несколько уровней иерархии, можно отыскать нужную информацию, такую, например, как размер стипендии студента по фамилии ИВАНОВ. В результате для многоуровневой иерархической базы данных время ответа на запрос может быть весьма большим.

Сетевая база данных. Она обладает несколько большей гибкостью по сравнению с иерархической, поскольку в сетевой базе данных можно установить связи не только между файлами соседних уровней, но и «перескакивать» через один или несколько уровней. Так, например, в базе данных на рис. 9.6 связь может быть установлена между списком факультетов и таблицей номеров студенческих групп, и тогда станет возможным отыскать номер студенческой группы без предварительного поиска промежуточной таблицы, содержащей перечень специализаций данного факультета.

Реляционная база данных. Эта база данных обладает наиболее гибкой логической структурой. При иерархической, или сетевой, организации базы данных ответы на одни виды запросов занимают значительно меньше времени, чем на другие. Более того, сказать, на какие запросы идет много времени, а на какие мало, можно только после того, как база данных уже создана. Например, в иерархической базе данных, созданной согласно рис. 9.6, наибольшее время займет ответ на запрос о размере стипендии конкретного студента, поскольку поле «Сумма», содержащее размер стипендии, находится в таблице самого последнего уровня. К тому же лишь только после затраты большого труда на создание такой базы данных может выясниться, что «неудобные» для базы запросы, требующие большого времени на ответ, поступают чаще всего. От подобной неприятности избавлена реляционная база данных, потому что все поля в логических записях такой базы равноправны: нет главного поля и подчиненных ему полей. Напри-

мер, если преобразовать изображенную на рис. 9.6 иерархическую базу данных в реляционную, то получится файл, содержащий 935+455=1390 логических записей (по одной записи на каждого студента института).

Каждая из этих записей может содержать, например, следующие поля: факультет, специализация, группа, фамилия, сумма. Так, студенту ИВАНОВУ из группы М-243, согласно рис. 9.6, соответствует следующая логическая запись:

Машиностроительный	Металлорежущие станки	М-243	50

Не следует думать, что не возникает проблем при списке информации в реляционной базе данных. Если 1390 логических записей, подобных этой, не подчиняются никакому заранее установленному в файле порядку, то ответ на запрос о размере стипендии студента ИВАНОВА связан со сплошным просматриванием всех записей файла. С увеличением общего количества записей в файле такая процедура поиска потребует весьма большого времени. Чтобы ускорить процесс поиска, записи в файле можно упорядочить, например, по алфавитному порядку фамилий студентов. Упорядочение записей позволяет ускорить их поиск с помощью так называемой двоичной схемы или двоичного поиска.

Согласно двоичной схеме выбирается запись, находящаяся посередине файла. По фамилии студента в этой записи легко судить, в какой из двух половин файла находится искомая запись. Затем процесс повторяется до тех пор, пока не будет найдена нужная запись.

Пусть, например, в самой середине файла оказалась запись с фамилией ОСОКИН. Сравнивая первые буквы этой фамилии и искомой фамилии ИВАНОВ, легко понять, что искомая запись находится в первой половине файла. Пусть далее записью, которая делит первую половину файла пополам, оказалась запись:

Электроэнергетический	Микропроцессорная техника	Э-314

ЗАЙЦЕВ	0

Тогда искомая запись находится во второй четверти файла. Продолжая далее точно так же деление второй четверти файла пополам, получим 1/8-ю и 1/16-ю части файла и т. д. В конце концов будет найдена нужная запись и дан ответ на запрос о размере стипендии студента ИВАНОВА.

Двоичный поиск осуществляется быстрее, чем поиск просмотром. Пусть файл, в котором осуществляется поиск, включает n логических записей. Если эти записи не упорядочены, то окажется, что среднее арифметическое количество записей, которое нужно просмотреть, чтобы ответить на большое количество запросов, приближается к $g/2$ с ростом числа запросов.

Если записи упорядочены, то при двоичном поиске количество просмотренных при ответе на любой запрос записей не будет превышать числа $\log_2 n$, округленного до ближайшего целого. Поскольку $\log_2 n$ во всяком случае намного меньше $n/2$ при большом n, то максимальное время, необходимое для ответа на запрос при двоичном поиске, значительно меньше среднего времени ответа при поиске в неупорядоченном файле.

Однако упорядоченный файл имеет и недостатки. Во-первых, при большом количестве логических записей пополнение такого файла новыми записями связано с перемещением старых записей, чтобы не нарушить упорядоченность файла в целом. Во-вторых, всякая запись упорядоченного файла может быть найдена только по значению того поля, по которому упорядочены все записи файла. Например, если бы все 1390 записей в примере были упорядочены в файле не по фамилиям, а, скажем, по номерам студенческих групп, то найти размер стипендии студента ИВАНОВА с помощью двоичного поиска, не зная номера группы этого студента, было бы невозможно. Конечно, можно создать несколько дубликатов одного и того же файла, упорядоченных по значению разных полей, но для этого потребуется много дополнительного места в памяти компьютера.

Еще более гибкая, чем двоичная схема, процедура поиска связана с *хешированием* (от англ. hashing — перемешивание).

Согласно этой процедуре каждой логической записи в файле по некоторому правилу ставится в соответствие число, которое аналогично адресу этой записи в памяти компьютера. Например, если пронумеровать все буквы русского алфавита, то отдельные буквы фамилии студента И В А Н О В будут иметь соответственно двухразрядные номера 10, 03, 01, 15, 16, 03. Составленное из этих номеров число 100301151603 порождает адрес логической записи в памяти компьютера. Сложность такого подхода заключается в том, что адреса логических записей будут получаться очень большими (например, адрес записи с фамилией ИВАНОВ получился больше 100 миллиардов!), так что для раз-

мещения этих записей потребуется невероятно большой объем памяти компьютера, и при этом память может оставаться очень слабо заполненной, поскольку большинство комбинаций букв не образуют фамилий. Выход из этого затруднения можно найти, если предложить другое правило вычисления адреса логической записи. Например, если сложить номера (10+3+1+15+16+3= =48) букв фамилии ИВАНОВ, то полученную сумму 48 тоже можно рассматривать как адрес записи и разместить эту запись в 48-й ячейке памяти. Как видим, здесь адрес не является очень большим числом, и для размещения записи уже не требуется, чтобы память компьютера имела большой объем. Однако такой метод вычисления адреса будет довольно часто приводить к коллизиям, т. е. ситуациям, когда у двух разных записей получится один и тот же адрес. Например, для записи, соответствующей студенту по фамилии ИВАНОВ, тоже получится адрес 48.

Вообще, всякий метод вычисления адреса логической записи по ее содержимому называется *хеш-функцией*. Практически никакая хеш-функция не гарантирует от коллизий, поэтому на случай коллизии договариваются использовать некоторую другую (дополнительную) хеш-функцию. Этого обычно бывает достаточно для устранения всех коллизий. Если хеш-функция выбрана удачно, то логические записи распределяются более или менее равномерно по ячейкам памяти компьютера (точнее, по той области памяти, которая отведена под данный файл), и коллизии будут встречаться редко. Если бы, например, заранее была известна относительная доля появлений каждой буквы алфавита в большом количестве фамилий студентов, то можно было бы раз и навсегда сконструировать самую удачную из всех возможных хеш-функций. Однако такой благоприятной ситуации никогда не бывает.

При хешировании значение хеш-функции, т. е. адрес логической записи, вычисляется всякий раз при поиске этой записи в файле. Вычислить хеш-функцию несложно, и когда ее значение, т. е. адрес искомой записи, известно, выбор нужной логической записи осуществляется прямо по адресу. Поэтому хеширование — чрезвычайно быстрый метод поиска. При использовании хеширования дополнение файла новыми логическими записями тоже эффективно, поскольку при этом нет нужды сохранять какую-либо упорядоченность файла в целом.

Все рассмотренные методы поиска информации в настоящее время получили широкое распространение в системах управления небольшими базами данных для персональных компьютеров. Для СУБД, создаваемых на мощных компьютерах, разработаны другие, более сложные методы поиска, требующие более сложных, но зато и более эффективных программ.

Действие пользователя. Кратко рассмотрим типичные действия пользователя реляционной базы данных, желающего с помощью СУБД обрабатывать, например, стипендиальные ведомости студенческих групп. Каждая СУБД имеет определенный набор ко-

манд, которые она может выполнять. СУБД, разработанные для персональных компьютеров 5...10 лет назад, выполняют команды, которые представляют собой некоторые глаголы (обычно английские). Для общения с такими СУБД пользователю нужно набирать на клавиатуре эти глаголы, заставляя компьютер выполнять нужные действия. Многие СУБД, разработанные в последнее время, предоставляют пользователю более удобную возможность вводить команды не в виде глаголов, а путем непосредственного указывания на графический символ на экране, который соответствует желаемому действию. Такое указывание на графический символ (его обычно называют *пиктограммой*) заранее исключает все возможные ошибки пользователя, связанные с неправильным написанием команд в виде глаголов. Кроме того, каждая пиктограмма имеет легко запоминающийся внешний вид, что позволяет пользователю быстро осваивать набор команд данной СУБД.

Для удобства изложения будем предполагать, что в нашем распоряжении имеется разработанная в 1983 г. СУБД dBASE II (от англ. database — база данных). После запуска эта СУБД запрашивает у пользователя текущую дату и переходит в *командный режим*, т. е. ожидает от пользователя ввода очередной команды. Для создания файла стипендиальных ведомостей пользователю следует напечатать глагол CREATE (создать). В ответ на эту команду dBASE II запрашивает имя будущего файла. Дав этому файлу имя (например, Vedomost), пользователь получает приглашение сообщить имена и длины и типы всех полей, т. е. определить структуру логических записей этого файла. В данном случае каждая логическая запись файла Vedomost будет состоять из пяти полей: факультет, специализация, группа, фамилия, сумма. При этом поле «сумма» будет числовым, а все остальные поля — символьными (о типах данных см. гл. 3). СУБД dBASE II предлагает пользователю оформить структуру логических записей в виде табл. 9.1.

Здесь символы C и N в графе TYPE обозначают символьное (от англ. character — символ) и числовое (от англ. numeric — числовое) поле соответственно. Длины символьных полей выбраны с учетом поме-

Таблица 9.1. **Заполняемая пользователем таблица, описывающая структуру логических записей файла**

NAME (имя поля)	TYPE (тип поля)	WIDTH (длина поля)	DECIMAL PLACES (кол-во десятичных цифр дробной части числовых полей)
001 Faculte	C	20	
002 Specialis	C	40	
003 Familia	C	15	
003 Cruppa	C	5	
005 Summa	N	6	2

щаемой в них информации. Длина числового поля SUMMA и количество цифр в его дробной части выбраны так, чтобы в нем могли размещаться числа, имеющие до трех десятичных цифр в целой части (рубли) и две десятичные цифры в дробной части (копейки). Таким образом, с учетом десятичной точки длина поля SUMMA получается равной 6, а количество цифр в его дробной части — 2.

После заполнения пользователем таблицы СУБД dBASE II предлагает сейчас же начать ввод логических записей. Если пользователь отказывается от ввода, то СУБД переходит в командный режим. Если пользователь соглашается начать ввод, то СУБД выполняет команду APPEND (добавить). По этой команде на экране высвечиваются имена всех пяти полей. Справа от каждого имени отводится зона для ввода информации в соответствующее поле. В верхней строке экрана указывается номер вводимой логической записи. Все это изображено на рис. 9.7.

Тип вводимых пользователем данных контролируется в соответствии с табл. 9.1. Например, СУБД не позволяет вводить нецифровые символы в поле SUMMA.

После завершения ввода логических записей файл Vedomost будет содержать сведения о размере стипендий всех студентов института. Если студент не получает стипендию, ему будет соответствовать логическая запись со значением 0 в поле SUMMA.

Приведем несколько примеров простейших действий, которые может предпринять пользователь для обработки файла Vedomost.

Если пользователь хочет подсчитать общую сумму стипендий, выплачиваемых всем студентам института, то ему следует ввести команду SUM Summa (суммировать поле SUMMA). Если же пользователь хочет подсчитать сумму стипендий, получаемых студентами одной группы (например, группы М-243), ему следует ввести команду

SUM SUMMA FOR GRUPPA="M-243"

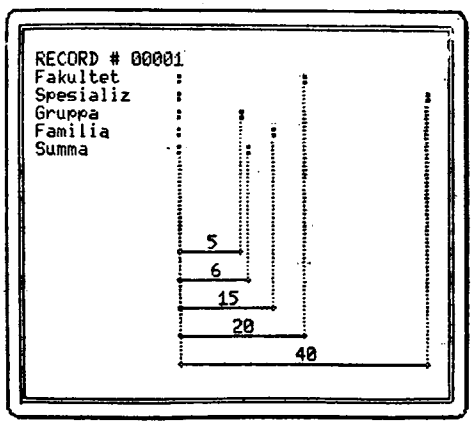


Рис. 9.7. Вид экрана при вводе информации

предписывающую суммировать поле SUMMA только для логических записей, содержащих значение М-243 в поле GRUPPA.

Если пользователь хочет подсчитать количество студентов в институте (т. е. количество логических записей файла Vedomost), ему следует ввести команду COUNT (подсчитать).

Если же пользователь хочет подсчитать количество студентов только в группе М-243, ему следует ввести команду

COUNT FOR GRUPPA="M-243"

предписывающую подсчитать количество только тех логических записей, которые содержат значение М-243 в поле GRUPPA.

Если пользователь хочет найти в файле Vedomost логическую запись, соответствующую студенту из

группы М-243, следует ввести команду LOCATE (найти):

```
LOCATE FOR GRUPPA="M-243"
```

В ответ на эту команду dBASE II выдает номер найденной записи или сообщит об отсутствии такой записи в файле.

Если пользователь хочет создать новую ведомость с именем MOV путем упорядочения логической записи старого файла Vedomost по возрастанию значений поля SUMMA, то следует ввести команду SORT (упорядочить):

```
SORT ON SUMMA TO NOV
```

Предположим, что перечисленные простейшие действия пользователь хочет выполнять в будущем многократно. В этом случае все свои действия ему целесообразно оформить в виде программы. Чтобы перевести СУБД dBASE II в режим ввода программы, пользователю следует ввести команду MODIFY COMMAND (модифицировать программу). В ответ СУБД запросит имя будущей программы. Пусть, например, пользователь дает своей программе имя Primer. Далее СУБД очищает экран, и пользователю следует ввести все перечисленные выше команды, которые программа Primer должна будет выполнять:

```
SUM SUMMA
SUM SUMMA FOR GRUPPA="M-243"
COUNT
COUNT FOR GRUPPA=M-243"
LOCATE FOR GRUPPA="M-243"
SORT ON SUMMA TO NOV
```

По окончании ввода программы СУБД переходит в командный режим, а текст программы Primer заносится в память. Теперь для запуска программы Primer пользователю достаточно ввести команду DO Primer (выполнить Primer). По окончании выполнения программы СУБД снова перейдет в командный режим.

Возможность составления программ на языке, понятном СУБД, является мощным средством разработки прикладных программ с использованием баз данных.

Глава 10

РАЗРАБОТКА ПЕДАГОГИЧЕСКИХ ПРОГРАММНЫХ СРЕДСТВ

В этой главе рассматривается понятие «педагогическое программное средство» (ППС), приводятся примеры ППС различной структуры: от простейших линейных до программ типа «проблемная среда»; предлагается поэтапная методика разработки ППС, построенная на личном опыте авторов; этап разработки и оформления сценария ППС, наиболее важный из всех этапов, пояснен на конкретном примере; рассматриваются принципы проектирования диалога обучающего с компьютером; дается анализ автоматизированных обучающих систем.

10.1. ПОНЯТИЕ О ПЕДАГОГИЧЕСКИХ ПРОГРАММНЫХ СРЕДСТВАХ И ИХ КЛАССИФИКАЦИЯ

Одно из перспективных направлений использования компьютеров — их применение в сфере образования. Способы использования компьютеров в образовании весьма разнообразны: на учебных занятиях, во время консультаций, в неучебной работе, в организации и управлении учебным процессом. Эти возможности компьютеров важно знать будущим инженерам-педагогам для работы в профессионально-технических училищах, техникумах и других учебных заведениях системы профтеобразования, где применение компьютеров становится сегодня ощутимой реальностью.

Использование компьютеров в учебном заведении должно быть комплексным. Поэтому уже сейчас предполагается на основе компьютерой техники и новых информационных технологий постепенно создать так называемую компьютеризированную среду обучения. В такой среде компьютер выступает не только как объект изучения в отдельных дисциплинах, но и как средство обучения и управления учебной деятельностью.

Рассмотрим некоторые возможности компьютера как средства обучения, направленного на организацию компьютеризированного обучения.

Компьютеризированное обучение представляет со-

бой рациональную систему распределения функций между преподавателем и компьютером при освоении или закреплении новых знаний, умений и навыков, обеспечивающую эффективное обучение. Эффективность компьютеризированного обучения достигается за счет некоторых бесспорных преимуществ компьютера перед другими средствами обучения.

Прежде всего это возможность организовать диалог с обучаемым: представлять учебный материал, задавать вопросы обучаемому, оперативно и конкретно реагировать на ответы обучаемого. Обсуждать с обучаемым алгоритм решения задачи, осуществлять контроль правильности решений и ответов обучаемого, представлять возможности обучаемому задавать вопросы и т. п.

Индивидуализация обучения — следующее преимущество компьютера. С помощью компьютера можно построить процесс обучения, учитывающий возможности каждого обучаемого усваивать новую учебную информацию и в соответствии с этим направлять процесс обучения по различным ветвям — уровням, от простых к более сложным понятиям, задачам, навыкам.

Также важно, что каждый обучаемый имеет возможность активно работать весь период обучения с компьютером, сам инициировать диалог и концентрировать свое внимание на наиболее трудных для него вопросах учебного материала.

Как техническое средство, компьютер представляет обучаемому учебную информацию в виде текстов, рисунков, чертежей, графиков, движущихся объектов, звукового сопровождения (рис. 10.1, 10.2). Современные изобразительные достоинства, которые еще могут быть усилены другими техническими средствами, работающими под его управлением, превращают компьютер в необходимого помощника преподавателя.

Большие возможности компьютера в обучении раскрываются при моделировании с его помощью различных физических явлений, процессов, технологий (рис. 10.3). Особенно это важно при выработке производственных навыков и умений, отработке действий в экстремальных ситуациях (рис. 10.4).

Много творческих возможностей предоставляет

компьютер преподавателю в организации нетрадиционных методик обучения.

При оценке роли компьютера в обучении следует иметь в виду и те недостатки, которые могут возникнуть из-за его применения в методически неоправ-

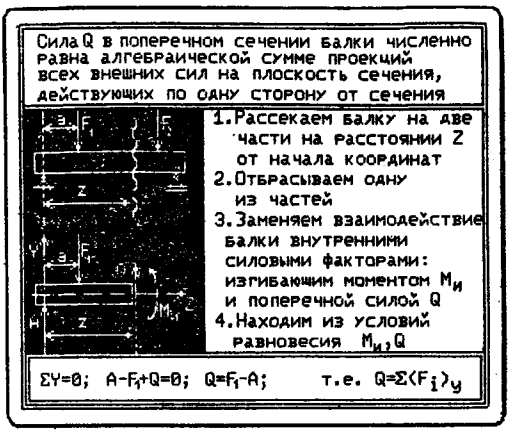


Рис. 10.1. Кадр учебной информации по технической механике

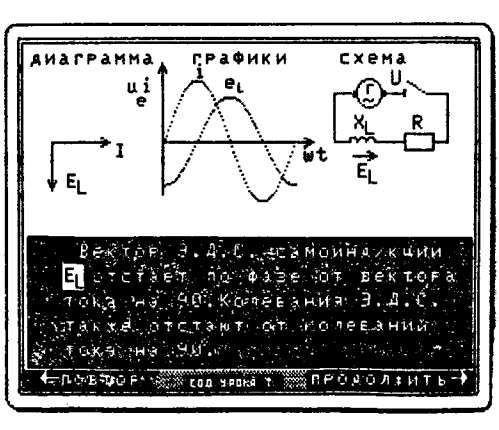


Рис. 10.2. Кадр учебной информации по электротехнике

данных ситуациях, при неполной реализации возможностей, ограниченности и неадекватности взаимодействия с обучаемым.

Некоторым условием использования компьютеров в обучении должна стать педагогическая компетентность преподавателей, высокий уровень их компьютерной грамотности.

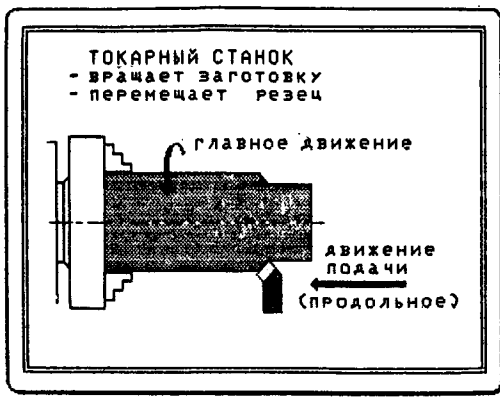


Рис. 10.3. Кадр учебной информации по спецтехнологии обработки металлов

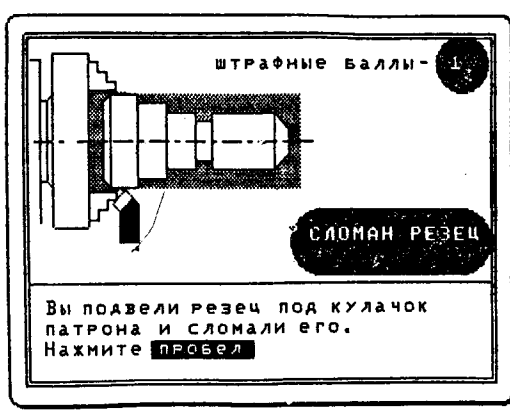


Рис. 10.4. Кадр учебной информации по технологии токарной обработки

Реализация возможностей компьютеров в обучении осуществляется с помощью компьютерных *программ учебного назначения.* Эти программы называют *обучающими* или, в более общем случае, *педагогическими программными средствами* (ППС).

Педагогические программные средства создаются и используются для достижения определенных педагогических целей и задач обучения. Они включают в себя учебный материал, который необходимо усвоить обучаемому, и управляющую часть, определяющую последовательность изучения этого учебного материала. Следовательно, вышеперечисленные достоинства компьютера концентрируются в возможностях ППС, с которыми работает обучаемый, а сам компьютер выступает в виде технического средства, реализующего обучающую программу.

В настоящее время разработано несколько тысяч ППС по самым различным областям знаний. Эти программы существенно различаются своими возможностями. Имеются очень простые ППС, предусматривающие последовательную выдачу учебных текстов, и достаточно сложные интеллектуальные программные комплексы.

Линейные обучающие программы направлены на последовательное изложение нового учебного материала в темпе, удобном для обучаемого. Такие программы, по существу, представляют собой последовательность кадров с учебной информацией (рис. 10.5), темпом сменяемости которых управляет обучаемый. Он имеет возможность, нажимая на различные клавиши, перейти к следующему или вернуться к предшествующему кадру программы. Структура таких программ представлена на рис. 10.6.

Контролирующие программы предназначены для проверки знаний, способностей или умений обучаемых. Наиболее простые контролирующие программы включают ряд вопросов или заданий, которые последовательно задаются обучаемому (рис. 10.7). Ответы обучаемого комментируются на экране и в итоге программа выдает протокол правильных или неправильных ответов обучаемого и, возможно, итоговую оценку (рис. 10.8).

Контролирующие программы могут иметь и другую структуру. В частности, такие программы за счет

развитых комментариев можно использовать для организации проблемных или игровых методов обучения.

Обучающе-контролирующие программы (рис. 10.9) представляют собой различные комбинации первых двух типов программ. Они могут быть линейной или разветвленной структуры.

Линейная структура обучения включает последовательное чередование обучающей и контролирующей частей программы. Обучающая часть программы может составить несколько кадров, объединенных в определенный обучающий фрагмент.

При работе с такой программой индивидуальные различия между обучаемыми проявляются в основном

Рис. 10.5. Один из последовательности учебных кадров по электротехнике

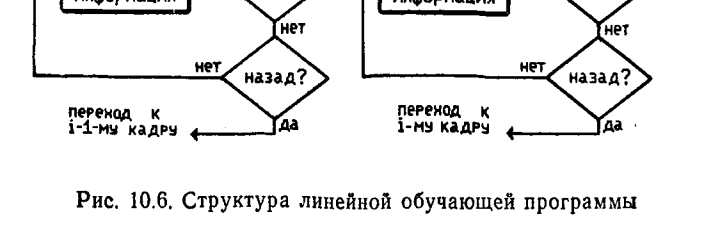


Рис. 10.6. Структура линейной обучающей программы

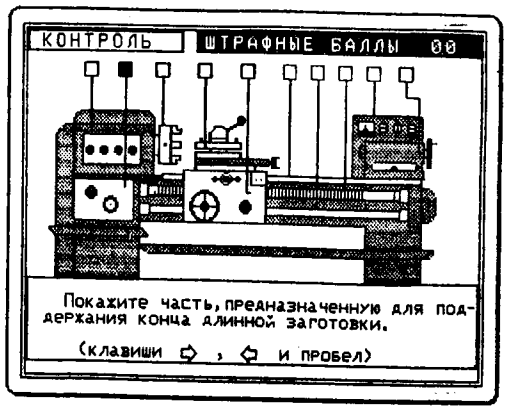


Рис. 10.7. Кадр контроля знаний по устройству токарно-винторезного станка

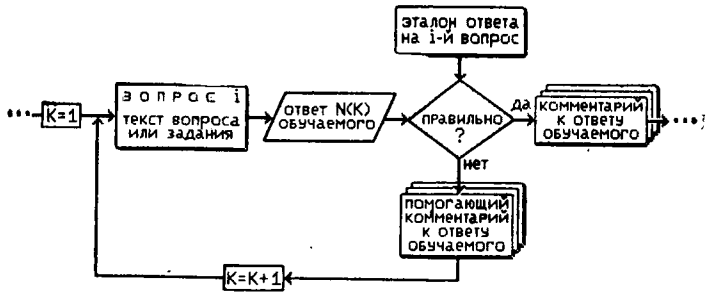


Рис. 10.8. Структура контролирующей программы

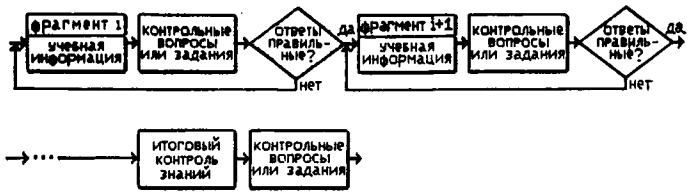


Рис. 10.9. Структура обучающе-контролирующей программы

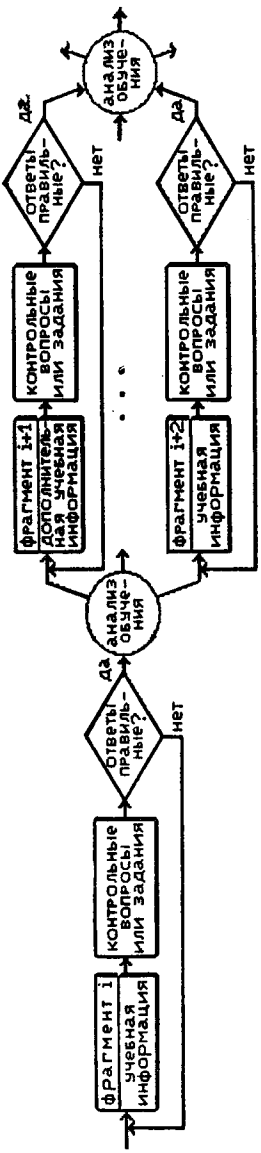


Рис. 10.10. Структура разветвленной обучающей контролирующей программы

в различном времени обучения и в результатах контроля.

Обучаемые, быстро справляющиеся с контрольными заданиями программы, могли бы в оставшееся у них время получить дополнительную учебную информацию и в соответствии со своими способностями продвинуться дальше в изучении предмета. Такие возможности обеспечивают разветвленные обучающе-контролирующие программы (рис. 10.10).

Программы представляют обучаемым информацию, рассчитанную на некоторого усредненного обучаемого. На основании анализа качества ответов обучаемого на вопросы или результатов контроля происходит разветвление на различные по сложности уровни обучения.

Обучающе-контролирующие программы часто используют и для закрепления пройденного учебного материала, тренажа определенных навыков и умений.

Имитационно-моделирующие программы основываются на уникальной способности компьютера моделировать сложные процессы и явления, графически отображая их на экране

монитора в наглядной форме. Это большая группа эффективных обучающих программ. Она включает программы, с помощью которых можно изучать отдельные явления и процессы по их модельному описанию. В эту же группу входят программы, которые предоставляют средства для проектирования новых машин, технологические процессы и системы управления по имеющимся моделям, описывающим конфигурацию и параметры отдельных устройств или узлов.

Имитационно-моделирующие программы, в которых помимо модели явления используется определенная игровая ситуация, контроль и оценка деятельности обучаемого, образуют самостоятельную подгруппу игровых программ.

Измерительно-контролирующие программы для сбора первичной информации в компьютер с измерительных датчиков и приборов, установленных на лабораторном оборудовании, предназначены для ее обработки и представления на этой основе состояния изучаемого оборудования в наиболее удобном, информативном виде. Такие программы используются при изучении явлений на основе проведения лабораторных экспериментов, исследовании в лабораторных условиях систем регулирования, программировании действий роботов, изучении на физических моделях вредных и опасных для человека процессов.

Обучающие программы типа «проблемная среда» включают набор вышеперечисленных типов программ, имеющих определенную предметную направленность. Обычно такого рода программные средства создаются для всестороннего изучения какой-либо узловой темы или раздела учебной дисциплины.

Выбор тактики учения с помощью «проблемных сред» выбирает сам обучаемый. Например, он (по заданию преподавателя или по личной инициативе) решает ряд конкретных задач, используя моделирование, автоматизированное проектирование, получение консультативной помощи и т. д.

Особую группу среди учебных средств составляют так называемые учебные инструментальные средства программирования, которые

позволяют изучить современные информационные технологии для обработки числовой, текстовой и графической информации, имеющиеся средства хранения и поиска информации и пр.

В реально создаваемых программах учебного назначения присутствуют различные части из вышеперечисленных типов ППС.

10.2. ЭТАПЫ РАЗРАБОТКИ ПЕДАГОГИЧЕСКИХ ПРОГРАММНЫХ СРЕДСТВ

Разработка ППС представляет собой процесс, связанный с проектированием компьютеризированной системы обучения, в котором учитываются психофизиологические особенности обучаемых и технические возможности компьютера.

Система обучения является сложной конструкцией. Она включает в себя два важных компонента — процесс передачи знаний, умений и навыков от обучающего к обучаемому и систему управления этим процессом.

Рассмотрим методику создания ППС, основанную на последовательности действий, направленных от конечных целей обучения к педагогическим задачам и средствам их достижения. Следует сразу оговориться, что в настоящее время теоретическая база, особенно ее психолого-педагогическая составляющая по разработке обучающих программ, пока только разрабатывается. Поэтому рассматриваемая здесь методика в значительной степени опирается прежде всего на личный опыт авторов.

Разработка ППС осуществляется поэтапно.

Этап 1. Определение конечных целей обучения по предмету (разделу, теме) и обоснование разработки обучающих программ. На этом этапе осуществляется процедура определения дидактических, развивающих и воспитательных целей учебной деятельности. Безусловно, лучше всего это делать, продвигаясь от общих целей предмета к частным целям тем и их частей. Однако если важность какой-то темы в общей совокупности целей предмета определена, можно сосредоточить свое внимание на формулировке ее целей и их поурочной разбивке. Эта работа требует определенного педагогического опыта, особенно учитывая ее

неформальный характер. Здесь можно выделить ближайшие и более отдаленные дидактические, развивающие и воспитательные цели. Кроме того, формулировка целей осуществляется на разных уровнях.

На концептуальном уровне цели формируются в общем виде. Например, дидактическая цель темы «Алгоритм и его свойства» предмета «Основы информатики и вычислительной техники» может быть сформулирована следующим образом: раскрыть понятие алгоритма, показать важную роль функциональной связи понятий «информация — алгоритм — компьютер», определяющей процесс автоматической обработки информации. Допустим, эта тема изучается на нескольких занятиях. Тогда цель одного из них, например, первого занятия — дать представление об алгоритмах и их исполнителях, показать фундаментальность понятия алгоритма, ознакомить обучаемых со свойствами алгоритмов и способами их описания.

На технологическом уровне цели детализируются до конкретных задач обучения в виде совокупности знаний, умений и навыков. Причем эти задачи должны быть сформулированы так, чтобы можно было оценить степень их достижения. Так, в результате первого занятия обучаемый должен знать определение алгоритма, его свойства, способы описания алгоритмов; должен уметь производить словесную запись алгоритма, составлять блок-схемы линейных алгоритмов для типовых исполнителей.

С учетом дидактических целей решается вопрос о способах их достижения исходя из набора средств, которые доступны преподавателю. В частности, здесь определяются те части занятия, которые будут поддерживаться с помощью ППС (возможно в сочетании с другими средствами).

Следует иметь в виду, что разумным основанием для разработки или использования ППС является получение педагогического эффекта. Иначе говоря, на ППС должны возлагаться такие функции в обучении, которые не могут быть столь же эффективно реализованы без него. Например, оперативная проверка знаний у группы обучаемых, выработка устойчивых логических или производственных навыков и пр.

Среди различных видов ППС наиболее целесооб-

разно разрабатывать программы, позволяющие моделировать различные явления или процессы, которые сложно или опасно исследовать в лабораторных или реальных условиях.

Выбор тех или иных частей процесса обучения, поддерживаемых ППС, обусловлен пониманием дидактических возможностей современных компьютеров и взвешенным подходом к оценке соотношения эффективности и затрат на компьютеризацию.

Таким образом, на первом этапе разработки ППС после формирования целей обучения и выбора наиболее целесообразной совокупности дидактических средств для их достижения становится понятным и обоснованным применение ППС для определенных частей учебных занятий. Здесь может быть установлена очередность разработки ППС в зависимости от их педагогической эффективности и трудоемкости.

Принципиально важным является то, что уже на первом этапе необходима взаимная увязка разработки ППС и методики их использования в учебном процессе.

Этап 2. Анализ и отбор учебного материала для ППС. На этом этапе осуществляется тщательный анализ учебного материала, который войдет в ППС и будет направлен на формирование у обучаемых знаний, умений и навыков, определяемых на первом этапе. Здесь речь идет о выборе наиболее существенных и выигрышных с точки зрения возможностей компьютера частей учебного материала. Естественно, что содержание учебного материала, включаемого в ППС, должно соответствовать современному уровню развития науки, техники и технологии, а также предполагать преемственность знаний, полученных на предшествующих этапах обучения. По возможности в этот материал должны входить рисунки и мультипликация (в меньшей степени текстовая информация), но с оглядкой на возможности применяемых компьютеров.

Весьма вероятно, что планируемая эффективность ППС может не достигаться из-за технических ограничений, и тогда от компьютеризации изучения данного раздела следует отказаться.

Результирующим документом после выполнения первых двух этапов разработки ППС является техни-

ческое задание на создание ППС, если это ППС новое или модифицируемое.

Этап 3. Выбор метода обучения и структуризация учебного материала. Учебный материал может излагаться разными методами. Например, используется последовательное изложение учебного материала с индуктивной или дедуктивной логикой. Учебный материал можно излагать и в историческом аспекте, а также путем создания проблемной или игровой ситуации.

Каждый из методов может охватывать весь цикл обучения от объяснения нового материала до постановки учебных задач и управления процессов их решения или его отдельные части. Выбор того или иного метода обучения осуществляется автором ППС и определяется эффективностью достижения поставленных целей обучения. Он должен стимулировать продуктивное, творческое мышление, вызывать интерес и мотивировать обучение.

В соответствии с выбранным методом формируется последовательность изложения учебного материала. Здесь главным является проектирование последовательности так называемых *обучающих воздействий.*

В системе обучения в качестве управляющего воздействия на обучаемого выступают обучающие воздействия. К ним относятся: выдача обучающей информации, вопроса, учебной задачи, подсказки, указания к проведению эксперимента и т. п. Обучающие воздействия — это динамическая составляющая ППС. Они имеют различную эффективность достижения конечных целей обучения. Наименее эффективным является выдача учебного текста, наиболее эффективным — контроль знаний, проведение компьютерного эксперимента. В ППС в должной мере должна присутствовать мотивационная составляющая, обеспечивающая осознанность работы в ППС.

Очень важным элементом структуризации материала является выбор формы изложения. Придание ей игрового характера, положительной мотивации в ходе обучения неразрывно связано с активностью и самостоятельностью действий обучаемого.

Заключительным продуктом этого этапа является укрупненная блок-схема ППС. В схеме обычно предусматривается контроль исходного уровня знаний,

включение мотивационных элементов, последовательность изложения учебного материала, меню возможных действий обучаемого, подсказки компьютера, точки контроля знаний, процедура сбора информации о ходе обучения и выработки рекомендаций для дальнейшего обучения.

Этап 4. Конструирование учебного сценария. При взаимодействии с компьютером обучаемый в каждый момент времени имеет дело с некоторой информацией, представленной в той или иной форме на экране дисплея. Такая визуализированная информация называется *кадром.*

Упорядоченная совокупность кадров с описанием структуры управления их модификацией или сменой называется *педагогическим сценарием* ППС. При конструировании сценария детально прорабатывается каждый кадр с учебной информацией и их логическая последовательность в зависимости от реакции обучаемого (например, рис. 10.1, 10.2). Обычно сценарий оформляется в табличной форме. Еще раз подчеркнем, что конструкция кадра во многом определяет эффективность ППС. Минимум текстовой информации, яркость, образность, динамичность демонстраций значительно увеличивают наглядность сложных явлений и процессов. При этом должны выдерживаться эргономические рекомендации к размещению учебной и управляющей информации на экране, цветовому решению, звуковому сопровождению и т. п. для максимально возможного учета психофизиологических особенностей восприятия информации обучаемым.

Наряду с созданием педагогического сценария на этом этапе составляется инструкция для программиста, детально указывающая структуру кадра, приемы выделения важной информации, модификацию информации в кадре, сочетание цветов, формы сбора информации о процессе обучения.

Этап 5. Программная реализация. Создание ППС на языке программирования или с помощью инструментальных средств для автоматизации этой работы (редакторов, специальных языков описания курсов, операционной среды автоматизированных обучающих систем и пр.) является наиболее трудоемким этапом.

По некоторым оценкам на 1 ч общения обучаемого с ППС требуется 150...200 ч на его создание. Имен-

но на этом этапе достигаются такие важные показатели ППС, как модульность ее структуры, надежность работы в различных режимах, защита от несанкционированных нажатий на клавиши, рациональное использование оперативной памяти, дизайнерская проработка обучающих кадров, организация «дружественного» диалога с обучаемым. Эти показатели влияют на будущую дидактическую эффективность создаваемого программного продукта.

На этом же этапе создается необходимая методическая документация, составляются описание применения ППС, методические указания пользователям — преподавателю и обучаемому.

Этап 6. Экспериментальная проверка ППС и их корректировка. Выбор учебного материала и методика его изложения должны быть подтверждены в ходе экспериментальной проверки. Для этого разработанные ППС используются на занятиях в нескольких группах обучаемых, и по полученным результатам делается предварительный вывод о эффективности ППС. Обнаруженные недоработки устраняют, вносятся изменения в сопровождающую документацию.

Этап 7. Сдача ППС в экспертный совет и внедрение. Каждое ППС представляет интерес не только как средство обучения, но и как концентрация опыта проектирования методики компьютеризированного обучения. Кроме того, ППС являются промышленным продуктом, который должен отвечать определенным стандартам и потребностям определенной группы потребителей.

При головных организациях страны, курирующих внедрение ППС для обучения различных категорий учащихся, созданы экспертные советы. Они призваны оценивать ППС и обобщать опыт их разработки и применения. Совет дает заключение о целесообразности и широте распространения оцениваемых ППС, после чего они передаются в специальный фонд, где могут приобретаться учебными заведениями, заинтересованными в их применении.

10.3. РАЗРАБОТКА И ПРЕДСТАВЛЕНИЕ СЦЕНАРИЕВ ППС

Педагогический сценарий аккумулирует важнейшие элементы проектируемого процесса обучения. Поэтому на его разработке остановимся подробнее.

Выбрав рациональный метод обучения, разработчик ППС приступает к *структуризации* учебного материала. Понимание того, что при наличии ППС обучаемые могут пользоваться учебниками, пособиями и другими привычными средствами классно-урочной системы, приводит к локальности решаемых с помощью ПСС задач и концентрации внимания на действительно необходимом учебном материале, нуждающемся в компьютерной поддержке.

Структуризация и последовательность расположения учебной информации по кадрам должны производиться на основе предполагаемой модели усвоения учебного материала. Иными словами, с точки зрения формирования у обучаемых системы умственных действий и операций.

Целесообразно строить обучающую последовательность таким образом, чтобы на основе определенного исходного уровня знаний раскрыть сущность выполняемых действий и операций, начиная с анализа условий до поиска и реализации способа решения. При этом желательно комментировать логику рассуждений и объяснение тупиковых ситуаций.

Основными элементами, управляющими формированием системы умственных действий и способствующими достижению целей обучения, являются обучающие воздействия. Как отмечалось ранее, к ним относятся порции нового учебного материала, контрольные вопросы и задачи, анализ изучаемого объекта или процесса, эвристическая беседа, машинный эксперимент. Эти воздействия в сочетании образуют крупные функциональные блоки педагогического сценария. Каждый такой блок разворачивается в некоторую совокупность учебных кадров. Несмотря на то что деление на кадры учебного материала, включаемого в сценарий, является весьма условным, оно позволяет формализованно представить описание сценария для последующего программирования.

По характеру представления информации кадры можно разделить на статические и динамические. *Статический* кадр с информацией появляется перед обучаемым в своем законченном виде, а *динамический* перестраивается в процессе демонстрации.

При размещении учебной информации в кадре желательно однотипную информацию размещать в зара-

нее определенных местах экрана — *информационных полях*. Такую разбивку на поля проводит автор педагогического сценария, ориентируясь на методическую целесообразность и эргономические критерии: минимальность числа полей, размещение наиболее важной информации в середине или в верхней левой части экрана, приоритет в размерах полей, небольшое количество используемых цветов и т. п.

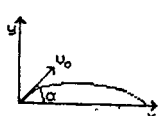
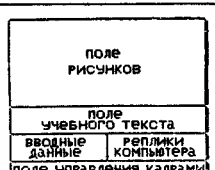
В зависимости от содержательной стороны информации в кадре их подразделяют на информационные, контролирующие, инструктивные, кадры-реплики. *Информационные* и *контролирующие* кадры несут основную нагрузку соответственно в изложении содержания учебного материала и при проверке качества его усвоения. *Инструктивные* кадры предназначены для информирования обучаемого о процедурах работы с ППС и выдаче рекомендаций о дальнейшем ходе обучения. *Кадры-реплики* обеспечивают оперативную обратную связь на ответы обучаемого в виде поощрительных или направляющих подсказок.

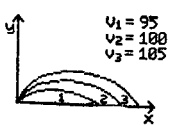
В сценарии ППС кадры могут перекрывать друг друга. Например, на информационном или контролирующем кадре может появиться реплика или инструкция о дальнейшей работе и пр. В начале работы с ППС сценарием предусматриваются вводные кадры, объясняющие назначение, структуру и состав ППС, а также имеющие мотивационный характер. Далее следуют информационные, в зависимости от назначения ППС, или сразу контролирующие кадры. После работы с рядом логически завершенных информационных кадров целесообразно включать контролирующие кадры. Вопросы и задания в них должны относиться к только что изученному материалу.

Контролирующие кадры имеют разнообразную структуру. В наиболее простом случае вслед за вопросом вызывается меню возможных ответов, в том числе правильных. Обучаемый должен набрать номер правильного ответа из меню. Этот метод контроля часто используется в ППС, хотя и обладает рядом недостатков.

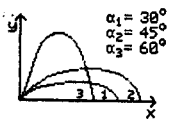
На поставленный вопрос обучаемый может ответить и числовым результатом или словом из представленного множества. Это ответы более сложного характера, близкого по форме к естественному. Одна-

Таблица 10.1. Сценарий обучающей программы «Движение тела, брошенного под углом к горизонту»

№ кадра	Содержание кадра		Возможные действия обучаемого		Реакции программы	
	Информация на экране	Описание кадра	Перечень действий обучаемого	Примечание	Анализ действий обучаемого	Направление переходов
1/п	Движение тела, брошенного под углом к горизонту	Внизу указывается разработчик программы и год разработки Сообщение: «Нажмите на любую клавишу» Музыкальная заставка	Нажатие любой клавиши		Если нажата любая клавиша	То переход к кадру 2/п
2/п		 На экране появляются оси координат и текст до точки (1). Затем рисуется тра-	1. Усвоение информации. 2. Нажатие клавиши →. 3. Нажатие клавиши 4. Нажатие клавиши 5. Нажатие клавиши ←		Если нажата клавиша → Если нажата клавиша ← Если нажата клавиша → Если нажата клавиша ←	То переход к кадру 3/п То переход к кадру 1/п То переход к кадру 7/п То переход к кадру 1/п

	Траектория движения тела, брошенного под углом к горизонту (1) Определяется начальной скоростью v_0 (2) и углом α Поле управления кадрами ↑начало ←назад →вперед ↓выход	ектория полета—[формула (10.1)] Текст до точки (2). На рисунке высвечивается вектор v_0, далее следует текст до конца, а на рисунке появляется угол α Внизу появляется поле управления кадрами				
3/п		Появляются оси координат и весь текст.	1. Усвоение учебной информации 2. Нажатие клавиши →. 3. Нажатие клавиши		Если нажата клавиша → Если нажата клавиша ← Если нажата клавища →	То переход к кадру 4/п То переход к кадру 1/п То переход к кадру 7/п

Продолжение табл. 10.1

№ кадра	Содержание кадра		Возможные действия обучаемого		Реакции программы	
	Информация на экране	Описание кадра	Перечень действий обучаемого	Примечание	Анализ действий обучаемого	Направление переходов
	Дальность и высота полета тела увеличиваются при возрастании начальной скорости (при α=const). Поле управления аналогично кадру 2/п	При разных начальных скоростях высвечиваются последовательно траектории полета	4. Нажатие клавиши. 5. Нажатие клавиши ←		Если нажата клавиша	То переход к кадру 2/п
4/п	 Высота полета тела возрастает при увеличении угла бросания, дальность полета максимальна при угле бросания α=45° (при v=const)	Оформление кадра аналогично предыдущему Углы бросания (град) до 45(1) 45(2) и более 45(3)	Аналогично предыдущему кадру		Если нажата клавиша → Если нажата клавиша ← Если нажата клавиша → Если нажата клавиша ←	То переход к кадру 5/п То переход к кадру 1/п То переход к кадру 7/п То переход к кадру 3/п

	Поле управления аналогично кадру 2/п					
5/п	Проверим, как усвоен учебный материал с помощью игры «Письмо Робин Гуда узнику замка Ноттинген». Письмо привязывается к камню, а камень посылается через бойницу стены замка в окно темницы. С какой начальной скоростью и под каким углом должен бросить камень Робин Гуд? Попадите с минимальным числом попыток Поле управления аналогично кадру 2/п	Вывод текста на весь экран	Аналогично кадру 3/п		Если нажата клавиша → Если нажата клавиша ← Если нажата клавиша → Если нажата клавиша ←	То переход к кадру 6/п То переход к кадру 1/п То переход к кадру 7/п То переход к кадру 4/п
6/п		Обучаемый вводит значение скорости и угла бросания На рисунке из руки Робин Гуда вычерчивается соответствующая траектория полета	1. Усвоение исходной информации 2. Набор цифр		Если набраны цифры скорости и угла бросания	То проверяется правильность их набора и соответствие допустимому диапазону, вычерчивается траектория.

Продолжение табл. 10.1

№ кадра	Содержание кадра		Возможные действия обучаемого		Реакции программы	
	Информация на экране	Описание кадра	Перечень действий обучаемого	Примечание	Анализ действий обучаемого	Направление переходов
	Укажите начальную скорость (число от 10 до 300) на угол бросания (от 15 до 75°)		3. Нажатие клавиш поля управления кадрами			Когда траектория попадает через бойницу в темницу, выдается сообщение «Робин Гуд попал». Указывается номер попытки и переход к кадру 7/п. В противном случае вычерчивается траектория до соприкосновения с препятствием, выдается сообщение «Мимо» и указывается номер следующей попытки, траектория стирается

					Если нажата клавиша →	То переход к кадру 7/n
					Если нажата клавиша ←	То переход к кадру 1/n
					Если нажата клавиша →	То переход к кадру 7/n
					Если нажата клавиша ←	То переход к кадру 5/n
7/n	Вы справились с заданием за P попыток. Работа с программой закончена До свидания	Картинка висит на экране до нажатия на определенную клавишу преподавателем				

ко и зафиксированная, и произвольная последовательность ключевых слов для ответа усложняет программную реализацию, хотя значительно повышает адекватность ответов и естественность диалога.

После правильного ответа на контрольный вопрос целесообразно давать поощрительные реплики и по возможности необходимый комментарий. Неправильные ответы также сопровождаются соответствующей репликой и приводится направляющая подсказка.

В заключении сценария ППС обычно указывается форма протокола, фиксирующего ход и результат обучения, а также приемы сбора необходимой для этого информации.

Сценарий ППС представляется в формализованном виде, обычно с помощью таблицы. В ней помимо описания содержания учебного материала содержится логическая структура управления переходами между информационными кадрами в зависимости от ответов обучаемого или выбранной тактики обучения. Возможный вариант таблицы сценария представлен в табл. 10.1.

Если содержание учебного кадра носит сложный композиционный характер, то он может вычерчиваться отдельно и служить приложением к таблице. При сложной логике переходов между кадрами можно дополнительно к табличному представлению сценария составить покадровую блок-схему педагогического сценария.

В качестве примера рассмотрим сценарий обучающей программы по физике на тему: «Движение тела, брошенного под углом к горизонту». Целью создания программы является изучение основных закономерностей движения тела, брошенного под углом к горизонту, и моделирование траектории полета. Обучающая программа состоит из двух частей: информационной, направленной на усвоение закономерностей движения тела, брошенного под углом к горизонту, и моделирующей, связанной с моделированием траектории необходимой конфигурации. Будем считать, что обучаемый усвоил изученные закономерности, если он сумел получить необходимую траекторию полета за определенное количество попыток.

Движение тела, брошенного под углом к горизон-

ту, описывается формулой (сопротивлением воздуха пренебрегаем):

$$y = x \operatorname{tg} \alpha - x^2 \frac{g}{2v_0 \cos^2 \alpha}. \qquad (10.1)$$

Конечные точки траектории на оси x определяются формулами

$$x_0 = 0; \qquad (10.2)$$

$$x_к = v_0^2 \sin 2\alpha / g. \qquad (10.3)$$

Пример сценария обучающей программы приведен в табл. 10.1.

10.4. ОРГАНИЗАЦИЯ ОБУЧАЮЩЕГО ДИАЛОГА

Процесс обучения с помощью ППС имеет диалоговый характер. Предусмотренный сценарием диалог носит форму обмена сообщениями между компьютером и обучаемым. При его реализации должны быть созданы такие условия общения, которые вызывают у обучаемого чувство удовлетворения. Но диалог должен быть построен не только в удобной форме, но и направлен на стимулирование любознательности и развитие профессиональных навыков обучаемого. Для работы с ППС от него не должно требоваться специальных навыков. Обычного лингвистического и технического объема знаний и умений должно быть достаточно для ведения диалога с компьютером.

При проектировании диалогов выработался некоторый набор принципов, принимаемый за основу при разработке сценариев. Учет этих принципов скорее позволит избежать грубых ошибок, но не является гарантией разработки эффективного диалога. Только тщательное изучение подготовленности обучаемых, их требований, стиля размышления позволит определить действительно полезные конструкции.

Рассмотрим каждый из этих принципов.

Учет логики действий и корректировка ошибок обучаемого. При построении сценария принимается во внимание, что действия обучаемого делятся на группы, каждая из которых имеет свою внутреннюю логику.

В зависимости от степени педагогической проработки реакции компьютера на действия пользователя

могут носить разный характер. Например, на ввод ответа обучаемым может выдаваться безадресная реплика.

"НЕВЕРНО. ПОПРОБУЙТЕ ЕЩЕ РАЗ".

Это наиболее простая форма коррекции ошибки, не очень удачная и ее можно использовать только при исправлении механических ошибок.

Более развитой формой реакций компьютера на ответ обучаемого является предметное разъяснение ошибки. Например, на ответ обучаемого компьютер выдает сообщение:

" ПОДУМАЙТЕ ЕЩЕ. ТОК В ОБЩЕЙ ЧАСТИ ЦЕПИ ДОЛЖЕН БЫТЬ РАВЕН СУММЕ ТОКОВ В РАЗВЕТВЛЕНИЯХ".

Следующей формой ответа является педагогически направленная реплика. В ней помимо указания на конкретную предметную ошибку приводятся разъяснения о последствиях введенного ответа, например:

"АМПЕРМЕТР В ЦЕПИ ВКЛЮЧЕН НЕВЕРНО. ИЗМЕРИТЕЛЬНАЯ СИСТЕМА АМПЕРМЕТРА ОБЛАДАЕТ МАЛЫМ СОПРОТИВЛЕНИЕМ, ПОЭТОМУ ЕГО СЛЕДУЕТ ВКЛЮЧАТЬ ПОСЛЕДОВАТЕЛЬНО. ИЗМЕНЕНИЕ ШКАЛЫ ИЗМЕРЕНИЙ ОСУЩЕСТВЛЯЕТСЯ ШУНТОМ, ПОДКЛЮЧАЕМЫМ К АМПЕРМЕТРУ ПАРАЛЛЕЛЬНО. МЕТОДИКА РАСЧЕТА ШУНТОВ РАССМАТРИВАЕТСЯ В ПЕРВОЙ ЧАСТИ."

Реакции компьютера должны видоизменяться в зависимости от логики рассуждения обучаемого при ответах на вопросы или выполнения заданий, а также от числа попыток дать правильный ответ.

Сценарием должна быть предусмотрена форма обращения к обучаемому на «Вы», не допускающая отрицательных оценок его способностей, категорических реплик.

Не исключена ситуация, когда обучаемый выдает ответ, не предусмотренный автором ППС. Здесь можно заготовить серию уточняющих вопросов или просто посоветовать обратиться к преподавателю.

Соответствие содержания диалога категории обучаемых. Соблюдение этого принципа должно приблизить диалог к привычному для данного круга обучаемых виду (по профессии, уровню образования, опыту) и предпочтительным, привычным для них формам

представления учебного материала, принятым обозначениям.

Реализация обратной связи и эргономичность представления информации. Реализация гибкой обратной связи является главной компонентой в диалоге. Быстрый ответ на действия обучаемого, простые расчеты должны выполняться в пределах 300 мс.

Важно, чтобы обучаемый постоянно чувствовал реакции ППС на каждое свое действие. Отсутствие реакции, неясность или затруднительность получения ясного ответа будят беспокойство и порождают неудовлетворенность.

Учебная информация, меню возможных режимов работы с ППС и реакции-ответы компьютера должны быть максимально информативными.

Наиболее важная информация, требующая концентрации внимания, выделяется средствами компьютерной графики (подчеркивания, размещения в окнах, выделения цветом, мигания и т. п.).

Все текстовые сообщения должны быть достаточно лаконичными, термины и определения сформулированы из текстового окружения, с которым встречается обучаемый.

В меню режимов целесообразно пользоваться так называемыми пиктограммами — стилизованными маленькими рисунками, напоминающими требуемое действие или режим. Ответы, набираемые обучаемым на клавиатуре компьютера, должны дублироваться на экране монитора.

Диалог должен способствовать тому, чтобы обучаемый чувствовал полноценность собственной деятельности. Однако не следует злоупотреблять диалогом. В ППС должна выдерживаться разумная частота обращений к обучаемому, не вызывающая у него представления о непонятливости компьютера.

Оказание помощи в учении. Для продуктивности диалога у обучаемого должна быть возможность получить помощь. Об этом следует в самом начале сценария предупредить обучаемого. Нажатием определенной клавиши или через систему подсказок обучающий может получить ответ, предложенный разработчиком ППС, но за это ему снижают оценку или в протокол обучения заносят факт обращения за помощью.

Кроме такого рода помощи обучаемый должен

иметь возможность получить операционную помощь, связанную с объяснением тех режимов работы, которые ему предоставлены, с назначением специальных клавиш и т. п.

Любая помощь, оказываемая обучаемому, должна адекватно соответствовать его потребностям и осуществляться своевременно.

Рассмотренные здесь принципы проектирования диалога следует творчески применять при разработке конкретных ППС. Структура и содержание ППС во многом зависят от выбранного метода обучения, содержания учебного материала, педагогического опыта разработчика, знаний в области психологии обучения, возможностей компьютера и т. д.

Опыт разработки удачного диалога можно почерпнуть из перечня ППС, рекомендуемого преподавателем для изучения этого вопроса.

10.5. АВТОМАТИЗИРОВАННЫЕ ОБУЧАЮЩИЕ СИСТЕМЫ

Одно из направлений автоматизации описания сценариев ППС в инструментальной программной среде компьютера и последующего использования такого рода программных средств для обучения связано с применением средств автоматизированных обучающих систем (АОС) на базе компьютеров.

Автоматизированная обучающая система представляет собой комплекс программно-технических и учебно-методических средств, обеспечивающий в диалоге с компьютером активную индивидуальную учебную деятельность.

При проведении занятий в классах, подключенных или оснащенных АОС, возрастает активность обучаемых, самостоятельно перерабатывается большой объем учебной информации, расширяются возможности преподавателя по управлению учебной деятельностью. По имеющимся отечественным и зарубежным данным, применение АОС в 1,5...2 раза сокращает время и повышает качество обучения.

Использование АОС направлено на выполнение компьютером следующих функций: управление учебной деятельностью; хранение и выдача учебной информации; моделирование явлений, процессов и закономерностей; анализ сообщений и ответов обучае-

мых: регистрация, хранение и обработка результатов учебной деятельности обучаемых.

Пользователями АОС могут быть преподаватели, обучаемые, авторы автоматизированных учебных курсов, диспетчер.

Преподаватель организует работу обучаемых с учебными средствами АОС — определяет цели и формы занятий, контролирует работу обучаемых.

Обучаемый — основной пользователь АОС. Пользуясь предоставленными средствами, он может обучаться и контролировать свои знания по различным учебным предметам.

Автор обучающих курсов составляет диалоговые программы на основе принципов обучения конкретной дисциплине и алгоритмической модели автоматизированного обучения.

Диспетчер ведет библиотеку курсов, банк данных и организует работу пользователей с имеющимися средствами.

Основным техническим средством АОС является компьютер. Для этих целей используются большие компьютеры серии ЕС ЭВМ, мини- и микрокомпьютеры.

Пользователи АОС взаимодействуют с программными учебными средствами с помощью клавиатуры и дисплеев, являющимися средствами отображения вводимой и результирующей информации.

Программные средства АОС обеспечивают ввод, хранение, обработку и предъявление учебной информации. Они оформляются чаще всего в виде пакета прикладных программ или программной системы. Структура программной системы представлена на рис. 10.1.

Учебно-методическое обеспечение АОС представляет собой совокупность учебного материала, методических указаний, библиотеки курсов и статистических данных обучения.

Автоматизированные обучающие системы делятся на специализированные, предназначенные для обучения какому-то одному предмету, и универсальные, обеспечивающие возможность одновременного обучения нескольким предметам. Значительная часть ранних специализированных АОС была связана с обучением математике, языкам программирования, основам

Рис. 10.11. Структура программной системы

вычислительной техники. Потребности эффективного использования компьютеров, особенно больших, и имеющиеся у них возможности приводят в настоящее время к преимущественной разработке универсальных АОС.

Технически и программно АОС развиваются в нескольких направлениях: по пути интеграции АОС с системами коллективного пользования на базе больших компьютеров и в направлении миниатюризации систем на базе СМ ЭВМ и персональных компьютеров.

В США характерной системой первого направления является система PLATO-IV фирмы CDC, работающая на базе сети CYBERNET, состоящей из 15 больших компьютеров. С ней одновременно могут работать 1000 пользователей по различным учебным курсам.

У нас в стране первые разработки АОС также связаны с большими компьютерами. Одна из первых разработок (1977 г.) — система программирования обучающих курсов (СПОК) выполнена в институте кибернетики им. В. М. Глушкова АН УССР для ЕС ЭВМ. В последующем (1982—1985 гг.) ее развитые редакции АОС — ВУЗ, АОС — ВУЗ/ОСКАР, АОС ВУЗ — М стали типовыми АОС для вузов страны.

В настоящее время существует несколько версий

системы АОС — ВУЗ, обладающих различными возможностями.

Системы АОС — ВУЗ/ФОКУС, АОС — ВУЗ/ПРИМУС, АОС — ВУЗ/АТОС представляют собой интегрированные пакеты, расширяющие диалоговые возможности АОС — ВУЗ за счет включения в нее развитых средств коллективного пользования и разделения времени. Пакет АОС — ВУЗ/БОНД обеспечивает совместную работу АОС с системой управления базами данных.

Обучение в этих системах осуществляется по программам, заранее составленным авторами курсов. В них возможности обучения ограничены вариантами порционного изучения учебного материала, которые предусмотрел автор учебных курсов. Здесь трудно учесть индивидуальные особенности восприятия учебного материала обучаемым.

Более гибкое управление познавательной деятельностью связано с системами ПРОЛОГ — ЕС и ЗАПСИБ/ПРОЛОГ — ЕС, в которых генерация компонентов обучающего курса строится на основе модели изучаемой предметной области. В таких системах у обучаемого появляется возможность обращения с вопросами к различным фрагментам этой модели. В сочетании с целенаправленной программой обучения, составленной автором курса, это позволяет заметно повысить эффективность обучения.

Второе направление связано с развитием миникомпьютеров (СМ ЭВМ) и локальных сетей учебного назначения на базе персональных компьютеров. Среди обучающих систем, продолжающих идеологию построения АОС — ВУЗ для мини- и персональных компьютеров (ДВК), является мобильная обучающая система АОС — М.

Особенности персональных компьютеров, связанные с техническим упрощением ведения диалога с обучаемым, ограниченные ресурсы компьютера по быстродействию и памяти повлияли на архитектуру и функциональные возможности АОС — М.

Прежде всего, учитывая разнообразие применяемых персональных компьютеров для обучения, встала проблема переносимости (мобильности) АОС — М с одного вида компьютера на другой. Это потребовало

разработки специальной технологии написания программного обеспечения системы.

На первом этапе работы система определяет категорию пользователя (автор, обучаемый, диспетчер) по его индивидуальному шифру. В зависимости от категории пользователя диалог с системой развивается по разным направлениям. Диалог с пользователем поддерживает программа-интерпретатор. Одной из команд для интерпретатора является запрос на вызов учебного курса. Если соответствующий курс зарегистрирован в системе, он выдается пользователю.

Создание учебных курсов в АОС — М осуществляется по тем же принципам, что и в системе АОС — ВУЗ.

Подготовка автоматизированных курсов в этих системах включает: определение или уточнение целей обучения, конкретизируя задачи, возлагаемые на лекционные и лабораторные занятия, и задачи, решаемые в рамках АОС; разработку логических схем действий обучаемых и алгоритмов, при которых подготавливают все необходимые учебные и методические материалы для описания их в АОС.

Для упрощения технологии подготовки курсов целесообразно пользоваться типовыми схемами организации диалога. Например, обучаемому выдается текст учебного материала, затем вопросы, упражнения или задачи с сопровождающей их решение дополнительной информацией и выдается результирующий протокол обучения.

Язык описания курсов (ЯОК). Он используется для описания (программирования) учебного материала. В алфавит ЯОК входят буквы русского и латинского алфавитов, цифры и специальные символы. Основными элементами языка являются операторы, метки, макросредства, специальные области памяти курса. Операторы делятся на три уровня.

Операторы *первого уровня (проблемные)* включают: QU — вопрос, RD — чтение, PR — проблему. С их помощью обучаемому можно задавать вопрос, представить на экране учебную информацию.

В состав операторов *второго уровня (обработки ответа)* входят: CA — правильный ответ, WA — неправильный ответ, AA — предполагаемый ответ, EA — конец ответа и пр.

В состав операторов *третьего уровня (служебные)* входят: EP — ожидание ввода, TY — вывести, LB — метка, BR — перейти, RE — управление экраном и пр.

В качестве примера представим на ЯОК учебный материал по курсу общей электротехники.

```
LB    ТЕОРИЯ
DE
RD    Для переноса зарядов в замкнутой цепи источник
      электроэнергии затрачивает энергию, равную произ-
      ведению ЭДС источника на количество электричест-
      ва, перенесенного через эту цепь, т. е. E* Q
PR
DE
RD    Таким образом, источник энергии производит полез-
      ную работу, равную A=U *Q, где U — напряжение
      на зажимах приемника. Так как Q=I *T, формулу
      работы можно записать в виде
                        A=U *I* T
      С учетом того, что U=I* R,
                        A = I ** 2* R* T
PR
DE
RD    Мощностью называется работа, производимая в 1 с.
      Мощность определяется по формулам
            P=A/T=U* I=U ** 2/R=I ** 2/R.
LB    ВОПРОС 1
QU    ЧЕМУ РАВНО Р ПРИ U=100 В и I=2A?
CA    200
TY    ВЕРНО
AA    ДВЕСТИ
TY    ВЕРНО, НО ЛУЧШЕ ОТВЕТИТЬ ЦИФРОЙ
WA    50
TY    НЕВЕРНО
. . . . . .
```

Из этого текста программы ясна структура записи учебного материала на ЯОК и тот достаточно высокий уровень знаний в области информатики, который требуется от автора автоматизированного курса.

Ввод материала курса осуществляется в следующем порядке: регистрация курса, подключение курса в режиме автора, генерация стандартных разделов курса, ввод учебного материала, проверка текстов и корректировка курса в режиме обучаемого.

Для миниЭВМ (СМ-4, СМ-1420) и персональных компьютеров (ДВК-2М, ДВК-3 и пр.) в Московском институте электронной техники разработана автоматизированная система диалогового обучения АСТРА/МИКРО, которая в диалоге с преподавателем на естественном языке с использованием принципов «ме-

ню» и «интервью» позволяет сформулировать учебные курсы для группового или индивидуального обучения и контроля знаний.

В среде АСТРА/МИКРО создаваемые учебные курсы хранятся в библиотеке. При этом сохраняется информация об авторе курса, дате создания и последней корректировке. Система обеспечивает проверку корректности курса, копирование фрагментов курса, санкционированный доступ к курсу и другие сервисные функции.

АСТРА/МИКРО имеет развитые средства диалогового управления базой данных с курсовым обеспечением, средства сбора статистики обучения и контроля, средства машинной графики, возможности обеспечения расчетов в режиме калькулятора и пр.

Мобильная инструментальная диалоговая обучающая система (МИДОС) разработана в Московском энергетическом институте. Базовым компьютером для системы МИДОС является ДВК-2М и старшие модели компьютеров этой серии.

При работе с МИДОСом не требуются знания языков программирования. Автор, создавая автоматизированный учебный курс (АУК), предварительно разбивает учебный материал на отдельные обучающие кадры, описывает назначение каждого кадра и указывает граф-схему переходов от кадра к кадру и условия этих переходов.

В системе МИДОС автор работает с объектами своей предметной области, не пользуясь каким-либо специальным языком.

Вводимый преподавателем обучающий курс записывается на отдельном, рабочем гибком магнитном диске.

Рассмотрим подробнее работу с такого рода системой. Работа МИДОСа начинается с заставки. Система запрашивает имя курса, с которым предполагается работать, и проверяет его наличие на рабочем диске. Если курс объявлен в системе, требуется указать пароль (ключевое слово) и свою фамилию. После этого работа с курсом начинается с входного меню.

Автору предлагается на выбор три возможности: вводить или редактировать структуру курса; вводить или редактировать тексты курса; отлаживать курс в режиме имитации обучения.

Ввод структуры курса состоит в указании номера кадра и описании переходов в зависимости от условий стандартных действий обучаемого типа «дальше», «назад», «помощь». При создании структуры курса можно обращаться к подпрограммам, написанным на языке БЕЙСИК, для создания динамических моделей или текстов.

В этом же режиме указывается тип высказывания обучаемого на задаваемый ему вопрос или задачу (если таковые имеются в кадре) из меню: стандартные высказывания, число, слово (фраза), ключевое слово, алгебраическое выражение. Здесь же вводятся эталонные ответы и переходы в зависимости ответов обучаемого.

После окончания работы в этом режиме на диске фиксируется структура курса.

Ввод текста происходит после указания номера кадра и его вызова системой. Для ввода в текст кадра графиков, таблиц, рисунков используется режим «псевдографика».

В начале работы с готовым курсом на экране распечатывается информация для обучаемого по работе с курсом — как вводить высказывания, в какой форме, как пользоваться клавиатурой и т. д.

Обучаемый, воспринимая учебную информацию, имеет возможность в зависимости от темпа ее восприятия и понимания передвигаться по курсу вперед или назад, просить дополнительную помощь.

При необходимости вычислений обучаемый может запросить режим «калькулятор».

Статистика о ходе обучения сохраняется, и на каждом шагу обучаемый может судить об успехе своей работы.

Внедрение рассмотренных систем осуществлено в основном в вузах. Их массовое применение в техникумах, училищах и школах сдерживается дороговизной и качеством применяемых компьютеров, сравнительно высокими требованиями к компьютерной грамотности преподавателей, отсутствием качественных методик оценки эффективности компьютерного обучения и пр.

Модель компьютера	Процессор	Быстродействие, тыс. оп/с	Емкость		Внешний накопитель	Устройство отображения информации	Операционная система, языки программирования
			ОЗУ	ПЗУ			
«Агат»	6502	200	32	16	НГМД ЕС-5089	Монохромный монитор на 2048 символов, 512 • 256 точек	ДОС, БЕЙСИК
«Электроника БК-0010 (ш)»	К1801ВМ1	500	Пользователя 16, экранной 16	32	Кассетный магнитофон	Бытовой телевизор, 1600 с символов, 512*256 точек	Системные компоненты в ПЗУ, ФОКАЛ, БЕЙСИК
«Корвет»	КР580ИК 80А	600	64	32	НГМД (133 мм, 2*360К)	Монохромный монитор (телевизор) на 1536 символов, 512*256	МикроДОС СР/М БЕЙСИК, ПАСКАЛЬ, ФОРТ РАПИРА, СИ
УКНЦ	К18016М	500	64	16	НГМД (133 мм 2*720К)	Монохромный монитор на 1920 символов, 640*264 точек	ФОДОС, БЕЙСИК, ФОРТРАН, ПАСКАЛЬ

Приложение 2. Краткий обзор по языку программирования ПАСКАЛЬ

ПАСКАЛЬ является структурным языком высокого уровня, позволяющим определять пользовательские типы данных и писать программы с использованием принципа иерархической организации. В настоящем приложении представлен обзор лишь некоторого подмножества языка, охватывающего его основные элементы.

Основные символы языка ПАСКАЛЬ — буквы, цифры и специальные символы.

В языке используются буквы латинского алфавита. Для определенных целей можно пользоваться и буквами русского алфавита.

Для отображения чисел используются цифры от 0 до 9.

Специальные символы включают:

а) знаки операций:

\+ — * / =<> < > <= >= :=

б) ограничители:

. , (:) [:] {:} V : ; '

Ключевые (зарезервированные) слова языка ПАСКАЛЬ

AND	END	NIL	SET
ARRAY	FILE	NOT	THEN
BEGIN	FOR	OF	TO
CASE	FUNCTION	OR	TYPE
CONST	GOTO	PACKED	UNTIL
DIV	IF	PROCEDURE	VAR
DO	IN	PROGRAM	WHILE
ELSE	MOD	REPEAT	WITH

Числа записываются по аналогии с записью на языке БЕЙСИК.

Последовательность символов, заключенных в апострофы, представляет собой символьную константу.

Переменная обозначается последовательностью букв и цифр, начинающейся с буквы. Распознается переменная по первым восьми символам.

Для записи арифметических выражений используются арифметические операции: + (сложение), — (вычитание), * (умножение), / (деление), MOD (остаток от деления), AMOD B), VID (целая часть при делении, AVID B).

Правила записи арифметических выражений аналогичны правилам языка БЕЙСИК.

В арифметических выражениях можно использовать стандартные функции, представленные в табл. П2.1.

Операции отношения и логические операции записываются по аналогии с языком БЕЙСИК.

Программа на языке ПАСКАЛЬ состоит из заголовка и тела, называемого еще блоком. В заголовке программы вслед за служебным словом program указывается имя программы, а в круглых скобках — список параметров, связывающий программу

Таблица П.2.1. **Стандартные функции языка ПАСКАЛЬ**

№ п/п	Математические функции	Запись на языке ПАСКАЛЬ	Примечания
1	$\|X\|$	ABS (X)	
2	arctg x	ACCTAN (X)	Функция из интервала $-\pi/2<X<\pi/2$. Аргументы тригонометрических функций заданы в радианах
3	cos x	COS (X)	
4	e^x	EXP (X)	
5	ln x	LN (X)	
6	Ближайшее целое kX	ROUND	$X>0$
7	sin x	SIN (X)	
8	X^2	SQR (X)	
9	$\sqrt{X}$	SQRT (X)	$X>0$

с операционной системой. Обычно это имена входного (input) и выходного (output) файлов. Заканчивается заголовок точкой с запятой.

Для записи комментария используются фигурные скобки.

Блок следует за заголовком, программа заканчивается точкой.

Блок имеет две основные части: раздел описания данных и раздел операторов.

Под *данными* подразумеваются те объекты, с которыми работают операторы программы. При описании данные должны быть обязательно отнесены к одному из типов, определенных в языке.

В разделе описаний блока задаются имена данных, определяется их тип, возможные значения.

Для описания данных используются описатели. К описателям стандартного (базового) типа относятся: integer (целый), real (действительный), boolean (булевский), char (символьный).

Тип относится ко всем константам (CONST), функциям (FUNCTION), процедурам (PROCEDURE) и переменным.

Кроме стандартного типа данных к так называемым скалярным или простым типам данных относятся и перечисляемые данные, которые позволяют разработчику программы в разделе описания type создавать собственные типы данных, задаваемые перечислением.

В языке имеются и сложные типы данных. Среди четырех сложных типов упомянем лишь об одном — регулярном типе данных, определяющем массивы с помощью описателя array.

Например, блок описания данных на языке ПАСКАЛЬ может иметь вид:

```
program вычисление (input, output); {заголовок} {начало блока описания данных}
const
a=4.5;
```

Таблица П2.2. **Основные операторы языка ПАСКАЛЬ**

Наименование оператора	Структура оператора	Примечания	Примеры применения
Оператор присваивания	V : = A	V — переменная или функция; A—выражение	X : =2; Y = Y + 1; L : =a mod b
Операторы ввода данных	READ (A1, A2, ..., AN)	Чтение из файла input N значений и присваивание их переменным A1, A2, ..., AN	READ (X, Y)
	READ (A1, A2, ..., AN)	Чтение из файла input N значений и пропуск остальных значений до следующей строки файла и присваивание их переменным A1, A2, ..., AN	READLb (A, B, C)
Операторы вывода данных		Реализует вывод значений переменных в файл OUTPUT в одну строку	WRITE (X, Y)
	WRITELN (A1, A2, ..., AN)	Осуществляет вывод значений переменных A1, A2, ... AN в файл OUTPUT и переход к следующей строке	WRITE (A, B, C)
Оператор безусловного перехода	GOTO	N — метка оператора, к которому предписывается переход	PROGRAM... LABEL 50, 15; begin goto 50 50 : x : = 4
Оператор условных переходов	IF THEN S1 [ELSE S2]	L — булевское выражение; S1, S2 — простые или сложные операторы; ветвь ELSE может отсутствовать	IF X < 0 THEN Y : = 1/X ELSE Y : = X
Оператор цикла с предусловием	\\ H LE ALOS	A — выражение; S — оператор	WHILE X > 0 DO X=SQRT(X)
Оператор цикла с постусловием	REPEATS	S — оператор; A — выражение	REPEAT X=SQRT(X) UNTIL X > 0.1

Продолжение табл. П2.2

Наименование оператора	Структура оператора	Примечания	Примеры применения
Оператор цикла с параметром	FOR I : = A DO (DOWNTO) B DO S;	I — управляющая переменная; A, B — выражения, задающие начальное и конечное значения для I; S — оператор. При возрастании I пишется DO, при уменьшении — DOWN TO	FOR I : =1 TO 5 DO Y[I] = =SIN (I); FOR J : =10 DOWN TO 1 DO Y [J] = = cos (X[J])

```
b=100;
c='сумма=';
VAR
X, Y, Z, f: real;
d:array [1...b] of real;
{конец блока описания данных}
```

В разделе операторов блока размещаются составной оператор, который представляет собой совокупность исполняемых операторов, ограниченных словами begin и end. Операторы друг от друга отделяются точкой с запятой.

Операторы языка ПАСКАЛЬ бывают простыми и сложными.

Простые операторы не содержат внутри себя операторов. Сложные операторы включают в себя простые.

К простым относятся оператор присваивания, безусловного перехода, пустой оператор, оператор процедуры. К сложным — операторы условных переходов, цикла с предусловием, цикла с постусловием, цикла с параметром, оператор варианта и оператор присоединения. Ниже перечислены основные операторы языка ПАСКАЛЬ.

Приложение 3. Программа учебной дисциплины «Информатика и вычислительная техника для студентов инженерно-педагогических специальностей

1. Введение

Информация в современном обществе. Примеры накопления, хранения, обработки и передачи информации в учебной деятельности, на производстве, в быту.

Информатика как научная дисциплина. Основные составляющие информатики.

Значение информатики для ускорения научно-технического прогресса.

Цели и задачи изучения информатики студентами инженерно-педагогических специальностей. Содержание дисциплины «Информатика и вычислительная техника». Ее связь с другими дисциплинами специальности.

2. Основные сведения о компьютерах

Компьютер как универсальное средство обработки информации. Назначение и взаимодействие основных функциональных устройств компьютера: процессора, памяти, устройства ввода и вывода информации.

Характеристики и классификация компьютеров. Примеры конкретных моделей компьютеров.

Правила безопасности и основные санитарно-гигиенические требования при работе с компьютером.

Взаимодействие пользователя с компьютером.

3. Технология решения задач с использованием компьютера

Компьютер как инструмент для решения вычислительных задач и моделирования явлений и процессов, средство хранения и систематизации информации, управления техническими устройствами и технологическими процессами.

Этапы решения задач. Постановка задачи и построение математической или информационной модели. Алгоритмизация, программирование, отладка и тестирование программ. Анализ результатов.

Особенности решения задач с применением готовых алгоритмов и программ.

4. Алгоритмизация

Понятие алгоритма. Способы описания алгоритмов. Основные алгоритмические структуры — линейная, разветвляющаяся, циклическая. Компьютер как исполнитель алгоритмов.

Базовые типы данных. Понятие о структурах данных. Стеки. Линейные списки. Массивы.

Обработка числовой информации. Алгоритмы обработки числовых массивов. Примеры обработки числовой информации: алгоритмы ввода и вывода массивов, вычисление суммы, произведения и количества определенных элементов массива, упорядочение элементов массива.

Основные операции с символьной информацией. Примеры алгоритмов обработки символьной информации: определение количества слов в тексте, вставка и удаление слов текста, замена фрагментов текста, упорядочение символьной информации.

Машинные методы построения и преобразования плоских графических изображений: перенос, поворот, масштабирование.

Нисходящее проектирование алгоритмов. Основной и вспомогательный алгоритмы. Взаимодействие между основным и вспомогательными алгоритмами.

Проверка правильности построения алгоритмов.

5. Программирование алгоритмов

Классификация языков программирования. Общая характеристика языков программирования (БЕЙСИК, ПАСКАЛЬ или других языков).

Описание данных. Запись арифметических и логических выражений на языке программирования.

Структура программы. Операторы присваивания, ввода и вывода информации. Операторы безусловного перехода и ветвления. Операторы организации циклов.

Организация основной и вспомогательных программ. Принципы модульного программирования. Операторы для создания подпрограмм вызова.

Примеры программирования задач обработки числовой и символьной информации. Примеры обработки массивов.

Элементы машинной графики. Построение геометрических примитивов. Создание статических и динамических изображений средствами языка программирования. Преобразование графических изображений. Примеры.

Особенности программирования задач с применением готовых подпрограмм.

Классификация ошибок в программах. Виды отладки. Способы выявления и локализации ошибок. Тестирование программ.

6. Программное обеспечение компьютеров

Базовое программное обеспечение компьютера. Назначение и структура операционной системы. Примеры операционных систем. Интерпретаторы и компиляторы языков программирования. Обеспечение работы с файлами. Взаимодействие пользователя с операционной системой.

Прикладное программное обеспечение компьютера. Понятие о пакетах и комплексах прикладных программ (ППП). Примеры ППП для конкретных компьютеров.

Инструментальное программное обеспечение. Системы текстообработки. Графические редакторы. Принцип построения баз данных, система управления базами данных. Электронные таблицы. Графические пакеты программ. Автоматизированные обучающие системы.

7. Разработка педагогических программных средств

Психолого-педагогические основы применения компьютеров в учебном процессе. Функциональные возможности компьютеров в обучении.

Понятие педагогического программного средства (ППС). Классификация ППС. Выбор учебного материала для компьютеризированного обучения. Разработка сценария ППС. Организация диалога между компьютером и обучаемым. Особенности программной реализации ППС. Примеры ППС.

Методика использования компьютеров в различных видах учебной деятельности.

8. Дополнительные сведения о компьютерах

Системы счисления, применяемые в компьютере. Перевод чисел из одной системы счисления в другую. Арифметические операции с двоичными числами.

Основные сведения о представлении и преобразовании информации в компьютере.

Понятие о командах компьютера. Структура команд. Представление команд в компьютере.

Устройства компьютера. Процессор: арифметико-логическое устройство, устройство управления, основной алгоритм работы процессора. Память: виды памяти, структура памяти. Устройства ввода и вывода информации: клавиатура, дисплей, принтер, плоттер.

Локальные сети компьютеров. Принципы построения сетей. Понятие о протоколе обмена в сети.

Комплекты учебной вычислительной техники (КУВТ). Примеры и основные параметры КУВТ.

9. Информатика в современном обществе

Краткая история развития компьютерной техники поколения компьютеров, развитие элементной базы. Развитие программного обеспечения.

Использование компьютеров в системах организационного управления отраслями и предприятиями (АСУ). Управление технологическими процессами и производствами (станки с числовым программным управлением, промышленные роботы, гибкие автоматизированные производства) и проектирования (САПР). Применение компьютеров в науке, образовании и культуре.

Предоставление знаний. Логическое программирование. Понятие об экспертных системах.

Перспективы развития информационных технологий.

СПИСОК ЛИТЕРАТУРЫ

1. *Безбородов Ю. Н.* Индивидуальная отладка программ. М.: Наука, 1982.

2. *Брябрин В. М.* Программное обеспечение персональных ЭВМ. М.: Наука, 1988.

3. *Ван Тассел Д.* Стиль, разработка, эффективность, отладка и испытание программ. — М.: Мир, 1981.

4. Диалоговые системы. Современное состояние и перспективы развития/Под ред. *А. М. Довгяло.* — Киев: Наукова думка, 1987.

5. *Йодан Э.* Структурное программирование и конструирование программ. — М.: Мир, 1979.

6. *Кнут Д.* Искусство программирования для ЭВМ. Т. I. — М.: Мир, 1976.

7. *Коган Б. М.* Электронные вычислительные машины. — М.: Энергоиздат, 1985.

8. *Кушниренко А. Г., Лебедев Г. Б.* Программирование для математиков. — М.: Наука, 1988.

9. *Машбиц Е. И.* Методические рекомендации по проектированию обучающих программ. Киев: Институт психологии Министерства просвещения УССР, 1986.

10. МикроЭВМ в учебных заведениях. Кн. 8/Под ред. *Л. Н. Преснухина.* М.: Высшая школа, 1988.

11. МикроЭВМ/Под ред. *А. Дирксена.* М.: Энергоатомиздат, 1982.

12. *Пул Л.* Работа на персональном компьютере. — М.: Мир, 1986.

13. *Уолш Б.* Программирование на БЕЙСИКе. — М.: Радио и связь, 1988.

14. *Фили Дис., Ван Дем А.* Основы интерактивной машинной графики. Т. 1, Т. 2. — М.: Мир, 1985.

15. *Черемных С. В., Гиглавый А. В., Полак Ю. Е.* От микропроцессоров к персональным ЭВМ. — М.: Радио и связь. 1988.

16. Электронные вычислительные машины. Кн. 1/Под ред. *А. Я. Савельева.* — М.: Высшая школа, 1987.

СОДЕРЖАНИЕ

Учебное издание

Вьюхин Виктор Викторович
Кудымов Сергей Васильевич
Накрохин Владилен Георгиевич
Ларионов Валерий Николаевич
Мучник Владимир Лазаревич
Школьник Михаил Иосифович

ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА

Зав. редакцией *Н. И. Хрусталева.* Ведущий редактор *И. Е. Якушина.* Редактор *Л. А. Гусакова.* Художественный редактор *Т. М. Скворцова.* Художник *В. В. Гарбузов,* Технический редактор *А. К. Нестерова.* Корректор *Р. К. Косинова*

ИБ № 8838.

Изд. № СТД-725. Сдано в набор 12.07.91. Подп. в печать 20.11.91. Формат 84×108¹/₃₂. Бум. тип. 2. Гарнитура Литературная. Печать высокая. Объем 15,12 усл. печ. л. 15,33 усл. кр.-отт. 14,50 уч.-изд. л. Тираж 22 000 экз. Зак. № 817

Издательство «Высшая школа», 101430, Москва, ГСП-4, Неглинная ул., д. 29/14.

Изготовлено в книжной типографии Министерства печати и информации России

600000, г. Владимир, Октябрьский проспект, д. 7