УДК 373:002
ББК 73я72
И74

Материалы книги разработаны и подготовлены в Объединении ИнфоМир

Авторы:
А. Г. Кушниренко, А. Г. Леонов, М. Г. Эпиктетов,
В. В. Борисенко, М. А. Кузьменко,
Б. А. Назаров, С. Б. Ханжин

И74 **Информационная культура**: Кодирование информации. Информационные модели: 9—10 класс: Учеб. для общеобразоват. учеб. заведений. — 2-е изд. — М.: Дрофа, 1996. — 208 с.: ил.

ISBN 5—7107—0769—4

Новый школьный курс «Информационная культура» рассчитан на 11 учебных лет, но подключиться к его изучению можно и в 9-м классе, поскольку курс имеет образовательную, общеразвивающую направленность.

В учебнике рассматриваются способы передачи информации — компьютерные сети, электронная почта, модем; возможности использования ЭВМ в реальных ситуациях (основные понятия бухгалтерского учета и применение ЭВМ в бухгалтерии); компьютерная графика и решение задач по геометрии с помощью компьютера.

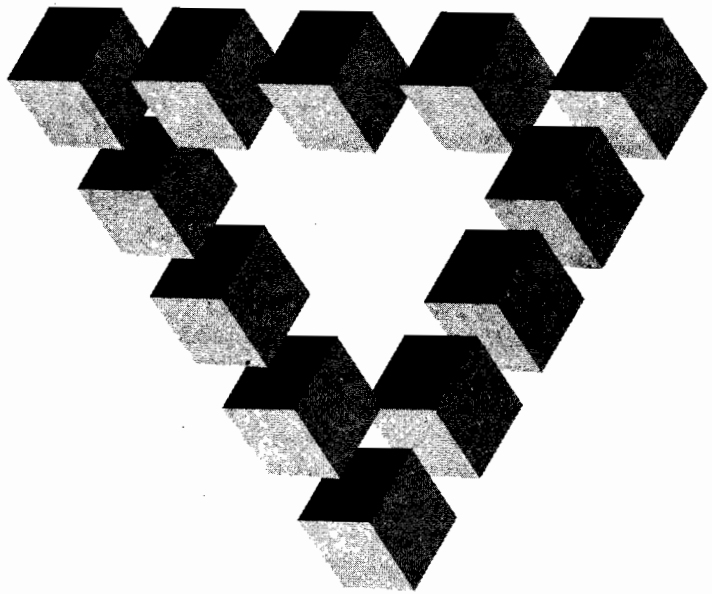
ББК 73я72

ИНФОРМАЦИОННАЯ КУЛЬТУРА

КОДИРОВАНИЕ ИНФОРМАЦИИ

КЛАСС

9



Глава 1 ПОВТОРИМ ПРОГРАММИРОВАНИЕ

Каждый должен немножечко уметь программировать

Сегодня, в конце XX века, трудно прожить, не умея складывать, вычитать и умножать хотя бы двузначные числа, не зная азов арифметики.

Точно так же в XXI веке трудно будет прожить, не умея хоть чуть-чуть программировать, не понимая основных понятий программирования.

Числа и другие понятия арифметики помогают понятнее и проще объяснять и физику, и биологию, и экономику (хотя многое можно объяснить и без чисел). Точно так же основные понятия программирования помогают понятнее и проще рассказывать о процессах обработки информации в современном обществе.

Еще несколько веков назад арифметика считалась (и была) трудной наукой, но с ее основами сегодня справляются младшеклассники. Точно так же и программирование когда-то считалось таинственным и трудным делом, а сегодня его основы доступны каждому школьнику.

Хотя эта глава называется «Повторим программирование», вы справитесь с ней и в том случае, если ни разу в жизни не программировали, а всего лишь играли в компьютерные игры или занимались на компьютере какими-то другими делами.

§ 1. ВВЕДЕНИЕ

1.1. Программа — план будущих работ компьютера

Современные словари дают, среди прочих, два значения слова *программа*:

1) план того, что должно быть сделано, выполнено;
2) закодированная информация, вводимая в компьютер для управления его деятельностью.

Первое значение этого слова было известно задолго до появления компьютеров. В те времена планы будущих действий составлялись людьми и для людей. С появлением компьютеров у человечества появились послушные «рабы», готовые неутомимо, безошибочно, быстро и в срок выполнять придуманные людьми планы. Такой план будущих работ, составленный в расчете на его выполнение компьютером, называется ***компьютерной программой*** (или просто ***программой***).

При составлении программы для компьютера важно записать ее в правильной форме, доступной компьютеру. Именно поэтому при описании второго значения слова *программа* словари подчеркивают не только назначение компьютерной программы (она нужна для работы компьютера), но и ее особую форму: программа должна быть ***закодирована***, т.е. записана в специальной форме.

1.2. Языки программирования

Набор правил, по которым закодирована компьютерная программа, называется ***алгоритмическим языком***. Алгоритмические языки возникли «искусственным путем», т.е. были придуманы людьми в последние 50 лет.

Русский, английский, французский и другие естественные языки используют разные алфавиты и разные слова для обозначения одних и тех же понятий. Например, *дерево* (рус.), *tree* (англ.) и *arbre* (фр.), или *ждать* (рус.), *wait* (англ.) и *attendre* (фр.). Точно так же и разные алгоритмические языки используют разные обозначения и конструкции для выражения одних и тех же понятий. Чтобы освоить основные понятия программирования, достаточно познакомиться с ними в каком-нибудь одном алгоритмическом языке. Мы будем использовать *школьный алгоритмический язык*, придуманный в 1985 г. академиком А. П. Ершовым (1931—1988). Этот язык кое в чем сходен с Алголом и Паскалем, но больше всего он похож на новые Бейсики, появившиеся в начале 90-х годов.

1.3. Действующие лица и исполнители
(процесса составления и выполнения программы)

Программы составляются людьми и выполняются компьютерами. При выполнении программы компьютер чаще всего не только перерабатывает информацию «внутри себя», но и командует одним или несколькими ***исполнителями*** — автоматическими устройствами, связанными с компьютером.

Обязанности между людьми, компьютерами и исполнителями делят по возможности так, чтобы каждый занимался тем, что он умеет делать лучше всего.

ЧЕЛОВЕК — гордится своей миллионолетней историей, изобретением языка, земледелия, скотоводства, собственности, денег, науки, искусства, политики, книгопечатания, телефона, телевидения и компьютеров; может понять и изобрести все на свете, включая новые языки программирования; готов составить любую программу на любом языке программирования.

КОМПЬЮТЕР — гордится тем, что за 50 лет полностью изменил жизнь $^1/_{10}$ человечества во всех областях, начиная от детских игр и кончая торговлей и финансами; берется обучиться любому языку программирования и выполнить любую программу, записанную на одном из известных языков программирования; способен неограниченно долго управлять любыми подключенными к нему исполнителями.

ИСПОЛНИТЕЛЬ — гордится тем, что у него есть раз и навсегда фиксированная и всем известная Система Команд; ничего не желает знать ни о каких программах и языках программирования и в пределах своей системы команд готов сколь угодно долго выполнять приказы Человека, Компьютера и других исполнителей; гордится своей четкостью и исполнительностью и считает делом чести отказаться исполнять невыполнимую в данный момент команду.

1.4. Язык программирования и система программирования

Теоретически язык программирования не привязан ни к какой конкретной модели компьютера, и изучать такой язык можно без всякой ЭВМ, записывая программы мелом на доске и карандашом в тетради.

На практике, однако, во много раз лучше иметь под руками компьютер, «понимающий» изучаемый язык программирования, т.е. способный выполнять составленные на нем программы.

Саму программу тоже лучше писать на компьютере — легче будет ее печатать в красивом виде, вносить исправления, сдавать на проверку учителю или давать списывать одноклассникам (со своей дискеты на дискету приятеля).

Еще удобнее, если на компьютере установлена так называемая ***система программирования*** — комплекс программ, облегчающий составление, исправление и выполнение программ на изуча-

емом языке программирования. Примерами систем программирования могут служить **Logo Writer** для языка Logo, **Turbo Pascal** и **ПаскальМир** для языка Паскаль или **КуМир** для школьного алгоритмического языка.

Мы будем использовать школьный алгоритмический язык и систему программирования КуМир на компьютерах типа IBM PC или Apple Macintosh.

Программирование становится более наглядным, если к компьютеру подключены разнообразные дополнительные устройства — исполнители, например:

- Робот — самодвижущаяся транспортная тележка;
- Чертежник — устройство рисования пером на бумаге;
- Вездеход — гусеничный вездеход с лазерным локатором;
- Двуног — шагающий робот.

Не горюйте, если ваша школа еще не подключила или не купила эти устройства. Система КуМир умеет имитировать и наглядно показывать на экране работу перечисленных выше и некоторых других устройств.

1.5. Из истории языков программирования

(Как люди и компьютеры договорились о правилах записи программ)

Первая электронная вычислительная машина ЭНИАК была построена в США в 1943—1946 гг. Вычисления на ЭНИАКе проводились электронными блоками, а вот программа работы машины задавалась вручную с помощью механических переключателей и гибких кабелей со штекерами, вставляемыми в нужные разъемы. Можно сказать, что на этой машине программы не *записывались*, а *нащелкивались* и *навтыкивались*.

Поэтому вычисления на ЭНИАКе проводились быстро и автоматически, а вот любое изменение программы было чистым мучением.

Еще до окончания постройки ЭНИАКа машиной заинтересовался видный американский математик фон Нейман и сразу предложил более прогрессивную конструкцию ЭВМ, в которой:

- в памяти ЭВМ хранятся не только числа, но и сама программа;
- и то, и другое хранится в одном и том же виде, а именно в виде многозначных двоичных чисел.

Программа для ЭВМ фон Неймана записывается на так называемом ***машинном языке***, т.е. представляет собой нелепую с точки

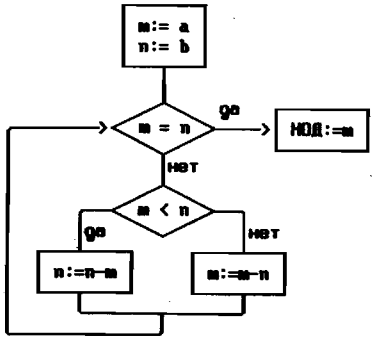


Рис. 1. Блок-схема программы вычисления НОД(a,b)

зрения человека последовательность двоичных чисел. Затем программа *перфорируется* человеком на бумажной ленте, которая вводится в память машины автоматически, без участия человека.

Придумывать и записывать программу сразу на машинном языке неудобно, поэтому фон Нейман предложил на начальном этапе разработки программы использовать более наглядную графическую форму записи — ***блок-схемы*** (рис. 1).

Но и блок-схемы не позволяют обойтись без неудобного для человека машинного языка. Рано или поздно блок-схему приходится вручную переводить на машинный язык. А эта нудная долгая работа не обходится без ошибок, которые потом приходится искать и исправлять.

Поэтому возникла идея записывать программу на так называемом ***языке программиста***, а перевод с языка программиста на язык ЭВМ поручить самой ЭВМ.

В течение буквально нескольких лет (к середине 50-х годов) все эти идеи были воплощены в жизнь разными коллективами математиков и программистов в разных странах. Одни использовали блок-схемы и переводили их на машинный язык вручную. Другие придумывали различные языки программиста, третьи разрабатывали ***компиляторы*** — программы автоматического перевода с языков программиста на машинные языки. Все это тут же проходило проверку на практике и к концу 50-х годов наибольшую популярность завоевали языки Алгол и Фортран. Термин «язык программиста» не прижился и вместо него стали говорить ***алгоритмический язык***, т.е. язык для записи алгоритмов.

Практика показала, что удобнее всего программировать на алгоритмических языках. А программирование на машинных языках и блок-схемы хотя и продолжают использоваться, но имеют ограниченное применение.

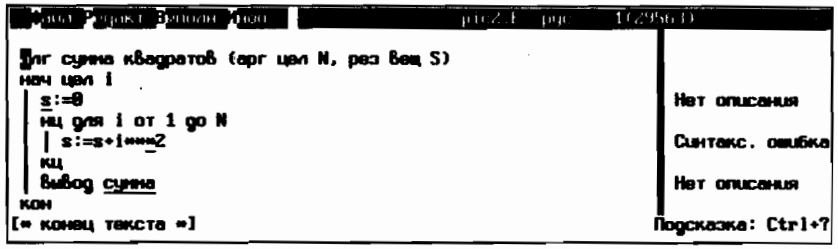


Рис. 2. Пример диагностики на полях КуМира

Основные понятия программирования, которые будут затронуты ниже, появились уже в первых алгоритмических языках Алгол и Фортран и перекочевали позже в большинство современных алгоритмических языков.

1.6. Язык и система программирования КуМир-Гипертекст

В 1985 г. академик А.П. Ершов придумал похожий на Алгол ***школьный алгоритмический язык***. Для повторения основных понятий будет использоваться этот язык и система программирования КуМир, разработанная на механико-математическом факультете МГУ. Текущая версия КуМира доработана предприятием ИнфоМир и используется во многих школах России (а также на мехмате МГУ).

Подобно тому, как современные текстовые редакторы проверяют грамматическую правильность набираемого текста, система КуМир проверяет правильность набираемой на компьютере программы (рис. 2), а также сообщает об ошибках при выполнении программы (рис. 3). Кроме того, система КуМир позволяет работать с гипертекстами (рис. 4).

1.7. Что такое гипертекст

Принципы гипертекста заложены в большинство современных компьютерных программ, и навыки работы с гипертекстами становятся обязательным элементом информационной культуры.

Гипертекст представляет собой обычный текст с картинками, в котором выделены (чаще всего цветом) отдельные знаки, слова, фразы или картинки, называемые ***полями***. С каждым полем в гипертексте связано определенное действие, например:

- переход в другое место того же гипертекста;
- переход в другой гипертекст;

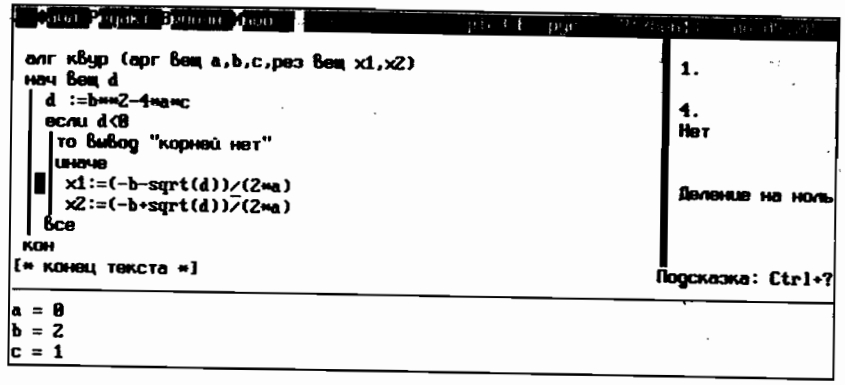


Рис. 3. Диагностика во время выполнения

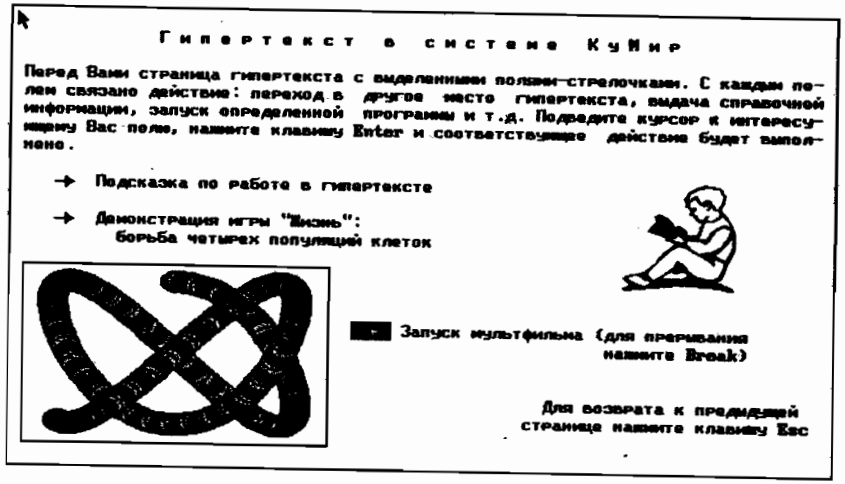


Рис. 4. Фрагмент гипертекста в КуМире

- проигрывание фрагмента звукозаписи или видеозаписи;
- запуск на выполнение определенной программы и т.д.

Если при чтении гипертекста установить курсор на одно из полей и нажать клавишу Enter, то выполнится действие, связанное с выбранным полем.

Загрузите гипертекст к главе 1 и вы сможете прочесть на экране дополнительный материал, увидите картинки, мультфильмы, демонстрации работы в системе КуМир, сможете ответить на контрольные вопросы. Более того, вы сможете решать задачи на составление алгоритмов, получая при необходимости подсказку или проверку вашего решения.

§ 2. ОСНОВНЫЕ КОНСТРУКЦИИ И ПОНЯТИЯ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ

На каждую компьютерную программу можно посмотреть с двух разных сторон. С одной стороны, это некоторый *текст*, составленный по определенным правилам из русских и латинских букв, цифр, знаков арифметических операций, знаков препинания, скобок и некоторых других специальных знаков. С другой стороны, программу можно *загрузить* в память компьютера и *выполнить*, заставив компьютер достичь целей, предусмотренных автором программы.

Программа составляется из отдельных фрагментов, называемых командами, операторами или конструкциями. Чтобы понять язык программирования, нужно для каждой команды разобраться:

1) как правильно *записать* команду;
2) как компьютер *выполняет* команду.

В наши дни изучение правил записи команд перестало быть проблемой. Подобно тому, как при работе на пишущей машинке не нужно помнить в деталях, как пишутся отдельные буквы — машинка их печатает сама целиком, — при написании программы на компьютере не обязательно помнить в деталях правила записи команд языка. По нажатии на нужную кнопку или выборе нужной строки меню компьютер сам вставит в правильном порядке все обязательные части выбранной команды. Более того, если в процессе написания программы компьютер обнаружит какие-то ошибочные или подозрительные места, то они немедленно будут отмечены в тексте программы.

2.1. Исполнитель Робот

ЭВМ могут не только вычислять, но и командовать различными устройствами — исполнителями. Исполнитель может быть связан с ЭВМ проводами, по радио, инфракрасными лучами, ультразвуком или еще каким-нибудь способом. Сейчас это совершенно не важно. А важно то, что каждый исполнитель умеет выполнять какой-то набор команд и эти команды можно записывать в выбранном языке программирования.

В КуМире в такой команде можно использовать большие и малые русские и латинские буквы и цифры, а также пробел. В других языках правила записи могут немного отличаться. Например, в Паскале пробел в команде не разрешается (вместо него можно использовать подчеркивание).

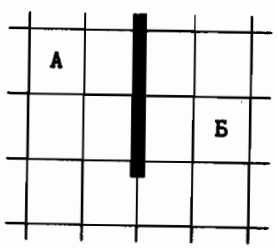
Исполнитель Робот умеет выполнять 17 команд, но нас пока интересуют 5 из них:

вверх	вниз	
вправо	влево	закрасить

Если вы уже дочитали учебник до этого места, самое время подойти к компьютеру. Загрузите гипертекст к главе 1 и выберите страницу «Знакомство с исполнителем Робот». За 5 минут вы посмотрите демонстрацию команд Робота и потренируетесь в ручном управлении Роботом.

2.2. Простейшая программа
(ЭВМ управляет Роботом)

Предположим, что Робот находится в клетке А и нужно перевести его в клетку Б:



Вы без труда решите эту задачу, управляя Роботом вручную. Перевести Робота в клетку Б можно многими разными способами. Например, так. Дадим Роботу две команды «вниз», потом три команды «вправо», наконец, одну команду «вверх». Это решение легко превращается в программу:

```
алг Переход в точку Б
нач
  вниз
  вниз
  вправо
  вправо
  вправо
  вверх
кон
```

В этой программе использована одна ***конструкция*** «алгоритм»:

```
алг «имя алгоритма»
нач
| ...
кон
```

и шесть так называемых ***команд вызова*** Робота: две команды вниз, три команды вправо и одна команда вверх.

Итак,

> в КуМире есть конструкция алг-нач-кон, позволяющая собрать вместе несколько последовательно выполняемых команд и назвать эту последовательность одним именем.

Подобная возможность есть и в других языках. Например, на языках Паскаль, Лого и Фортран приведенная выше программа могла бы выглядеть так:

Паскаль:

```
procedure to_pt_B;
begin
  down; down; right; right; right; up
end
```

Лого:

```
ЭТО В_ТОЧКУ_В
   ВНИЗ
   ВНИЗ
   ВОСТОК
   ВОСТОК
   ВОСТОК
   СЕВЕР
КОНЕЦ
```

Фортран:

```
      SUBROUTINE TOPTB
      CALL DOWN
      CALL DOWN
      CALL RIGHT
      CALL RIGHT
      CALL RIGHT
      CALL UP
      RETURN
      END
```

В КуМире в начало алгоритма обычно включаются еще так называемые ***команды контроля*** «дано» и «надо», на которых мы не будем останавливаться.

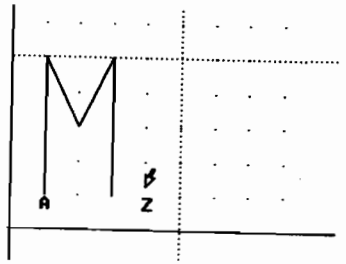
2.3. Вспомогательные алгоритмы

Начнем этот раздел с исполнителя Чертежник. Чертежник умеет рисовать пером на листе бумаги. Перо можно поднимать, опускать и двигать. При движении опущенного пера за ним остается след. Чертежник чуть-чуть сложнее Робота: в командах движения пера нужно задать информацию о том, куда его сдвинуть.

Запустите гипертекст к этой главе и на странице «Знакомство с исполнителем Чертежник» вы увидите примеры работы Чертежника.

Вот пример программы управления Чертежником:

```
алг буква М
 дано |перо в точке A и поднято
 надо |нарисована буква М размером 2 х 4,
      |перо в точке Z и поднято (см.рисунок)
нач
| опустить перо
| сместиться на вектор(0, 4)
| сместиться на вектор(1,-2)
| сместиться на вектор(1, 2)
| сместиться на вектор(0,-4)
| поднять перо
| сместиться на вектор(1, 0)
кон
```



Этот алгоритм есть не что иное, как указание компьютеру нарисовать букву М и оставить поднятое перо в начале следующей буквы.

А теперь представим себе, что нужно нарисовать слово МММ. Первый способ сделать это — написать новый алгоритм, в котором будет втрое больше команд:

```
алг слово МММ
 дано |перо поднято
 надо |нарисовано слово МММ, перо поднято
нач
| опустить перо
| сместиться на вектор(0, 4)
| сместиться на вектор(1,-2)
| сместиться на вектор(1, 2)
| сместиться на вектор(0,-4)
| поднять перо
| сместиться на вектор(1, 0)
| опустить перо
| сместиться на вектор(0, 4)
| сместиться на вектор(1,-2)
```

```
    сместиться на вектор(1, 2)
    сместиться на вектор(0,-4)
    поднять перо
    сместиться на вектор(1, 0)

    опустить перо
    сместиться на вектор(0, 4)
    сместиться на вектор(1,-2)
    сместиться на вектор(1, 2)
    сместиться на вектор(0,-4)
    поднять перо
    сместиться на вектор(1, 0)
кон
```

Но есть и другой, более удобный способ:

```
алг МММ
 дано |перо поднято
 надо |нарисовано слово МММ, перо поднято
нач
| буква М
| буква М
| буква М
кон
```

Алгоритм слово МММ использовал только команды Чертежника. А вот алгоритм МММ команды Чертежника в явном виде не использует, так как выполняется с помощью ранее написанного алгоритма буква М. Поэтому алгоритм буква М называют ***вспомогательным*** для ***основного*** алгоритма МММ. При выполнении основного алгоритма МММ компьютер трижды выполняет вспомогательный алгоритм буква М.

В принципе любой алгоритм, написанный с помощью вспомогательных, можно переписать «одним куском», без всяких вспомогательных алгоритмов. Однако применение вспомогательных алгоритмов обычно упрощает написание алгоритма и делает алгоритм более понятным.

Выберите в гипертексте страницу «Вспомогательные алгоритмы» и посмотрите несколько примеров выполнения программ со вспомогательными алгоритмами.

2.4. Аргументы вспомогательных алгоритмов

Одной и той же дрелью можно сверлить дырки разного размера, меняя сверла. Точно так же один и тот же вспомогательный

алгоритм может выполнять разные работы при смене так называемых ***аргументов***.

Вот пример алгоритма с одним аргументом:

```
алг квадрат (арг вещ a)
 дано | перо Чертежника в левом нижнем углу
      | будущего квадрата и поднято
 надо | нарисован квадрат с длиной стороны a, перо
      | Чертежника в исходной точке и поднято
нач
| опустить перо
| сместиться на вектор( a, 0)
| сместиться на вектор( 0, a)
| сместиться на вектор(-a, 0)
| сместиться на вектор( 0,-a)
| поднять перо
кон
```

Задавая перед выполнением алгоритма разные числовые значения аргумента a, можно получать квадраты разных размеров.

Запустите гипертекст и посмотрите, как выполняется этот алгоритм.

2.5. Цикл «N раз» — самая простая составная команда

В любом языке программирования есть так называемые ***составные команды***. Такая команда может содержать внутри себя другие команды, в том числе и составные. В свою очередь, эти составные команды тоже могут содержать другие команды, то есть команды могут вкладываться друг в друга подобно матрешкам.

Самая простая составная команда — это ***цикл «N раз»***. Она позволяет кратко записать длинную последовательность одинаковых команд.

Запустите гипертекст и посмотрите, как эта команда работает на примере алгоритма аллея.

```
алг аллея
 дано | перо Чертежника в начале будущей аллеи
 надо | нарисована аллея из 6 елочек, перо в начале
      | следующей аллеи
нач
| сместиться в точку(2,11)
```

```
  нц 6 раз
  | елочка
  кц
  сместиться на вектор(-18,-5)
кон
алг елочка
 дано | перо Чертежника в нижней точке будущей
      | елочки
 надо | нарисована елочка, перо в нижней точке
      | следующей елочки и поднято
нач
  сместиться на вектор(0,3); опустить перо
  нц 4 раз
  | сместиться на вектор(-1,-1); поднять перо
  | сместиться на вектор( 2, 0); опустить перо
  | сместиться на вектор(-1, 1)
  | сместиться на вектор( 0,-0.5)
  кц
  сместиться на вектор(0,-1); поднять перо
  сместиться на вектор(3,0)
кон
```

Команда

```
нц 6 раз
| елочка
кц
```

заставляет компьютер 6 раз подряд выполнить один и тот же алгоритм елочка. Перед каждым новым выполнением алгоритма елочка перо Чертежника будет на 3 единицы правее предыдущего положения. Вот и получится аллея из 6 елочек, расположенных в ряд (рис. 5).

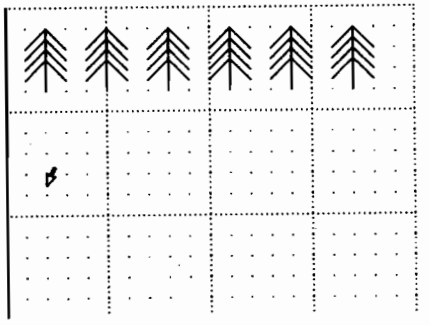


Рис. 5. Аллея из 6 елочек

2.6. Обратная связь при управлении исполнителями

При управлении исполнителями ими можно не только командовать, но и получать от них информацию, пользуясь так называемыми командами ***обратной связи***. Разберем это на примере Робота. Кроме пяти команд управления, у Робота есть еще команды

обратной связи, при выполнении которых Робот сообщает информацию об обстановке вокруг себя:

```
лог сверху свободно
лог снизу свободно
лог справа свободно
лог слева свободно
лог клетка закрашена
```

Служебное слово лог показывает, что это команды ***логические***. В ответ на логическую команду Робот отвечает да или нет.

Пользуясь командами обратной связи, можно написать программу, которая будет успешно решать задачу в разных обстановках.

Выберите в гипертексте страницу «Команды обратной связи у Робота» и посмотрите, как работают эти команды.

2.7. Команда повторения «цикл "пока"»

(Обратная связь при выполнении программы)

Выполнение всех разобранных нами до сих пор программ можно было предсказать во всех деталях, глядя на их текст и ничего не выполняя. Но выполнение программы может и не быть таким прямолинейным, может зависеть от информации, которую компьютер накопил или получил извне в процессе выполнения программы.

Для использования такой информации в ходе выполнения программы во всех языках программирования есть специальные команды ветвления и повторения.

Самая интересная и важная из них — это ***цикл «пока»***. Вот пример записи этой команды:

```
нц пока снизу свободно
| вниз
кц
```

Выберите в гипертексте страницу «Цикл "пока"» и посмотрите, как происходят:

- *диалог «ЭВМ-Робот» при выполнении цикла «пока»;*
- *закрашивание клеток в цикле «пока»;*
- *поиск закрашенной клетки в цикле «пока».*

Хотя внешне цикл «пока» выглядит совсем просто, выполнение его компьютером имеет ряд особенностей:

- цикл может ни разу не выполниться;
- цикл может работать до бесконечности;
- условие цикла не проверяется «в середине цикла».

На той же странице гипертекста можно наглядно ознакомиться с этими особенностями.

2.8. Команда ветвления «если»

Команда ветвления «если» позволяет при выполнении программы выполнять или пропускать некоторые действия или выбирать один из двух вариантов выполнения программы.

Например, пусть клетка, в которой стоит Робот, должна быть закрашена. Если клетка не закрашена, то ЭВМ должна скомандовать Роботу «закрасить клетку». Если же клетка уже закрашена, то ничего делать не нужно. Записывается это так:

```
если клетка не закрашена
|то закрасить
все
```

Команда «если» может входить в другие составные команды

```
нц 10 раз
| вправо
| если клетка не закрашена
| |то закрасить
| все
кц
```

или содержать внутри себя другие составные команды

```
если снизу свободно
|то
| нц пока снизу свободно
| | вниз
| кц
|иначе
| нц пока сверху свободно
| | вверх
| кц
все
```


Выберите в гипертексте страницу «Команда ветвления "если"» и ознакомьтесь с примерами выполнения команды «если».

2.9. Величины и их типы

При составлении программ надо уметь запоминать, изменять и использовать информацию в памяти ЭВМ. Для этого в языках программирования используются так называемые *переменные величины*. В КуМире они называются просто ***величины***.

Термин «величина» заимствован из математики и физики, поскольку в алгоритмах для решения математических или физических задач величины алгоритмического языка соответствуют математическим или физическим величинам.

Имя, значение и тип величины

Каждая величина имеет имя, значение и тип. ***Имя*** величины (например, n, x, d) служит для обозначения величины в алгоритме. Во время выполнения алгоритма в каждый конкретный момент величина имеет какое-то ***значение*** (например, 22 или −107) либо *не определена*. Если значением величины может быть только целое число, то величина называется *целой* или *целочисленной*, если любое вещественное число — то *вещественной*. Эта характеристика величины (в данном случае является ли величина целой или вещественной) называется ***типом*** величины. Обычно кроме числовых величин в языке программирования есть величины и других типов: логические, литерные и другие.

Для того чтобы запомнить или изменить значение величины, в КуМире есть специальная команда — ***команда присваивания*** (:=). Например, после выполнения команды

```
n:=0
```

значение n станет нулевым, а после выполнения команды

```
n:=n+1
```

значение n увеличится на 1.

Арифметические операции обозначаются в КуМире так:

сложение	x+y
вычитание	x-y
умножение	x*y

деление	x/y	
возведение в степень	x**y	
частное от деления нацело	div(x,y)	(x и y — целые)
остаток от деления нацело	mod(x,y)	(x и y — целые)

Описание величин

Для того чтобы ЭВМ могла работать с величиной, нужно указать тип и имя величины, например, цел n. Такое указание называется ***описанием*** величины. Хотя в некоторых языках описания величин могут быть опущены или вообще не требуются, в большинстве языков каждая величина должна быть описана. Вот пять наиболее распространенных типов величин:

Тип величины	КуМир	Паскаль	Си
целая	цел	Integer	int
вещественная	вещ	Real	float
логическая	лог	Boolean	–
символьная	сим	Char	char
литерная	лит	String	char[]

Правила работы с величиной зависят от того, где и как она описана. В КуМире величина может быть описана внутри алгоритма после слова нач:

```
нач цел n
| ...
```

Такая величина называется ***промежуточной***. Ее можно использовать только внутри данного алгоритма. В Паскале и Си подобные величины называются *локальными*. Промежуточные (локальные) величины могут иметь одинаковые имена, если они используются в разных алгоритмах.

Выберите в гипертексте страницу «Величины в алгоритмическом языке» и посмотрите, как величины могут использоваться при управлении Роботом.

Целью выполнения программы может быть не только *управление* каким-то исполнителем, но и *вычисление* какой-то информации, нужной непосредственно человеку или другой программе.

Например, программа может вычислять, как будет расти вклад в банке при заданном месячном проценте.


```
алг сумма вклада
нач вещ s       | текущая сумма на счету
| вещ процент | рост вклада за месяц в процентах
| процент:=20
| s:=1          | начальный вклад 1 млн.руб.
| нц 12 раз
| | s:=s+s*процент/100
| кц
| вывод нс,"Через год сумма будет",s,"млн.руб."
кон
```

В этой программе величины s, процент имеют тип вещ, так как могут принимать и целые, и дробные значения.

Выберите в гипертексте страницу «Величины в алгоритмическом языке» и посмотрите, как программа «Рост вклада» вычисляет сумму вклада и показывает графически рост вклада в течение года.

2.10. Почему нельзя обойтись без табличных величин

Современные компьютеры способны оперировать громадными объемами информации: миллионами и миллиардами чисел, букв и других символов. Придумать отдельное имя для каждого числа, каждого символа и т.д. совершенно невозможно. Поэтому элементарные порции информации нужно как-то группировать, объединять. Простейший и наиболее эффективный способ такой группировки — ***таблицы***. Табличная величина имеет имя и состоит из нескольких элементов. Элементы нумеруются целыми числами, чаще всего начиная с единицы. Например, таблица

```
цел таб n[1:1000]
```

содержит 1000 целых чисел, занумерованных от 1 до 1000, а таблица

```
вещ таб v[0:1000]
```

содержит 1001 вещественное число.

Для работы с элементом таблицы нужно указать имя таблицы и номер элемента в ней (этот номер называется ***индексом*** элемента). Например, первый элемент таблицы n обозначается как n[1]. Если целая величина i меняет свои значения от 1 до 1000, то n[i] поочередно обозначает все элементы таблицы n.

Элементы таблицы могут иметь двойную или даже тройную нумерацию:

```
вещ таб цена[1:100,1:20],
цел таб цвет[0:30,0:30,0:30]
```

При выполнении программы информация может как записываться в таблицу, так и считываться из нее. Для перебора элементов таблицы удобна команда ***цикл «для»***. Например, заполнение таблицы `вещ таб k[1:10]` нулями можно записать так:

```
нц для i от 1 до 10
| k[i]:=0
кц
```

В этом цикле команда `k[i]:=0` будет выполнена 10 раз: первый раз для `i` = 1, второй — для `i` = 2, ..., последний — для `i` = 10.

Выберите в гипертексте страницу «Табличные величины и цикл "для"» и посмотрите, как работают с таблицами простейшие алгоритмы.

2.11. Литерные величины

Для работы с текстовой информацией в компьютере используются ***литерные величины***. Литерные значения записываются в виде строки символов, заключенных в кавычки, например `"информатика"`. Запись `лит t` описывает одну литерную величину `t`. Значение `t` можно задать командой присваивания, например

```
t:="информатика"
```

Для обозначения `i`-го элемента литерной величины используется запись `t[i]`. Например, первый символ строки `t` обозначается `t[1]`, второй — `t[2]` и т.д. Таким образом, в этом примере `t[1]="и"`, `t[2]="н"` и т.д.

Число символов в строке `t` называется ее ***длиной***. Например, длина строки `"мама"` равна 4, а длина строки `"Нет выхода!"` равна 11. После выполнения приведенной выше команды присваивания длина строки `t` — она обозначается `длин(t)` — будет равна 11.

Две строки можно соединить в одну с помощью операции соединения. Она обозначается знаком `+`. Например, если `t="Ма"`, то `t+t` будет равно `"МаМа"`, а `t+"ма"` будет равно `"Мама"`.

Литерные величины и десятичные числа

Многозначное целое число может быть изображено в виде строки цифр. Например,`t="1234567890"` изображает десятизначное число. Первая цифра этого числа равна символу `"1"`, последняя — символу `"0"`.

Следует отличать операции над числами и соответствующими строками цифр. Например, `10+1=11`, но `"10"+"1"="101"`, а вовсе не `"11"`. Десятичное изображение числа `n` в виде строки цифр обозначается `лит(n)`: `лит(11)="11"`, `лит(0)="0"`, `лит(-11)="-11"`.

Литерные величины будут использованы ниже в алгоритмах перевода чисел из двоичной системы в десятичную, и обратно.

Выберите в гипертексте страницу «Литерные величины» и ответьте на контрольные вопросы.

Глава 2 КОДИРОВАНИЕ ИНФОРМАЦИИ БЕЗ КОМПЬЮТЕРОВ

§ 3. КОДИРОВАНИЕ ЧИСЕЛ

3.1. Что такое кодирование

Кодирование информации — это просто-напросто представление сведений в той или иной стандартной форме. Одни и те же сведения могут быть представлены, закодированы в нескольких разных формах. Совершенно разные сведения могут быть представлены в похожей форме.

С появлением компьютеров возникла необходимость кодирования (т.е. представления в формальном, стандартизованном виде) всех видов информации, с которыми имеет дело и отдельный человек и человечество в целом. Но решать задачу кодирования информации человечество начало задолго до появления компьютеров.

Грандиозные достижения человечества — письменность и арифметика — есть не что иное, как системы кодирования речи и числовой информации.

Дарвин считал, что обезьяну человеком сделал труд. Другие ученые считают, что человек стал Человеком благодаря своим успехам в кодировании информации, благодаря изобретению языка, письменности и способов кодирования и записи числовой информации.

3.2. Как считали в далеком прошлом

Считать могут не только люди. Установлено, что считать до трех могут и птицы (и многие другие животные). Если у птицы забрать из гнезда одно яйцо из пяти, то она не заметит пропажи, а вот если забрать одно из трех, то птица начинает проявлять беспокойство. Но от такого элементарного счета до понятия ***число*** еще очень и очень далеко.

Первобытные люди не знали чисел и использовали наглядное представление информации для запоминания того или иного количества предметов. Например, чтобы запомнить, что на охоте

было убито пять оленей, пещерный «летописец» просто рисовал их всех на стене пещеры.

3.3. Как считали в прошлом

Способы кодирования числовой информации — способы счета и представления чисел — в истории человечества последовательно менялись. Следы древних систем счета и представления чисел встречаются и сегодня в культуре и обычаях многих народов. К древнему Вавилону восходит деление часа на 60 минут и угла на 360 градусов. К древнему Риму восходит традиция записывать в ***римской записи*** небольшие числа: I, II, III, IV, V, VI, VII, VIII, IX, X, XI, XII, ... Например, часто пишут «XIX век», «XX век» вместо «19 век» и «20 век». К англосаксам — жителям Британских островов — восходит традиция счета дюжинами: в году 12 месяцев, в футе 12 дюймов, сутки делятся на два периода по 12 часов.

Много информации о распространенных в прошлом единицах счета до сих пор сохранилось в современных языках. Особую роль числа 40 при счете помнит русский язык: сохранилось выражение «сорок сороков», да и само слово *сорок* выбивается из основанного на десятке ряда числительных: двадцать, тридцать, пятьдесят, шестьдесят, семьдесят, восемьдесят. Во французском языке сохранились следы счета двадцатками: если в русском слово *восемьдесят* означает «восемью десять» то французское числительное *quatre-vingts* может быть переведено как «четырежды двадцать».

3.4. Возникновение понятия числа

Историкам не известно, когда появились понятия числа и количества, но они склонны считать, что это произошло на самых ранних стадиях развития нашей цивилизации. Началось все с подсчетов одинаковых конкретных предметов: пять ножей, пять оленей, пять деревьев. Затем, мало-помалу, понятие количества предметов — пять — стало отделяться от того, какие именно предметы считаются. Это и привело к возникновению понятия ***число***.

Следы того, как люди в глубокой древности обходились без общего понятия *число*, можно найти в языках и доныне живущих на земле примитивных племен. У них в языке сохранились различные числительные для различных предметов. Одни числительные используются для подсчета людей, другие — для подсчета круглых предметов, третьи — для продолговатых и т.д. На-

пример, в языке народа чишмиенов (Канада, провинция Британская Колумбия) есть целых семь видов числительных для подсчета предметов разной природы.

Да и в русском языке для счета одушевленных предметов есть особые формы числительных. Вспомните: «Один с сошкой, *семеро* с ложкой».

3.5. Как называть и записывать числа

Чтобы использовать числа, нужно их как-то называть и записывать, нужна какая-то система записи чисел — нужна ***система нумерации***.

Разные народы в разные времена использовали разные системы нумерации.

Еще недавно существовали племена, в языке которых было всего два числительных: «один» и «два». Это, конечно, не означало, что представители этих племен не могли сосчитать три предмета. Большие числа представлялись комбинациями. Так, например, у туземцев островов, расположенных в Торресовом проливе, было всего два числительных: *урапун* (один) и *окоза* (два). Большие числа назывались так: *окоза-урапун* (три), *окоза-окоза-урапун* (пять) и т.д. Правда, эта нумерация не успевала стать слишком громоздкой. Числа, начиная с семи, имели единственное обозначение — *много*.

Наверное, на первых порах счета до 7 и хватало для повседневных нужд, но жизнь не стояла на месте. Появлялись новые способы охоты, развивалось земледелие, и такого маленького запаса чисел начинало не хватать. Возникало желание увеличить верхнюю границу счета. Считать стали при помощи пальцев рук, ног и других частей тела. Например, те же островитяне для счета употребляли локти, запястья, плечи.

Так прокладывались пути к счету пятерками (пальцы одной руки), двадцатками (пальцы рук и ног) и десятками — счету предметов с помощью пальцев двух рук. Из счета десятками и выросла современная десятичная система.

3.6. Как считали древние египтяне

По современным данным, развитые системы нумерации впервые появились в Древнем Египте и Месопотамии. Мы многое знаем о египетской системе. До нас дошли надписи внутри пирамид, на плитах и обелисках. Эти надписи сделаны в виде картинок-иероглифов, и такой способ письма (кодирования речи) вообще

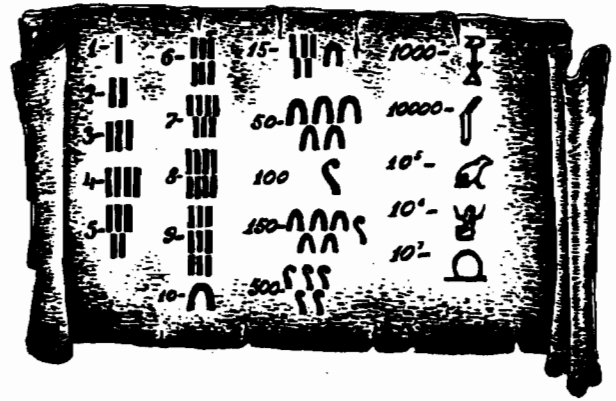


Рис. 6

характерен для ранних стадий развития человечества. Многие надписи сейчас уже прочитаны и расшифрованы. Сохранились также два математических папируса, позволяющих узнать об арифметике древних египтян.

Для записи чисел египтяне применяли иероглифы *один*, *десять*, *сто*, ..., *десять миллионов*. Все остальные числа записывались с помощью этих иероглифов и операции сложения (рис. 6). Так что в египетской записи чисел особую роль играли десятка и ее степени: 10, 100, 1000 и т.д.

Делили и умножали египтяне совсем не так, как мы. Особую роль у египтян играло число ***два*** и его степени: 2, 4, 8, 16 и т.д. Умножение и деление проводилось путем последовательного удвоения чисел.

Пусть, например, надо умножить 19 на 94. Египтяне последовательно удваивали число 94, причем в правом столбце записывали результаты удвоения, а в левом — соответствующие степени двойки. (Разумеется, записывали они это по-своему, но ниже вычисления показаны в современной записи. Суть дела от этого не меняется.)

1	94
2	188
4	376
8	752
16	1504

Удвоение продолжалось до тех пор, пока не оказывалось, что из чисел левого столбца можно составить множитель (в нашем случае $19 = 1 + 2 + 16$). Египтяне отмечали соответствующие строки вертикальными черточками и складывали те числа, которые стоят в этих же строках справа. В приведенном примере сложение трех чисел $94 + 188 + 1504$ дает искомое произведение 1786.

Деление египтяне проводили удвоением делителя. Пусть, например, требуется разделить 1786 на 19. В этом случае египтяне последовательно удваивали делитель и продолжали до тех пор, пока числа правого столбца оставались меньше делимого:

1	19
2	38
4	76
8	152
16	304
32	608
64	1216

Затем из чисел правого столбца они пытались составить делимое, и если это удавалось, то сумма чисел в левом столбце давала частное. В рассмотренном случае делимое 1786 можно составить как 1216+304+152+76+38, значит, частное будет 64+16+8+4+2 = 94.

Если бы делимое не делилось без остатка на делитель, из чисел правого столбца удалось бы составить лишь число, меньшее делимого. Так получились бы и частное и остаток.

Один из недостатков египетской системы — невозможность записи больших чисел без придумывания новых иероглифов. Иероглиф для десяти миллионов еще есть, а вот для ста миллионов уже нет. Этот недостаток преодолен в позиционных системах счисления (см. пункт 3.9).

Упражнения:

1. Умножить 13 на 30 с помощью удвоений.
2. Разделить с остатком 340 на 13 с помощью удвоений.

3.7. Алфавитные системы нумерации

Другой недостаток египетской системы — громоздкая запись чисел. Для записи числа девять египтяне девять раз повторяли иероглиф для единицы. Этого недостатка лишены алфавитные системы записи чисел, принятые в свое время у ионийцев, древних евреев, финикийцев, армян, грузин, а также и у славян.

Славянская алфавитная нумерация напоминала современную позиционную. В ней числа были закодированы буквами, а над этими буквами, чтобы избежать путаницы, ставился специальный знак — *титло*:

1 = $\tilde{\text{А}}$, 2 = $\tilde{\text{В}}$, 3 = $\tilde{\Gamma}$ и т.д.

Одной буквой кодировались числа от 1 до 9, затем 10, 20, ..., 90, и, наконец, 100, 200, ..., 900.

Для больших чисел использовались те же самые буквы с добавленными к ним специальными значками, например 10 000 обозначалось как

$\tilde{\text{а}}$

3.8. Римская система счисления

В римской системе счисления семь чисел обозначаются буквами:

1 – I	5 – V	10 – X	50 – L
100 – C	500 – D	1000 – M	

а остальные числа записываются комбинациями этих букв. Если в комбинации буквы идут в порядке от больших к меньшим, то соответствующие числа складываются. Например:

XXVII означает 10 + 10 + 5 + 1 + 1 = 27,
MMMD означает 1000 + 1000 + 1000 + 500 = 3500.

Если же какие-то буквы нарушают порядок, то их значения вычитаются из значения следующей буквы. Например:

IV означает 5 – 1 = 4,
XIX означает 10 + (10 – 1) = 19,
MCMXCIV означает 1000 + (1000 – 100) + (100 – 10)+ +(5 – 1) = 1994.

Если складывать и вычитать в такой системе еще можно без особого труда, то умножать очень сложно, а деление представляет собой почти непосильную проблему.

Вместе с тем в римской системе счисления есть одна важная идея: вклад буквы в число зависит не только от самой буквы, но и от порядка следования (позиции) букв в записи числа. Так, например, буква I дает вклад +1 в число VI и вклад –1 в число IV. Развитие этой идеи приводит к современным ***позиционным*** системам нумерации.

Упражнение:

3. Сложите числа MXLVII и MCMLIII.

3.9. Десятичная позиционная система счисления

Различные системы счета и записи чисел тысячелетиями сосуществовали и соревновались между собой, но к концу «докомпьютерной эпохи» особую роль при счете стало играть число *десять*,

а самой популярной системой кодирования чисел оказалась так называемая ***позиционная десятичная система***. В этой системе значение цифры в числе зависит от ее *места* (*позиции*) внутри числа.

Знаменитый французский математик и физик Лаплас сказал:

«Мысль выражать все числа десятью знаками, придавая им, кроме значения по форме, еще и значение по месту, настолько проста, что именно из-за этой простоты трудно понять, насколько она удивительна».

Десятичная система счисления пришла из Индии, где она появилась не позднее VI в. н.э.

В десятичной системе всего десять цифр: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, но информацию несет не только цифра, но также и место, на котором она стоит. В числе 444, например, три одинаковые цифры обозначают количество и единиц, и десятков, и сотен. А вот в числе 400 первая цифра 4 обозначает число сотен, а две цифры 0 сами по себе вклад в число не дают, а нужны лишь для указания позиции цифры 4.

Итак, в десятичной позиционной системе счисления особую роль играет число десять и его степени: 10, 100, 1000, 10 000, ... Самая правая цифра числа показывает число единиц, следующая цифра — число десятков, следующая — число сотен и т.д. Например, число 1995 составляют: 5 единиц, 9 десятков, 9 сотен и одна тысяча:

$$1995 = 5 + 9 \cdot 10 + 9 \cdot 100 + 1 \cdot 1000.$$

Поскольку $1000 = 10^3, 100 = 10^2, 10 = 10^1$, можно написать еще и так

$$1995 = 5 + 9 \cdot 10^1 + 9 \cdot 10^2 + 1 \cdot 10^3.$$

Для единообразия можно еще заменить 5 на $5 \cdot 10^0$ и изменить порядок слагаемых:

$$1995 = 1 \cdot 10^3 + 9 \cdot 10^2 + 9 \cdot 10^1 + 5 \cdot 10^0.$$

Отсюда уже ясно, как записать любое четырехзначное число:

$$N = a_3 \cdot 10^3 + a_2 \cdot 10^2 + a_1 \cdot 10^1 + a_0 \cdot 10^0,$$

где a_0, a_1, a_2, a_3 — десятичные ***цифры*** числа, каждая из которых может быть 1, 2, 3, 4, 5, 6, 7, 8, 9 или 0, причем цифра a_3 должна быть ненулевой (иначе число не будет четырехзначным).

Число 10, степени которого записаны в формуле выше, называют ***основанием*** системы счисления.

3.10. Формула для p-ичной позиционной системы счисления

Выбор числа 10 в качестве основания позиционной системы в значительной мере объясняется традицией, а не какими-то замечательными свойствами числа 10. Можно рассмотреть и системы счисления с другим основанием p.

Записать число N в p-ичной системе счисления — это значит записать его в виде

$$N = a_n p^n + a_{n-1} p^{n-1} + \cdots + a_1 p + a_0,$$

где каждый из ***коэффициентов-цифр*** a_i может быть 0, 1, 2, ..., $p - 1$, причем старшая цифра a_n ненулевая.

Взяв основание равным 2, получаем систему всего с двумя цифрами 0 и 1 и простой таблицей умножения:

$$0 \cdot 0 = 0, \quad 0 \cdot 1 = 0, \quad 1 \cdot 1 = 1.$$

К сожалению, в такой системе даже небольшие числа записываются слишком длинно. Например, число 1995 в двоичной системе записывается 11 цифрами:

$$1995_{10} = 11111001011_2$$

(в этой записи внизу после числа указано основание системы счисления).

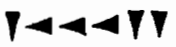
Если же взять за основание большое число, скажем, 60, то придется использовать 60 разных цифр, да и таблица умножения будет слишком длинной и заучить ее наизусть не удастся. Тем не менее такая позиционная система использовалась в Древнем Вавилоне примерно в 2500—2000 гг. до н.э.

3.11. Вавилонская шестидесятеричная система счисления

Вавилонская система имела основанием 60, и особую роль в ней играли числа 60, $60^2 = 3600$ и т.д. Младший знак числа означал число единиц, следующий знак — число «шестидесятков» и т.д. Каждый знак был числом от 1 до 59. Поскольку основанием было число очень большое — 60, вавилоняне записывали числа от 1 до 59 по десятичной системе, применяя принцип сложения. При этом они пользовались всего двумя знаками: вертикальным клином для обозначения 1 и горизонтальным для 10.

Таким образом, ***цифры***, т.е. все числа от 1 до 59, вавилоняне записывали по десятичной непозиционной системе, а число в

целом — по позиционной системе с основанием 60. Например, число 92 записывалось так:



Вавилонская нумерация имела и еще одну особенность: сначала знака для нуля не было вовсе. И если был изображен один вертикальный клин, нельзя было без дополнительных комментариев определить, какое число изображено: 1, 60, 3600 или какая-нибудь другая степень 60. Впоследствии вавилоняне придумали знак для обозначения пропущенного разряда (два горизонтальных клина, один над другим). Но, увы, этот знак не ставился в конце чисел.

Наконец, из-за такого большого основания таблица умножения была просто огромной, ее никто не запоминал, а при вычислениях использовалась готовая таблица, как мы совсем недавно пользовались таблицами логарифмов. Все это и предопределило судьбу вавилонской системы: дав толчок к развитию позиционных систем, вавилонская система была забыта, хотя отголоски ее былой распространенности можно заметить сейчас в делении часа на 60 минут и круга на 360 градусов.

3.12. Когда была придумана двоичная система счисления

Вопреки распространенному заблуждению, двоичная система счисления была придумана не инженерами — конструкторами электронных вычислительных машин, а математиками и философами задолго до появления компьютеров, еще в XVII—XIX вв. Великий немецкий ученый Лейбниц считал:

> *«Вычисление с помощью двоек... является для науки основным и порождает новые открытия... При сведении чисел к простейшим началам, каковы 0 и 1, везде появляется чудесный порядок».*

Позже двоичная система была забыта, и только в 1936—1938 гг. американский инженер и математик Клод Шеннон нашел замечательные применения двоичной системы при конструировании электронных схем.

3.13. Двоичная система — это очень просто

Разберемся в двоичной системе на примерах. Как узнать, чему равно девятизначное двоичное число $N = 111110100_2$? Составим таблицу из первых девяти степеней двойки: 2^0, 2^1, ... и поместим в нее цифры нашего двоичного числа.

n	8	7	6	5	4	3	2	1	0
2^n	256	128	64	32	16	8	4	2	1
111110100	1	1	1	1	1	0	1	0	0

Единички в этой таблице показывают, какие степени двойки нужно сложить, чтобы получить число:

$$N = 256 + 128 + 64 + 32 + 16 + 4 = 500.$$

Попробуем теперь найти двоичное представление какого-нибудь числа, скажем 393. Сначала запишем его как $393 = 256 + 137$. Затем запишем 137 как $128 + 9$. Далее запишем 9 как $8 + 1$. На каждом шагу от числа как бы «отщепляется» максимально возможная степень двойки. Получается, что

$$393 = 256 + 137 = 256 + 128 + 9 = 256 + 128 + 8 + 1.$$

Поставим в таблице единички под теми степенями, которые вошли в эту сумму, и нолики под теми, которые не вошли:

n	8	7	6	5	4	3	2	1	0
2^n	256	128	64	32	16	8	4	2	1
110001001	1	1	0	0	0	1	0	0	1

Выписывая эти единички и нолики подряд, получаем искомую двоичную запись числа 393:

$$393_{10} = 110001001_2.$$

Нахождение двоичного представления числа называется ***переводом*** числа в двоичную систему счисления.

3.14. Перевод чисел в двоичную систему счисления

Пусть нужно перевести в двоичную систему счисления число 234. Будем делить 234 последовательно на 2 и запоминать остатки, не забывая и про нулевые:

```
234 = 2*117 + 0
117 = 2* 58 + 1
 58 = 2* 29 + 0
 29 = 2* 14 + 1
 14 = 2*  7 + 0
  7 = 2*  3 + 1
  3 = 2*  1 + 1
  1 = 2*  0 + 1
```

Выписав все остатки, начиная с последнего, получим двоичное разложение числа: $234_{10} = 11101010_2$. Проверим этот алгоритм на числе 393:

```
393 = 2*196 + 1
196 = 2* 98 + 0
 98 = 2* 49 + 0
 49 = 2* 24 + 1
 24 = 2* 12 + 0
 12 = 2*  6 + 0
  6 = 2*  3 + 0
  3 = 2*  1 + 1
  1 = 2*  0 + 1
```

получаем: $393_{10} = 110001001_2$. Этот ответ совпадает с найденным в предыдущем пункте.

3.15. Алгоритм перевода числа в двоичную систему счисления

Общий алгоритм перевода числа в двоичную систему может быть записан так:

```
алг перевод в двоичную (арг цел N, рез лит t)
 дано  N>0
 надо | t=строка двоичного разложения числа N
нач цел x,остаток
 | x:=N
 | t:=""
 | нц пока x>0
 |  | остаток:=mod(x,2)
 |  | если остаток=0 то t:="0"+t иначе t:="1"+t все
 |  | x:=div(x,2)
 | кц
кон
```

При работе этого алгоритма для N = 393 будем последовательно получать:

x	остаток	t
393	1	"1"
196	0	"01"
98	0	"001"
49	1	"1001"
24	0	"01001"
12	0	"001001"
6	0	"0001001"
3	1	"10001001"
1	1	"110001001"

Запустите гипертекст «Алгоритм перевода в двоичную систему» и посмотрите, как работает этот алгоритм для разных N.

3.16. Алгоритм перевода из двоичной системы счисления в десятичную

Пример такого перевода был разобран в п. 3.13. Общий алгоритм можно записать так:

```
алг перевод из двоичной (арг лит t, рез цел N)
. дано | t строка из нулей и единиц
 надо | найти такое N,что t=(двоичное разложение N)
нач цел x, i, j
 | N:=0
 | нц для i от 1 до длин(t)
 |  | j:=длин(t)-i
 |  | если t[i]="1" то N:=N+2**j все
 | кц
кон
```

При работе этого алгоритма для t = "110001001" будем последовательно получать:

i	j	t[i]	2**j	N
1	8	"1"	256	256
2	7	"1"	128	384
3	6	"0"	64	384
4	5	"0"	32	384
5	4	"0"	16	384
6	3	"1"	8	392
7	2	"0"	4	392
8	1	"0"	2	392
9	0	"1"	1	393

Этот алгоритм работает прямо по определению двоичной системы: N вычисляется как сумма степеней двойки, соответствующих единицам в двоичном разложении t.

Запустите гипертекст «Алгоритмы перевода из двоичной системы» и посмотрите, как выполняется алгоритм для разных двоичных чисел.

3.17. Системы счисления, родственные двоичной

При работе с компьютерами иногда приходится иметь дело с двоичными числами, поскольку двоичные числа заложены в конструкцию компьютера. Двоичная система удобна для компьютера, но неудобна для человека — числа получаются очень длинными и их трудно записывать и запоминать. Конечно, можно перевести двоичное число в десятичную систему и записать его в таком виде, а потом, когда оно понадобится, перевести его обратно, но все эти переводы очень трудоемки. На помощь приходят системы, родственные двоичной — ***восьмеричная*** и ***шестнадцатеричная***. Перевод из родственной системы в двоичную и обратно может быть мгновенно выполнен в уме.

Начнем с восьмеричной системы. В этой системе 8 цифр: 0, 1, 2, 3, 4, 5, 6, 7. Цифра 1, записанная в самом младшем разряде, означает — как и в десятичном числе — просто единицу. Та же цифра 1 в следующем разряде означает 8, в следующем — 64 и т.д. Число 100_8 есть не что иное, как 64_{10}, а число 611_8 равно $6 \cdot 64_{10} + 1 \cdot 8_{10} + 1 = 393_{10}$.

Давайте переведем число 611_8 в двоичную систему. Нет ничего проще: достаточно *заменить каждую цифру на ее перевод в двоичную систему* и получится 9-значное двоичное число:

6 1 1
110 001 001

Обратно, если есть многозначное двоичное число, то для перевода в 8-ричную систему его нужно разбить на группы по три цифры справа налево и заменить каждую группу одной 8-ричной цифрой. Например:

$$111\,111\,111_2 = 777_8, \quad 11\,111\,101\,001_2 = 3751_8.$$

Запустите гипертекст «Системы счисления, родственные двоичной» и потренируйтесь в переводе из двоичной системы в восьмеричную и обратно.

Запись числа в 8-ричной системе достаточно компактна, но еще компактнее она получается в 16-ричной системе. Для первых 10 из 16 шестнадцатеричных цифр используют привычные цифры 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, а вот для остальных 6 цифр используют первые буквы латинского алфавита:

A – 10, D – 13,
B – 11, E – 14,
C – 12, F – 15.

Цифра 1, записанная в самом младшем разряде, означает просто единицу. Та же цифра 1 в следующем разряде означает 16_{10}, в следующем — 256_{10} и т.д. Цифра F, записанная в самом младшем разряде, означает 15_{10}, в следующем разряде — $15_{10} \cdot 16_{10}$ и т.д. Число 100_{16} есть не что иное, как 256_{10}, а число $AF0_{16}$ равно $10_{10} \cdot 16^2_{10} + 15_{10} \cdot 16_{10} = 2800_{10}$.

Часто нижний индекс 16 опускают, например, вместо BAD_{16} пишут просто BAD и это не приводит к ПЛОХИМ последствиям, поскольку в каждый момент понятно, о чем говорится — о числе или об английском слове *bad* (плохой).

Перевод из 16-ричной системы в двоичную делается переводом каждой цифры, например

A 0 F
1010 0000 1111

Запустите гипертекст «Системы счисления, родственные двоичной» и потренируйтесь в переводе из двоичной системы в шестнадцатеричную и обратно.

Упражнение:

4. Придумайте алгоритм перевода из 8-ричной системы счисления в 16-ричную. Переведите с его помощью число 12345670_8.

3.18*. Смешанные непозиционные системы счисления

Эти необычные системы счисления, хотя и давно известные математикам, получили неожиданные применения после появления компьютеров. В таких системах не одно **основание** p, а сразу несколько оснований $p_1, p_2, \ldots, p_m$. Эти m оснований должны быть различными простыми числами. Все числа смешанной системы с m основаниями являются m-значными.

Напомним, что *простым числом* называется натуральное число, большее 1, которое делится только на 1 и на самого себя. Например, 2, 3 и 5 — простые числа, а 4, 6 и 9 — не простые. Известно, что простых чисел бесконечно много и в ряду натуральных чисел они встречаются довольно часто. Вот все простые числа, меньшие 50: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47.

Выберем несколько простых чисел, например три числа $p_1 = 2$, $p_2 = 3$, $p_3 = 7$, и построим смешанную двоично-троично-семеричную систему. Каждое число в этой системе будет трехзначным: первая цифра будет 0 или 1, вторая — 0, 1 или 2, третья — 0, 1, 2, 3, 4, 5 или 6.

Перевод числа в двоично-троично-семеричную систему делается так. Для числа N вычисляем три «цифры»: остатки от деления N на 2, 3 и 7 соответственно. Вот что получится для небольших чисел N:

N	остатки от деления N			N	остатки от деления N		
	на 2	на 3	на 7		на 2	на 3	на 7
0	0	0	0	10	0	1	3
1	1	1	1	11	1	2	4
2	0	2	2	12	0	0	5
3	1	0	3	...	...	...	...
4	0	1	4	40	0	1	5
5	1	2	5	41	1	2	6
6	0	0	6	42	0	0	0
7	1	1	0	43	1	1	1
8	0	2	1	44	0	2	2
9	1	0	2	...	...	...	...

К сожалению, у числа $N = p_1p_2p_3 = 2 \cdot 3 \cdot 7 = 42$ «цифры» такие же, как у числа $N = 0$. Однако если взять два разных числа от 0 до 41, то их цифры не будут полностью совпадать. Так что эта система годится для чисел от 0 до 41: $000_{2\text{–}3\text{–}7} = 0$, $111_{2\text{–}3\text{–}7} = 1$, $022_{2\text{–}3\text{–}7} = 2$, $103_{2\text{–}3\text{–}7} = 3$, ..., $015_{2\text{–}3\text{–}7} = 40$, $126_{2\text{–}3\text{–}7} = 41$.

Замечательна эта система тем, что любое арифметическое действие в ней проводится поразрядно. Например,

$$\begin{array}{r} 103_{2\text{–}3\text{–}7} \\ + \\ 022_{2\text{–}3\text{–}7} \\ \hline 125_{2\text{–}3\text{–}7} \end{array} \qquad \begin{array}{r} 013_{2\text{–}3\text{–}7} \\ + \\ 113_{2\text{–}3\text{–}7} \\ \hline 126_{2\text{–}3\text{–}7} \end{array}$$

(Первая цифра суммы получается сложением первых цифр слагаемых, вторая — сложением вторых цифр, третья — сложением третьих.)

Однако при сложении цифр любого разряда нужно помнить, что результат должен быть остатком от деления на соответствующее простое число. Так что:

$$\begin{array}{r} 014_{2\text{–}3\text{–}7} \\ + \\ 125_{2\text{–}3\text{–}7} \\ \hline 102_{2\text{–}3\text{–}7} \end{array}$$

При сложении вторых цифр получилось 3, но в сумму записан 0. При сложении третьих цифр получилось 9, но третьей цифрой суммы записан остаток от деления 9 на 7, т.е. 2.

Умножение выполняется аналогично. Соответствующие цифры просто перемножаются:

$$\begin{array}{r} 125_{2\text{–}3\text{–}7} \\ \times \\ 021_{2\text{–}3\text{–}7} \\ \hline 015_{2\text{–}3\text{–}7} \end{array}$$

Здесь при умножении вторых цифр получилось число 4, но в произведение записана цифра 1 — остаток от деления 4 на 3.

Смешанные системы удобны для вычислений с очень большими целыми числами, скажем, 1000-значными десятичными. При умножении столбиком таких чисел придется сделать несколько миллионов операций с цифрами, в то время как умножение двух смешанных 1000-значных чисел требует всего-навсего несколько тысяч операций.

Запустите гипертекст «Смешанные системы счисления» и посмотрите, как записываются числа в различных смешанных системах счисления.

3.19*. Алгоритмы работы в смешанной системе счисления

Пусть m = 4 и в таблице целтаб p[1:m] записаны простые числа 2, 3, 5, 7. Поскольку остаток от деления числа A на число k записывается как mod(A,k), то число A в смешанной системе с основанием (2-3-5-7) можно записать как таблицу из m цифр целтаб a[1:m] по правилу:

Правило	*Примеры*		
	A=2	A=100	A=200
a[1] = mod(A,2)	a[1]=0	a[1]=0	a[1]=0
a[2] = mod(A,3)	a[2]=2	a[2]=1	a[2]=2
a[3] = mod(A,5)	a[3]=2	a[3]=0	a[3]=0
a[4] = mod(A,7)	a[4]=2	a[4]=2	a[4]=4

Алгоритм перевода числа A в систему с основанием (2-3-5-7) в общем виде можно записать так:

```
нц для i от 1 до m
| a[i]:=mod(A,p[i])
кц
```

Если числа A и B заданы в смешанной системе как

```
целтаб a[1:m], b[1:m],
```

то правила вычисления их суммы и произведения

```
целтаб c[1:m], d[1:m]
```

могут быть записаны так:

```
нц для i от 1 до m
| c[i]:=mod(a[i]+b[i],p[i])
| d[i]:=mod(a[i]*b[i],p[i])
кц
```

Упражнения:

5. Переведите в смешанную систему с основанием (2-3-5-7) числа 1, 2, 10, 100, $2 \cdot 3 \cdot 7$, $2 \cdot 5 \cdot 7$.
6. Переведите в десятичную систему число $1246_{2-3-5-7}$.
   Подсказка. Что получится, если к этому числу прибавить 1?

Запустите гипертекст «Смешанные системы счисления» и выполните компьютерные упражнения.

3.20. Геометрическое представление чисел

Числа могут быть представлены не только *алгебраически* специальными знаками на бумаге, папирусе или бересте, но и *геометрически*. Такое представление чисел широко используется в различных механических устройствах, например в спидометре.

Спидометр кодирует числовое значение скорости геометрически, как положение стрелки на круговой или линейной шкале. И это геометрическое представление ничуть не хуже представления цифрами, а в чем-то даже и лучше. Летчик, посмотрев на число, может ошибочно принять 600 км/ч за 900 км/ч, начать снижать скорость, и из-за нехватки скорости самолет войдет в штопор. Со стрелочным спидометром вероятность такой ошибки значительно меньше. Поэтому даже на самых современных самолетах часть приборов выводит данные не в виде символов, а в виде положения стрелки на циферблате, шкале и т.п.

Геометрическое представление экономических данных, результатов выборов, опросов общественного мнения широко используется в газетах, журналах, книгах. Такое представление называется ***диаграммой***.

Упражнения:

7. Спросите у 5—6-летнего ребенка, какие два из трех чисел 900, 600, 599 больше всего похожи друг на друга.
8. В авиации используется альтиметр — прибор для измерения высоты. Раньше он был похож на часы: круглый циферблат и две стрелки — одна показывала высоту в километрах, а другая — в метрах от 0 до 1000 — превышение над целым числом километров. Хотя стрелки и были разными, летчики их иногда путали, и это приводило к катастрофам. Как переделать альтиметр, чтобы он по-прежнему кодировал высоту, был похож на старый и не вводил летчиков в заблуждение? (Замечание: одну стрелку сделать нельзя — будет неточно; к тому же иногда надо быстро узнать высоту в километрах, не обращая внимание на метры.)

§ 4. МЕХАНИЧЕСКИЕ ВЫЧИСЛИТЕЛИ

4.1. Первые механические вычислители

До изобретения ЭВМ для счета использовались различные механические приспособления — абаки, счеты, арифмометры.

В абаке и счетах число кодируется положением камешков на

доске или костяшек на спицах. В арифмометре число кодируется положением (или поворотом) колесика с 10 зубцами. С помощью таких связанных колесиков можно построить суммирующее устройство. Впервые это сделал в 1641—1642 гг. француз Блез Паскаль. В суммирующей машине Паскаля десятичные цифры шестизначного числа задавались поворотами колесиков с цифровыми делениями, а результат считывался в шести окошечках.

Колесики были механически связаны, чтобы при сложении можно было учесть «перенос единицы» в следующий десятичный разряд. Колесико *единиц* было связано с колесиком *десятков*, колесико *десятков* с колесиком *сотен* и т.д. Если при повороте колесико переходило из положения 9 в положение 0, то следующее колесико подталкивалось и поворачивалось на один зубец. Этот поворот, в свою очередь, мог вызвать поворот на зубец следующего колесика и т.д.

Абак и счеты являются, в сущности, чисто запоминающими устройствами. Они только хранят информацию, а все действия проводит человек. А вот суммирующую машину Паскаля можно уже назвать «механическим вычислителем». Она автоматически проводит переносы единиц в следующий десятичный разряд, выполняя тем самым работу, которую раньше мог сделать только человек.

Абак, счеты и машина Паскаля мало помогают при умножении. Арифмометр с подвижной кареткой, имитирующий механически школьный алгоритм «умножение в столбик», сконструировал в конце XVII в. немецкий ученый Готфрид Лейбниц.

4.2. Перфоленты и перфокарты

Деревянные барабаны с отверстиями, бумажные ленты с отверстиями — ***перфоленты*** — и картонные карты с отверстиями — ***перфокарты*** — в течение всего XVIII в. использовались во Франции конструкторами ткацких станков в попытке заставить челнок станка работать автоматически, по программе, заданной расположением отверстий. Громадного успеха на этом пути добился Жозеф Мари Жаккард, французский изобретатель, сын лионского ткача. В 1804—1808 гг., в грозное для Европы время, когда Наполеон завоевывал одну страну за другой, он создал автоматический ткацкий станок, управляемый перфокартами. Наличие или отсутствие отверстий в одной перфокарте заставляло нить подниматься и опускаться при одном ходе челнока.

Станок Жаккарда был первым массовым промышленным

устройством, автоматически работающим по заданной программе. Этот станок был отмечен медалью Парижской выставки и вскоре только во Франции работало более 10 тысяч таких станков.

Неудивительно, что в проекте программируемой вычислительной машины Чарльза Бэббиджа, о которой будет еще сказано ниже, для задания программы работы машины также предусматривались перфокарты.

Перфокарты произвели переворот не только в ткацком деле, но и в статистике. Случилось это в конце XIX в.

Статистика постоянно сталкивается с проблемами обработки огромного количества информации. Когда сына немецких эмигрантов Германа Холлерита приняли на работу в статистическое управление при министерстве внутренних дел США, он тоже столкнулся с необходимостью ручной обработки гор бумаги — результатов переписи населения США 1890 г.

Доктор Джон Биллингс, возглавлявший работы по составлению сводных результатов переписи, высказал идею, что обработка может быть автоматизирована, если результаты переписи занести на перфокарты. Биллингс вдохновил молодого Холлерита на создание машины, которая механически выполняла бы работу многочисленных клерков. В 1884—1889 гг. Холлерит получает 34 патента на перфокарточные машины. Впервые машина Холлерита была опробована в 1887 г. В 1890 г. она победила в конкурсе машин для обработки результатов переписи населения США и с Холлеритом был заключен контракт на проведение переписи. Машина Холлерита оказалась быстрой, удобной, надежной, и результаты переписи 1890 г. были обработаны втрое быстрее, чем результаты предыдущей.

Развитые Холлеритом технологии хранения информации на перфокартах и изобретенные им электромеханические машины для обработки такой информации продвинули Человечество еще на один шаг вперед к компьютерной революции XX в.

Кстати, основанная Холлеритом фирма сегодня носит название IBM и является крупнейшим в мире производителем компьютеров.

4.3. Чарльз Бэббидж и его аналитическая машина

Одним из пионеров создания автоматических счетных машин был англичанин Чарльз Бэббидж, декан кафедры математики Кембриджского университета, той кафедры, которую когда-то

возглавлял Ньютон. Однажды, просматривая таблицу логарифмов и заметив массу неточностей, Бэббидж задумался, как можно было бы избежать вычислительных ошибок и опечаток в подобных таблицах.

К тому времени французскими учеными уже был применен любопытный метод вычислений, давший неплохие результаты. Большая задача разбивалась на небольшие части, состоящие лишь из простых операций, и поручалась большому количеству людей, ничего не знающих в математике, кроме арифметических операций, и выполнявших почти механически большое число таких операций.

Бэббидж решил для таких операций приспособить машины. В 1822 г. он опубликовал статью с описанием машины для вычисления и печати таблиц математических функций и в том же году построил рабочую модель, заслужившую восторженный прием Лондонского Королевского Общества.

Бэббидж получил от правительства небольшую начальную субсидию и начал постройку полноценной рабочей машины. Увы, задача оказалась сложнее, чем ожидалось, в конструкцию машины приходилось вносить исправления и усовершенствования, работа затягивалась. Выделяемых правительством средств постоянно не хватало, через несколько лет финансирование прекратилось и работы остановились. Но эти 10 лет не пропали даром. Бэббидж пришел к удивительной идее совершенно новой, универсальной машины — прообразу современных вычислительных машин.

В проекте Бэббиджа были предусмотрены все основные элементы, присущие современным компьютерам:

- ***склад*** для хранения чисел (память);
- ***фабрика*** для их обработки (арифметическое устройство);
- ***контора*** для управления обработкой (процессор).

Это был гениальный проект, но изобретение Бэббиджа опередило свое время; он умер, успев построить лишь отдельные компоненты универсальной машины. Конструкция машины была работоспособна, но ее невозможно было реализовать в полном объеме, пользуясь технологиями того времени.

§ 5. КООРДИНАТЫ

Закодировать можно не только числа, но и другую информацию. Например, информацию о том, где находится некоторый объект: планета, корабль, самолет, человек или фигура на шахматной доске. Величины, определяющие положение объекта, на-

зываются ***координатами***. Задание координат объекта позволяет точно и коротко указать положение объекта. Во многих случаях наиболее удобны числовые координаты, но встречаются и другие способы задания координат.

5.1. Координаты на шахматной доске

Рассмотрим шахматную доску. Вы хотите записать шахматную партию, чтобы потом спокойно проанализировать ее и найти, быть может, свои ошибки. Можно, конечно, зарисовывать доску после каждого хода, но это долго, требует много места и вообще неудобно. Попробуем закодировать ходы шахматных фигур. Указать вид фигуры нетрудно: Кр — король, Ф — ферзь, Л — ладья, С — слон, К — конь, отсутствие обозначения фигуры указывает на пешку. А как указать, с какой клетки и на какую сделан ход? Для этого на помощь приходят координаты.

Занумеруем горизонтали и вертикали шахматной доски от 1 до 8. Указав два числа, сперва номер горизонтали, а потом номер вертикали, мы сможем однозначно указать любую клетку на доске. Этот способ не вполне удобен, поскольку нужно постоянно помнить, что первая цифра указывает горизонталь. Удобнее сделать чуть-чуть по-другому: пронумеровать горизонтали по-прежнему цифрами, а вот вертикали обозначить первыми восемью латинскими буквами: a, b, c, d, e, f, g, h. Теперь для записи первого хода знаменитого гроссмейстера Бендера требуется совсем мало места — достаточно написать: ***e2–e4***.

Упражнение:

9. В России распространены так называемые русские шашки, игра эта идет на шахматной доске. Система координат на этой доске вводится точно такая же, как в шахматах. А вот в международных шашках борьба ведется на доске 10×10, но, главное, в этой игре принята принципиально другая система записи координат клеток. Придумайте какой-нибудь способ задания координат клеток для шашек на доске 10×10, отличный от «шахматного способа».

Подсказка. Шашки ходят только по черным клеткам.

5.2. Координаты в кинозале

В шахматы играют не все, а вот в кино был, наверное, каждый. Вы пришли в кино. Зал вмещает достаточно много людей. Как избежать неразберихи в размещении, споров из-за мест? От-

вет напрашивается сам собой — как-то сделать места различимыми. Например, можно попробовать пронумеровать места подряд. Но представьте себе, что надо найти кресло номер 257. Неудобно, не правда ли? Поэтому в кинотеатрах отдельно нумеруют ряды, и отдельно — кресла в этих рядах. Допустим, куплен билет, на котором написано: ряд 12, место 4. Два числа — 12 и 4 — и есть та информация, которая компактно и однозначно указывает расположение кресла. Эти два числа можно, следовательно, назвать ***координатами кресла*** в кинозале.

Основное требование к координатам — однозначное задание объекта. Другую информацию из координат иногда можно извлечь, а иногда нельзя. Например, вы купили два билета в hbox12-й ряд, места 14 и 15. Будут ли эти два места соседними? Для ответа на этот вопрос нужна дополнительная информация. Если в каждом ряду 28 мест и зал разделен проходом на левую и правую половины, то, увы, места не будут рядом, хотя координаты у них «соседние».

5.3. Координаты на плоскости

Перейдем теперь к более математическим объектам — точкам на плоскости — и введем числовые координаты для таких объектов.

Впервые это сделал французский математик и философ Рене Декарт. Он ввел на плоскости так называемую ***прямоугольную (декартову)*** систему координат.

Иногда, особенно в иностранной литературе, такую систему называют *картезианской*. Это связано с тем, что имя Рене Декарта (Descartes) на латинский манер пишется Cartesius. Именно от этого написания и пошло название «картезианская (cartesian) система координат».

Проведем на плоскости две перпендикулярные прямые. Точка пересечения этих прямых называется ***началом координат***. Обозначим начало координат через O, выберем на прямых положительное направление и обозначим лучи Ox и Oy. Выберем единицу масштаба и отложим ее на лучах. Эта конструкция полностью задает систему координат на плоскости, и теперь мы можем описать положение любой точки плоскости, указав два числа — координаты этой точки. Делается это так.

Пусть нам надо определить координаты произвольной точки M. Опустим из нее два перпендикуляра MA и MB на координатные оси. Два числа — длины отрезков OA и OB и есть, по определению, координаты точки M (длины отрезков OA и OB

нужно брать в алгебраическом смысле, т.е. они могут быть и отрицательными).

Нечто похожее было и в кинозале. Две перпендикулярные прямые — это две стены: та, где находится экран (расстояние до нее задается номером ряда), и боковая (расстояние до нее задается номером места). Правда, расстояния в кинозале всегда целые и измеряются несколько необычно — не в метрах или дециметрах, а в ***креслах***.

5.4. Системы координат бывают разные

На одной и той же плоскости можно ввести разные декартовы системы координат. Если выбрать другие координатные прямые или другую единицу масштаба, то для той же самой точки получатся другие координаты. Вспомните координаты на поле Чертежника. Начало координат находилось в левом нижнем углу поля. В системе команд Чертежника, однако, есть команда, меняющая систему координат. Не вдаваясь в детали, приведем пример. По команде

```
задать поле(-10,10,-7,7)
```

Чертежник сменит систему координат на своем поле и начало координат окажется в центре поля Чертежника. Так что, если до смены координат команда

```
сместиться в точку (0,0)
```

перемещала перо Чертежника в левый нижний угол поля, то после смены координат та же самая команда переместит перо в центр поля.

Можно рассмотреть и более общие системы координат. Например, взять разные масштабы на координатных прямых, или в качестве координаты брать не просто длину отрезка, а что-то более хитрое (например, логарифм длины), и так далее. Можно взять координатные прямые не перпендикулярными, а просто пересекающимися. Тогда получатся так называемые ***косоугольные*** координаты. Все описанные системы не менее важны, чем декартова прямоугольная система координат.

Полярная система координат на плоскости

Для ее построения выбирается произвольная точка плоскости O. Принято называть ее ***полюсом***. Выбирается луч, выходящий из полюса — ***ось координат***. Для произвольной точки M строится так называемый *радиус-вектор* OM — отрезок с конца-

ми O и M. Координатами точки будут два числа: длина радиус-вектора и угол между радиус-вектором и осью координат. Эти два числа называют ***радиус*** и ***полярный угол*** точки.

Полярный угол измеряется от 0° до 180° против часовой стрелки относительно оси и от 0° до −180° (исключая −180°) по часовой стрелке. Таким образом, для любой точки (кроме полюса) определены две координаты: радиус и угол. Осталось разобраться с координатами полюса. Радиус полюса, без сомнения, равен нулю, а вот полярный угол полюса не определен, его можно считать любым числом. Впрочем, для полюса полярный угол и не нужен. Если радиус какой-то точки равен нулю, то никакой дополнительной информации для задания точки не нужно. И так ясно, что эта точка есть полюс.

При использовании полярной системы координат часто возникает следующая неприятность: при непрерывном, плавном изменении положения точки полярный угол вдруг скачком меняется от 180 до −180 или наоборот. Чтобы избавиться от этой неприятности, вводят ***обобщенные полярные координаты***. В таких координатах полярный угол измеряется в обоих направлениях неограниченно, и неприятных скачков можно избежать.

Упражнения:

10. Каждой точке M первой четверти сопоставим два числа: длину отрезка OM и площадь прямоугольника $OAMB$. Можно ли использовать эти числа в качестве координат точки M?
11. Вывести формулы перевода из полярной системы в декартову в предположении, что начало декартовой системы координат совпадает с полюсом, а ось Ox совпадает с полярной осью.
12. Нарисовать кривую, заданную в обобщенных полярных координатах формулой $r = 5 \cdot (720° - \varphi)/720°$, где полярный угол φ меняется от 0 до 720° градусов.

Указание. Изменение φ от 0° до 720° означает, что точка кривой сделает два оборота вокруг начала координат. Разность $(720° - \varphi)$ при этом будет меняться от 720° до нуля, дробь $(720° - \varphi)/720°$ будет меняться от 1 до 0, а радиус r будет меняться от 5 до нуля. Указанную кривую можно нарисовать на бумаге вручную или составить программу для Чертежника. В последнем случае понадобятся формулы из упражнения 11.

5.5*. Координаты на местности и карте

Вернемся к практике. Как задать положение того или иного объекта: прохода между скалами, брода или родника?

Часто нужное место задается объяснением, как до него добраться от какого-то другого известного места.

«Высокое дерево на склоне Подзорной трубы, направление от N к NNE. Остров скелета ESE. Десять футов. Слитки серебра в серебряной яме. Отыщешь ее на склоне восточной горки, в десяти саженях к югу от черной скалы, если встать к ней лицом...»

Дж. Ф. Стивенсон. «Остров Сокровищ»

Здесь N (англ. North) означает Север, S (англ. South) — Юг, E (англ. East) — Восток, W (англ. West) — Запад.

Вместо такого путеводного рассказа можно использовать рисунки местности, но и они не обеспечивают необходимой точности в наш век самолетов, скоростных поездов, теплоходов. Попробуйте представить себе капитана корабля, объясняющего диспетчеру линии свое местонахождение: «Я доплыл до острова Хоккайдо, повернул на северо-восток, проплыл 10 миль» — и так далее. Или геолога, приблизительно зарисовывающего новое месторождение. Точность является неотъемлемым спутником современной цивилизации. И здесь на помощь приходят координаты на местности.

До этого мы сталкивались с координатами на плоскости или ее частях, а Земля представляет собой шар. Как же задать систему координат на земном шаре? Если речь идет о небольшом участке земной поверхности, то можно условиться считать его плоским, так как кривизной небольшого участка допустимо пренебречь. А если мы хотим получить представление о чем-то большом, например о территории всей России?

Последние несколько сот лет для этого, как правило, используют так называемые ***географические координаты — широту*** и ***долготу***.

Географическая широта — это угол между радиусом, проведенным в данную точку, и плоскостью экватора. Она отсчитывается от 0° до 90° по обе стороны экватора. Например: «24° северной широты». Географическая долгота — угол между плоскостью меридиана, проходящего через данную точку, и плоскостью начального (Гринвичского) меридиана. Долготы от 0° до 180° к западу от начального меридиана называют западными, к востоку — восточными.

Мореплаватели до недавнего времени определяли свои координаты с помощью двух приборов: секстанта, позволяющего измерить угол Солнца над горизонтом, и хронометра, показывающего время по Гринвичу. В наши дни для навигации используют ком-

пьютеры, обрабатывающие радиосигналы со спутников и наземных радиостанций.

Координаты на карте

Планы городов и другие карты часто покрываются сеткой квадратов, обозначаемых по горизонтали и вертикали цифрами и буквами. Индекс квадрата, скажем В3 или Д9, позволяет легко найти квадрат на плане. В путеводителе по городу для объектов обычно указываются индексы: кинотеатр «Салют» (В3), улица Центральная (В3—Д9) и т.п. Такие обозначения помогают найти объект на плане, но координатами объектов в городе они, строго говоря, не являются. Один и тот же индекс имеют обычно несколько объектов. Более серьезные карты позволяют найти и географические координаты объекта. Для этого на карту с определенным шагом наносятся линии, на которых широта или долгота постоянны. Например,

$$\text{широта} = 56°30', 56°45', 57°00', 57°15, \ldots$$
$$\text{долгота} = 35°15', 35°30', \ldots$$

Если карта охватывает небольшой участок земной поверхности далеко от полюсов, то такие линии равных широт и долгот можно сделать прямыми. В общем случае эти линии будут криволинейными. Поскольку поверхность Земли является шаром, а карта есть часть плоскости, при изображении местности на карте неизбежны разного рода искажения. Например, если соединить на карте два пункта прямолинейным отрезком, то полученный путь на местности не обязательно будет кратчайшим.

Упражнение:

13. Самолет-заправщик вылетает из Гринвича и летит 2000 км на запад. Там он встречается с другим самолетом, заправляет его, разворачивается и летит на восток до Гринвича. Верно ли, что маршрут самолета-заправщика кратчайший?

5.6*. Координаты вблизи поверхности Земли

Географические координаты были придуманы мореплавателями и хороши для задания положения кораблей, маяков, опасных подводных камней и других объектов на море и на суше.

Эти координаты позволяют закодировать положение объекта, находящегося на поверхности Земли или океана. Высота объекта

географическими координатами не охватывается. Для указания точного положения самолета, подводной лодки или эпицентра землетрясения географических координат недостаточно. Нужна еще одна координата — ***высота над уровнем моря***.

Уровень моря — это усредненный уровень вод Мирового океана. Если точка находится над уровнем моря, то говорят: столько-то метров (километров) над уровнем моря (или выше уровня моря), иначе — ниже уровня моря.

Широта, долгота и высота над уровнем моря хороши своей наглядностью и удобны для проведения навигационных расчетов человеком. Но в наше время такие вычисления повсеместно проводятся на компьютерах и наглядность становится не столь важной. На первый план выступает теперь удобство координат при составлении программ. Для этих целей оказываются более удобными математические системы координат в пространстве, привязанные к Земле, Солнцу или даже к звездам.

Например, в пространстве можно ввести декартову систему координат, поместив ее начало в центр Земли и проведя оси через три определенные точки земной поверхности. В такой системе координат положение объекта будет задаваться тремя числами. Для сложных вычислений такие координаты удобны. Например, измерив с помощью радиосигналов расстояние от объекта до нескольких спутников, можно очень точно определить три координаты объекта. А при выдаче информации человеку эти три математические координаты компьютер может легко преобразовать в наглядные для человека широту, долготу и высоту над уровнем моря.

5.7*. Абстрактные координаты

Вы, наверное, встречали объявления такого рода: «Продаю дешево садовый домик, позвоните по телефону 12–34–56 и оставьте свои **координаты**».

Какие координаты — не сказано. Быть может, автор объявления интересуется полярными координатами предполагаемого покупателя или его широтой и долготой? Разумеется нет! Продавца интересует информация, позволяющая найти покупателя. Годится и почтовый адрес, и телефон, и любая другая информация. Например, покупатель может оставить на автоответчике продавца такие координаты: «Я звоню по поводу садового домика, пришлите мне описание домика и вашу цену по адресу: п/о Пригородное, до востребования, предъявителю паспорта №237568».

Как видно из этого примера, в наши дни термин *координаты*

начинает терять свое узкоматематическое значение и понимается все более и более широко.

Координаты могут включать не только пространственную, но и временную информацию. Рассмотрим такую ситуацию. Вы купили железнодорожный билет. Информации на билете достаточно, чтобы найти свое место и уехать. Поэтому можно сказать, что на билете заданы координаты купленного вами места в поезде. Эти координаты состоят из номера поезда, номера вагона и номера места, а также из даты и времени отправления.

Нельзя ввести координаты всего на свете или неизвестно чего. Координаты всегда вводятся для каких-то объектов, вводятся где-то: для фигуры на шахматной доске, для кресла в кинозале, для точки на плоскости или в пространстве. По аналогии со множеством точек в трехмерном пространстве ученые называют все множество объектов, с которыми мы собираемся иметь дело, ***абстрактным пространством*** и говорят о введении ***абстрактных координат*** в этом пространстве.

Как координаты в соответствующих абстрактных пространствах можно рассматривать очень многое: номера телефонов, почтовые адреса, номера маршрутов городского транспорта, номер банковского счета и т.д.

Такая «координатная» терминология стала вводиться в процессе компьютеризации. Дело в том, что компьютер должен хранить информацию о каких-то объектах. Эту информацию нужно как-то закодировать, и по этой информации нужно уметь однозначно определять, к какому объекту она относится. Таким образом, часть информации должна однозначно задавать объект, а это и есть основное свойство координат. Так что естественно назвать эту часть информации координатами объекта.

§ 6. НОТНАЯ АЗБУКА. КОДИРОВАНИЕ ПРОСТЕЙШИХ МЕЛОДИЙ

Русский композитор Ю. А. Шапорин писал:

> *«Мне кажется, нет на свете другого искусства, которое так роднило людей, как музыка. Язык ее понятен каждому...»*

Он имел в виду, что миллионы людей понимают язык музыки в том смысле, что способны слушать и наслаждаться музыкой, не зная ни названий музыкальных инструментов, ни принципов создания музыкальных произведений. И в этом он, разумеется, прав. Но можно подойти к вопросу о языке музыки более узко, более технически, и обсудить вопрос о том, как «закодировать», записать музыкальное произведение так, чтобы его можно было безошибочно воспроизвести.

Как и всякий звук, музыка является не чем иным, как звуковыми колебаниями и, зарегистрировав эти колебания достаточно точно, звук можно будет впоследствии воспроизвести. Такие способы записи любого звука будут обсуждаться в главе 3. А сейчас нас интересует способ задания именно музыки, способы задания отдельных нот, мелодий, темпа и т.д. Такой способ задания музыки должен позволить композитору записать придуманное произведение, а исполнителю — воспроизвести его, ни разу не слышав исполнение.

6.1. Несколько слов о нотной азбуке

Задача кодирования музыкальной информации решалась человечеством многие века и привела в конце концов к современной нотной записи.

Звуки мелодий в глубокой древности записывали буквами, в средние века — особыми значками — ***невмами***, которые указывали на нисходящее и восходящее движение мелодии, не определяя точной высоты звуков.

В X в. невмы стали писать на линейках, точно определяющих высоту звука, то есть тон. Этим мы обязаны Гвидо Аретинскому, который и изобрел систему линий, а тонам дал названия.

Затем стали записывать звуки при помощи особых значков — ***нот*** — на пяти линейках. Каждая нота указывает продолжительность звука своим видом, а высоту звука — положением на линейках.

Высота звука, его тон, определяется частотой звукового колебания. Каждая нота имеет свою строго определенную частоту.

> Сейчас основной считается нота «ля» третьей октавы — 440 Гц. Остальные ноты определяются относительно нее, так как самым важным для музыкальных целей, для восприятия человеком, оказывается не абсолютная частота, а отношение частот. Еще до недавнего времени за тон «ля» третьей октавы была принята частота 435 Гц.

Кроме пяти основных линеек, существуют добавочные — пять вверху и пять внизу. Что означают ноты на линейках, указывает ключ, который ставят в начале пяти линеек. Для фортепьянной музыки характерны два ключа — скрипичный и басовый. Для записи хоровой и оркестровой музыки существуют и другие ключи: «до», «соль», «фа».

Мы описали только основные понятия нотной азбуки, полный язык нотной записи гораздо сложнее и многообразнее.

В некоторых случаях такой сложный язык не нужен и с ним успешно конкурируют более простые системы кодирования. Например, при пении под гитару для записи партии гитары используется система аккордов — созвучий. Каждый аккорд имеет свое название, и партия записывается просто в строку последовательными названиями аккордов. Например,

F Em7 A7 Dm и т.д.

6.2. Звуковая гамма. Ноты и октавы

Элементарный звук, так называемый ***чистый тон***, с точки зрения физики представляет собой колебание определенной частоты. Проиграв чистый тон в течение определенного времени, получают элементарную единицу музыкального произведения — ноту. Нота полностью определяется своей частотой и продолжительностью.

В силу особенностей человеческого восприятия те звуки, частоты которых отличаются ровно в 2 раза, кажутся похожими, «родственными». Интервал частот от 262 Гц до $2 \times 262 = 524$ Гц разбивается (не равномерно) на 12 частей, и полученные 12 звуков называют первой октавой. Разумеется, вы знаете названия семи нот: *до*, *ре*, *ми*, *фа*, *соль*, *ля*, *си*. Они соответствуют белым клавишам фортепьяно. Оставшиеся 5 нот называются *до-диез*, *ре-диез*, *фа-диез*, *соль-диез*, *ля-диез* и соответствуют черным клавишам.

Если удвоить частоты этих 12 звуков, получится вторая октава. Если еще раз удвоить, получится третья.

6.3. Задание простейших мелодий в КуМире

В компьютерных программах для простейших мелодий используется такой способ записи: указываются последовательно частоты и длительности звуков мелодии. В КуМире для этого есть команда

```
нота(арг цел f,t)
```

При выполнении этой команды встроенный динамик в компьютере будет воспроизводить ноту с частотой `f` Гц в течение `t` сотых долей секунды.

Чтобы сделать паузу, нужно указать вместо значения частоты `f` = 0. Например, `нота(0,5)` дает паузу в пять сотых секунды.

Попробуем составить программу, играющую мелодию «Чижик-пыжик». Первой нотой мелодии выберем ноту *ми* первой октавы, а длительность выберем 20 сотых секунды. Для начала сыграем 4 первые ноты. Составим программу:

```
алг Чижик
нач
| нота(330,20) | Чи
| нота(277,20) | жик-
| нота(330,20) | пы
| нота(277,20) | жик
кон
```

и проиграем ее, нажав F9. Узнать начало мелодии можно, но звучит как-то непривычно. Дело в том, что между отдельными нотами у нас не сделаны промежутки. Добавим после каждой ноты паузу в 5/100 секунды с помощью команды `нота(0,5)`:

```
алг Чижик
нач
| нота(330,20); нота(0,5)  | Чи
| нота(277,20); нота(0,5)  | жик-
| нота(330,20); нота(0,5)  | пы
| нота(277,20); нота(0,5)  | жик
кон
```

и снова нажмем F9.

Теперь мелодия звучит гораздо лучше. Поскольку придется делать паузу после каждой ноты мелодии, удобно ввести вспомогательный алгоритм

```
алг Нота(цел f,t)
нач
| нота(f,t);нота(0,5)
кон
```

делающий паузу в 5/100 секунды после каждой ноты. С помощью этого вспомогательного алгоритма основной алгоритм можно записать немного короче. Окончательный алгоритм выглядит так:

```
алг Чижик
нач
| Нота(330,20) | Чи
| Нота(277,20) | жик-
| Нота(330,20) | пы
| Нота(277,20) | жик
```

```
    Нота(349,20) | где
    Нота(330,20) | ты
    Нота(294,40) | бы-ыл?

    Нота(196,20) | На
    Нота(196,20) | Фон
    Нота(196,20) | тан
    Нота(220,10) | ке
    Нота(247,10) | -е

    Нота(262,20) | вод-
    Нота(262,20) | ку
    Нота(262,40) | пи-ил!
кон
алг Нота(цел f,t)
нач
| нота(f,t); нота(0,5)
кон
```

В этом алгоритме два раза встречаются ноты удвоенной длительности («бы-ыл», «пи-ил») и два раза укороченные вдвое ноты (в конце слова «Фонтанке»).

Выполните задания «Изменение высоты мелодии» и «Изменение темпа мелодии» темы «Нотная азбука»

6.4. Сохранение мелодий в файлах

Программа для мелодии *Чижик-пыжик* приведена только для того, чтобы показать, как в принципе можно создать мелодию на компьютере. Если же нужно работать с мелодиями практически, то удобнее иметь одну универсальную программу, проигрывающую любую мелодию, а сами мелодии хранить в файлах в каком-то закодированном виде. Например, в файле можно хранить последовательность пар целых чисел: (частота ноты, длительность ноты).

Тогда можно будет подготовить файл CHIZHIK.MUZ для мелодии *Чижик-пыжик*:

```
330 20 0 5 262 20 0 5 330 20 0 5 262 20 0 5
349 20 0 5 330 20 0 5 294 40 0 5 196 20 0 5
196 20 0 5 196 20 0 5 220 10 0 5 247 10 0 5
262 20 0 5 262 20 0 5 262 40 0 5
```


другой файл WECHERA.MUZ для мелодии *Подмосковные вечера*:

```
294 20 0 5 349 20 0 5 440 20 0 5 349 20 0 5
392 40 0 5 349 20 0 5 330 20 0 5 440 40 0 5
392 40 0 5 294 40 0 5
```

и так далее.

Проиграть каждый такой файл можно будет коротенькой универсальной программой

```
алг проиграть(лит файл)
нач цел f,t
| начать чтение(файл)
| нц пока не конец файла
| | ввод f,t; нота(f,t)
| кц
| кончить чтение
кон
```

Простейший музыкальный редактор

Если вы знакомы с нотной записью, вы можете попробовать записать простейшие мелодии с помощью простейшего музыкального редактора. Как и настоящие компьютерные музыкальные редакторы, он позволяет редактировать нотную запись, проигрывать ее, сохранять в файле, загружать из файла и т.д.

Запустите музыкальный редактор из темы «Нотная азбука» и попробуйте записать мелодию, ноты которой вам известны.

§ 7. КОДИРОВАНИЕ БОЛЬШИХ ОБЪЕМОВ ИНФОРМАЦИИ В БУХГАЛТЕРИИ

В этом параграфе мы рассмотрим еще один пример из жизни, который приводит к необходимости кодировать информацию. Речь пойдет о хранении и обработке данных, связанных с денежными расчетами. В каких областях требуются эти данные? Наверное, каждый представляет себе, что такое семейный бюджет. Может быть, у вас в семье записываются все полученные деньги и все потраченные, доходы и расходы. Записи накапливаются каждый день. За год оказывается исписанной тетрадка, может быть,

толстая, может быть, не очень. В ней всегда можно найти, когда, сколько и на что было истрачено.

По существу, тем же целям служит ***бухгалтерская отчетность*** на предприятии. От записей в семейном бюджете ее отличают, во-первых, большой объем и разнообразие учитываемой информации, а во-вторых, наличие определенных правил ведения *бухгалтерского учета*, устанавливаемых государством. Сейчас мы коротко опишем эти правила.

7.1. Бухгалтерский учет

В бухгалтерии учитываются *средства*, то есть деньги и материальные ценности (в денежном эквиваленте). Все средства в бухгалтерии учитываются дважды. Во-первых, с точки зрения их состава (деньги, готовая продукция и т.д.) и размещения (расчетный счет или касса предприятия). Такие средства называются ***активами***. Во-вторых, с точки зрения источников их образования (прибыль, уставный фонд и т.д.). Такие средства называются ***пассивами***.

Важнейшим понятием бухгалтерии является понятие ***баланса***. Баланс — это таблица, в левой части которой перечислены активы предприятия, а в правой части — пассивы. Так как актив и пассив являются двумя точками зрения на одни и те же средства, сумма активов баланса должна совпадать с суммой пассивов.

Все средства относятся к тем или иным счетам. ***Счётом*** называется определенная группа средств или источников их образования, выделенная по какому-либо признаку. Счета делятся на активные и пассивные в зависимости от того, учитывают они средства или источники их образования. Каждый счет имеет свой шифр, который используется вместо названия счета. Вот примеры счетов, которые используются в бухгалтерском учете:

Активные		*Пассивные*	
01	Основные средства	02	Износ основных средств
07	Строительные материалы	60	Долг поставщикам
40	Готовая продукция	80	Прибыль
50	Касса	85	Уставный фонд
51	Расчетный счет	86	Амортизационный фонд

Бухгалтерия учитывает *движение* средств. Единицей движения в бухгалтерском учете является ***проводка***. В каждой проводке участвуют два счета, один из которых *дебетуется*, а другой *кредитуется*. В случае, когда оба счета пассивные, дебет означает уменьшение средств на счете, а кредит — увеличение. Например, при начислении износа основных средств дебетуется счет 80, а кредитуется счет 02. Можно представить себе, что деньги вычитаются из прибыли и откладываются для будущего ремонта или замены изношенного оборудования.

При различных хозяйственных операциях конкретные суммы на тех или иных счетах изменяются, однако равенство актива и пассива не нарушается. По каждому счету ведется ведомость, в которой фиксируются приход и расход средств по этому счету. В конце каждого месяца на основании ведомостей составляются *журналы-ордера́*, в которых учитывается суммарный приход и расход средств на каждый счет или группу счетов. На основании журналов-ордеров в конце отчетного периода вносятся записи в *Главную книгу* и составляется *баланс предприятия*.

Представьте себе теперь крупное предприятие, в бухгалтерии которого ведутся несколько десятков счетов, по каждому из которых в течение месяца бывает несколько сотен или тысяч проводок. Поддержка равенства актива и пассива баланса, заполнение журналов-ордеров требуют при таких объемах колоссальных усилий. Добавьте к этому учет документов, на основании которых делаются проводки, составление платежных поручений, ведомостей зарплаты, приходных и расходных ордеров, и вы поймете, что ведение бухгалтерского учета — это объемная, кропотливая и ответственная работа. И это та область, в которой компьютер может сильно помочь человеку. Какие же бухгалтерские задачи можно поручить компьютеру?

7.2. Кодирование бухгалтерской информации

Прежде всего, компьютер можно использовать как хранилище информации о движении средств.

Про каждый счет нужно помнить следующую информацию: шифр счета, название счета, тип (активный или пассивный), остаток (текущее количество средств на этом счете). Информацию о счетах можно поместить в таблицу, в которой каждая запись (строка таблицы) имеет следующую структуру:

Таблица счето́в

шифр	*название*	*тип*	*остаток*

Например, если в кассе предприятия находится один миллион рублей, соответствующая запись в таблице счетов имеет вид

50	«Касса»	актив	1 000 000

Далее, необходимо хранить информацию о движении средств. Заметим, что дебетуемый и кредитуемый счета не могут быть выбраны произвольно. Существуют строго определенные типы проводок, которыми бухгалтер имеет право пользоваться. Поэтому при кодировании бухгалтерской информации необходимо закодировать этот список. Это можно сделать в отдельной таблице следующей структуры:

Таблица типов проводок

тип	*название*	*шифр дебета*	*шифр кредита*

Здесь *тип* — это своего рода шифр проводки, по которому из других таблиц можно ссылаться на таблицу типов проводок, *название* — это полное название проводки, *шифр дебета* и *шифр кредита* — это шифры счетов, участвующих в проводке. Например, таблица типов проводок может содержать следующие записи:

1	Продажа готовой продукции	40	51
2	Снятие денег с расчетного счета	51	50
3	Начисление износа основных средств	80	02

Про каждую проводку нужно знать, какому типу она принадлежит, в какой момент времени произошла и какое количество средств было переведено. Для кодирования информации о проводках используется таблица, структура записи в которой имеет вид

Таблица проводок

номер	*тип*	*дата*	*количество*

Если двадцатого ноября 1994 г. проводкой номер двадцать три тысячи четыреста пятьдесят шесть был начислен износ в размере ста тысяч рублей, то запись в таблице проводок будет иметь вид

23456	3	20.11.1994	100 000

Разумеется, объем хранимой в бухгалтерии информации не исчерпывается тем, что описано выше. Необходимо хранить документы, на основании которых сделаны проводки, отчетную информацию за определенный период времени и многое другое. В следующем пункте рассказывается, как можно пользоваться этой информацией.

Упражнения:

14. Как в таблице счетов записывается:
    а) наличие готовой продукции на сумму 1,5 миллиона рублей;
    б) прибыль в размере 800 тысяч рублей?

15. Запишите в таблицу проводок следующие операции:
    а) Снятие 300 тысяч рублей с расчетного счета, выполненное 10 декабря 1995 г. (проводка номер 23457).
    б) Продажа готовой продукции на сумму 500 тысяч рублей (11 декабря 1995 г., проводка номер 23458).

16. Как может выглядеть таблица документов в программе бухгалтерского учета?

7.3*. Обработка бухгалтерской информации

Компьютер может подсчитать итоговый оборот средств по любому счету за любой период времени. В журналах-ордерах требуется указывать итоговое количество средств, переведенных с одного счета (пусть он имеет шифр a) на другой счет (с шифром b). С помощью предложенного здесь способа кодирования это достаточно просто сделать. Сначала надо выбрать из таблицы типов проводок все записи, у которых шифр дебета равен a, а шифр кредита b. Для каждого типа надо выбрать из таблицы проводок все записи, у которых дата попадает в нужный период времени, и просуммировать значения в столбце «количество». Потом итоговые количества надо просуммировать по всем выбранным типам проводок. При большом количестве проводок компьютер позволит сберечь много сил и времени по сравнению с ручным подсчетом. И компьютер не допустит ошибок!

Программа бухгалтерского учета способна облегчить и повседневный труд человека. Например, при выдаче денег из кассы помимо бухгалтерской проводки необходимо заполнить расходный ордер. Эту работу также может выполнить компьютер: человек указывает сумму выдачи, а компьютер делает проводку и печатает ордер. Это экономит время и гарантирует, что сумма, указанная в ордере, в точности совпадает с суммой проводки. Таким образом, компьютер уменьшает риск бухгалтерских ошибок.

Упражнения:

17. Пусть таблица типов проводок записана в таблицах

```
целтаб тип[1:n], литтаб имя[1:n]
целтаб дебет[1:n],кредит[1:n]
```

а таблица проводок — в таблицах

```
целтаб номер[1:m],тип1[1:m],дата[1:m]
вещтаб колич[1:m]
```

Считается, что дата записывается в днях от начала года (т.е. первому января соответствует 1, второму января — 2, первому февраля — 32 и т.д.). Составьте программу суммирования средств, переведенных со счета a на счет b за январь текущего года.

18. Можно ли ускорить алгоритм из предыдущей задачи, если известно, что таблица тип1 упорядочена по возрастанию (т.е. проводки в таблице проводок сгруппированы по типам)?

Глава 3 КОДИРОВАНИЕ ИНФОРМАЦИИ НА КОМПЬЮТЕРЕ

§ 8. ИНФОРМАЦИЯ

В начале темы «Кодирование информации на компьютере» полезно разобраться, что же такое ***информация***. Само слово *информация* лишь сравнительно недавно — примерно в середине XX в. — стало превращаться в точный термин. До этого информацию воспринимали только как *нечто*, содержащееся в речи, письме или передающееся при общении. Сейчас смысл, вкладываемый в это понятие, сильно изменился и расширился. Появилась особая математическая дисциплина — ***теория информации***. Выяснилось, что информацию можно покупать и продавать, зарабатывать на ее хранении и передаче. Появились словосочетания «средства массовой информации», «защита информации», «информационный голод», «информационные услуги» и даже «информационное общество».

8.1. Нужно ли строго определять понятие «информация»?

Что же значит слово *информация* во всех этих словосочетаниях? Увы, точного определения этого термина, охватывающего все случаи, не существует. Хотя в теории информации и вводится несколько строгих определений, эти определения даются в весьма ограниченных предположениях и не охватывают всего богатства понятия «информация». В других математических теориях ученые вообще отказываются от попыток дать определение и рассматривают информацию как первичное, неопределяемое понятие, подобно понятию «множество» или «точка».

К счастью, точное определение в большинстве случаев и не нужно. Сотни поколений успешно учили геометрию по Евклиду, довольствуясь весьма приблизительными определениями основных понятий: «Точка есть то, что не имеет частей», «Линия есть длина без ширины», «Прямая линия есть та, которая одинаково расположена относительно всех своих точек» и т.п. А выда-

ющееся открытие математики XX в. состояло в том, что строго определить эти понятия невозможно, а главное — и не нужно.

Для практического освоения геометрии в школе достаточны интуитивные представления о точках, линиях и плоскостях. Точно так же для практического освоения понятия «информация» достаточно развить интуитивные представления об этом понятии.

В популярных книгах можно встретить различные определения информации. Если попытаться их проанализировать строго, то выяснится, что эти определения опираются на другие понятия: «сведения», «данные», являющиеся, в сущности, синонимами определяемого понятия. Это вовсе не означает, что эти определения бесполезны. Каждое из них способно создать у читателя новые интуитивные представления, показать новые оттенки этого понятия.

8.2. Информационный объем сообщения

В разговорном языке часто встречаются такие выражения, как «передача информации», «сжатие информации», «обработка информации». Здесь всегда подразумевается некое ***сообщение***, закодированное и переданное тем или иным способом. Обычно для кодировки сообщения выбирается какой-то набор допустимых знаков — ***алфавит***, и сообщение представляет собой последовательность знаков этого алфавита. Сообщение может содержать много информации, мало информации, а может и не содержать никакой информации вообще. Так называемая ***информативность*** сообщения зависит от многих причин. Например, от особенностей того, кто это сообщение принимает: для человека, не знакомого с китайским языком, количество информации в любом китайском тексте равно нулю.

Попытки точного определения термина «информативность сообщения» и введения единиц измерения информативности наталкиваются на большие трудности. Есть несколько разных подходов к этой проблеме.

Один подход состоит в том, что информация — это физическая величина, которая, подобно энергии, может преобразовываться из одной формы в другую и может быть измерена с помощью физических приборов.

> Сторонники такого подхода могли бы сказать, что количество информации в картинке, напечатанной игольчатым принтером, можно оценить по температуре печатающей головки. Если головка при печати разогрелась мало, то в картинке было мало информации.

Этот подход хотя и интересен, но бесполезен для практических применений.

Мы будем использовать другой подход. Будем считать, что существуют две меры для количества информации: внешняя (техническая) и внутренняя (семантическая, или смысловая). Технически количество информации в сообщении пропорционально количеству переданных символов и называется ***информационным объемом сообщения***.

> В настоящее время общепринято передавать сообщения в виде последовательностей символов–байтов и измерять информационный объем в байтах и производных единицах: килобайтах, мегабайтах и гигабайтах.

Информационный объем сообщения легко измерить, подсчитав количество символов сообщения. Этот подсчет не требует рассмотрения содержания сообщения. При этом подсчете смысл сообщения не играет никакой роли.

> Можно провести аналогию между информационным объемом сообщения и объемом железнодорожного вагона. Если из пункта А в пункт Б послан груз в вагоне, вмещающем 120 $м^3$, то ясно, что реальный объем груза в вагоне будет не больше 120 $м^3$. Реальный объем посланного груза может быть и меньше этой цифры, и вообще может оказаться равным нулю.

Семантическое, или смысловое, количество информации в основном рассматриваться не будет. В разных случаях это количество можно определить по-разному. Единого подхода, охватывающего все случаи, не существует. В пункте 8.3 разобран один способ определения «внутреннего» количества информации в сообщении.

8.3. Определение количества информации по Колмогорову

Одно из наиболее известных определений внутреннего количества информации в сообщении принадлежит великому русскому математику XX в. А. Н. Колмогорову. Идея этого определения очень простая. Предположим, что нужно передать сообщение по телеграфу, и, безотносительно к содержанию телеграммы, оплачивается каждая переданная буква. Чтобы платить поменьше, закодируем каким-то образом сообщение так, чтобы оно стало покороче, и пошлем закодированное сообщение. Чтобы получатель смог прочесть закодированное сообщение, пошлем ему еще и программу расшифровки. (Предполагается, что получателю заранее

известно, на каком языке записана эта программа, и есть договоренность, что получатель поместит полученное сообщение в файл с именем шифр, откуда его будет читать программа расшифровки.) При таком подходе достаточно будет заплатить за передачу зашифрованного сообщения и программы расшифровки. Основная идея Колмогорова в том, что тогда придется платить меньше, чем если бы передавалось исходное сообщение:

сложностью сообщения, по Колмогорову, называется минимальная плата по всевозможным методам шифровки этого сообщения.

Разберем один пример. Пусть в сообщении перечислены подряд все шестизначные десятичные числа, начиная с числа 262 144 и кончая числом 763 505. Закодируем это сообщение указанными двумя числами — это 13 символов (учитывая пробел между числами), гораздо короче, чем исходное сообщение. Программа расшифровки этого сообщения на школьном алгоритмическом языке может быть записана так:

```
алг расшифровка
нач цел m,n,i
| начать чтение("шифр")
| ввод m,n
| кончить чтение
| нц для i от m до n
| | вывод i
| кц
кон
```

Запись программы займет еще 124 байта (включая управляющие символы концов строк). Информационный объем исходного сообщения больше 500 тыс. символов, но нам удалось переслать его получателю, послав всего 13+124 = 137 символов. Значит, количество информации по Колмогорову в исходном сообщении не превосходит 137 и по сравнению с информационным объемом сообщения ничтожно мало. Возникает вопрос, минимально ли число 137? Его можно еще уменьшить, назвав алгоритм одной буквой и записав три команды в одной строке:

```
алг р
нач цел m,n,i
| начать чтение("шифр");ввод m,n;кончить чтение
| нц для i от m до n
| | вывод i
| кц
кон
```


В записи этой программы 110 символов, но это еще не предел. Само зашифрованное сообщение можно включить прямо в программу, и тогда достаточно будет передать получателю только эту программу:

```
алг р
нач цел i
| нц для i от 262144 до 763505
| | вывод i
| кц
кон
```

Запись программы сокращена уже до 68 символов. Но и это не предел. В числе 262 144 есть скрытые закономерности, позволяющие записать его не шестью знаками, а короче. Это позволяет укоротить программу еще на 1 символ:

```
алг р
нач цел i
| нц для i от 2**18 до 763505
| | вывод i
| кц
кон
```

Упражнения:

19. Укоротите запись этой программы еще на 1 символ.

20. Можно ли еще сократить послание?

21. Докажите, что существует 1000-значное десятичное число, которое не может быть напечатано никакой программой длины меньше 333 символов.

Определение Колмогорова в основном представляет теоретический интерес. На практике его использовать трудно. Это определение ничего не говорит о том, как найти оптимальный алгоритм шифровки и совершенно не учитывает время, которое уйдет на расшифровку сообщения.

Говоря дальше о *количестве информации*, мы всегда будем иметь в виду техническую меру информации, информационный объем сообщения. Говоря же о самой *информации*, мы будем иметь в виду те интуитивные представления, которые есть у читателя.

8.4. Информация и человеческое общество

Человек в своей жизни, от рождения до самой смерти, постоянно встречается с разнообразной информацией. Все органы чувств заняты восприятием и преобразованием внешней информации, в том числе и поступающей при общении с другими людьми. Если лишить ребенка человеческого общения, то из него не вырастет полноценный человек. Общение с другими людьми, обмен информацией между людьми составляют основу жизни как отдельного человека, так и всего человеческого общества. Уровень развития технологий хранения и обработки информации характеризует уровень развития человеческого общества в целом. Посмотрим, как развивались в обществе такие технологии.

Изобретенный человечеством язык представляет собой универсальную систему для представления совершенно произвольной информации. Однако до изобретения письменности получение информации от другого человека требовало устного общения. Передача информации от поколения к поколению была затруднена.

Изобретение письменности решило эти проблемы. С появлением письменности появилась возможность сохранять накопленную информацию очень долго, передача информации стала возможной без личных контактов. Стало реальностью целенаправленное накопление информации, полученной несколькими поколениями. Изобретение письменности и появление рукописных книг решили ряд проблем по улучшению работы с информацией в человеческом обществе. Появились устройства для хранения информации — книги. Появились и организации, занимающиеся эксплуатацией этих устройств — библиотеки.

В процессе использования книг для хранения и передачи информации в обществе выявилось узкое место этой технологии — низкая скорость и дороговизна процессов копирования и тиражирования информации. (Тиражирование — это копирование с целью получения большого количества копий.) Трудность тиражирования удалось преодолеть изобретением книгопечатания. Книгопечатание привело к широкому распространению информации, накопленной в эпоху рукописных книг. Вся информация, получаемая человечеством, стала доступной широкому кругу людей, ее количество стало расти лавинообразно. Появились новые технологии организации информации: научные журналы, каталоги в библиотеках и т.д. К началу XX в. человечество начало захлебываться в море накопленной им информации. Дошло до того, что заново изобрести что-либо стало во многих случаях дешевле, чем найти сведения об уже сделанном изобретении.

Но человеческая мысль не стояла на месте. Ряд изобретений

и открытий XVIII—XIX вв. подготовили изобретение компьютера — универсальной машины для хранения и переработки информации. Компьютеры в современном обществе взяли на себя значительную часть работ, связанных с информацией. По историческим меркам компьютеры и компьютерные технологии обработки информации еще очень молоды и находятся в самом начале своего развития. Мы живем в переходный период. В обществе еще много потоков информации, не вовлеченных в сферу действия компьютеров. Читателям этой книги выпало жить в то время, когда этот переходный период стремительно завершается. Компьютерные технологии сегодня преобразуют или вытесняют старые докомпьютерные технологии обработки информации. Завершится этот этап в ближайшее время построением в индустриально развитых странах глобальных всемирных сетей для хранения и обмена информацией, доступных каждой организации и каждому члену общества.

8.5. Информация, компьютер и человек

Встраивание компьютера в жизнь человека и общества — непростое дело. Компьютеру следует поручать то, что он может делать лучше человека. Человек должен оставить за собой то, чего не может компьютер.

Если спросить, каковы основные достоинства компьютера, то все скажут, что он умеет быстро считать. Кое-кто добавит, что компьютер может, не ошибаясь, выполнять долгие, монотонные, однообразные работы. Все это так, но главное свойство компьютера — возможность автоматического исполнения любых алгоритмов обработки информации.

Именно компьютеры позволили создать новые инструменты накопления, хранения и поиска информации — ***базы данных***. Именно компьютеры позволили обеспечить доступ к базам данных и другой информации на расстоянии. Именно компьютеры позволили человечеству замахнуться на выполнение грандиозной задачи по созданию всемирной информационной системы.

Информация должна не только обрабатываться внутри компьютера или путешествовать по компьютерным сетям от машины к машине. Она должна поступать из внешнего мира в компьютер и из компьютера во внешний мир. В частности, человек должен уметь вводить информацию в компьютер и получать ее от компьютера.

На заре развития компьютеров для ввода информации применялись перфокарты и перфоленты. Это был двухэтапный про-

цесс. Вводимые данные сначала кодировались путем пробивания отверстий в бумажной карте или ленте, а затем поступали в считывающее устройство, откуда и попадали в компьютер. Потом появились так называемые *консоли* — устройства, напоминающие пишущую машинку. Они позволяли довольно быстро вводить текст в компьютер и читать текст, напечатанный по командам компьютера на бумаге. Чуть позже появился *дисплей*, состоящий из *клавиатуры*, на которой набирался текст для ввода в компьютер, и *монитора* — телевизионной трубки, на которую компьютер выводил текст.

Самое популярное устройство вывода информации в наши дни — цветной графический монитор, позволяющий компьютеру выводить цветные изображения. Для ввода информации человек сегодня чаще всего использует *клавиатуру* и *мышь*. Для вывода информации на бумагу используются *принтеры*, для ввода — *сканеры* и *видеокамеры*. Все чаще используются речевые ввод и вывод.

Трудно предсказать, что станет общепринятым завтра. Появились переносные компьютеры без клавиатуры, которые могут распознавать и вводить рукописный текст. Изображение можно выводить на *инфошлем* — два миниатюрных экрана перед глазами создают стереоизображение. *Инфоперчатки* могут передавать в компьютер положения пальцев человека и, получая информацию от компьютера, оказывать сопротивление движениям человека. *Инфоскафандры* способны воспринимать положение тела человека и, по командам компьютера, имитировать прикосновение или давление на кожу человека. Все эти *инфоустройства* позволяют создавать так называемые ***искусственные реальности***, где человек оперирует в воображаемом, созданном компьютером мире, получая через свои органы чувств соответствующие комплексы ощущений.

8.6. Необходимость кодирования информации

Информация никогда не появляется в «чистом виде», она всегда как-то представлена, как-то ***закодирована***. Одна и та же информация может быть представлена различными способами. Количество учеников в классе может быть представлено в виде рисунка, буквенной записи, числа. При этом разнятся лишь способы представления, а сама информация остается неизменной. Не нужно думать, что выбор способа представления — это что-то второстепенное. От того, как представлена информация, зависит очень

многое. Так что в практических задачах важно не только найти *какой-нибудь* способ представления, но и выбрать тот способ, который наиболее удобен.

В параграфе 3 мы познакомились с системами счисления — способами кодирования чисел. Числа дают информацию о количестве предметов. Эта информация должна быть закодирована, представлена в какой-то системе счисления. Какой из известных способов выбрать, зависит от решаемой задачи. Иногда удобна двоичная система, иногда десятичная, а иногда и смешанная система.

До недавнего времени на компьютерах в основном обрабатывалась числовая и текстовая информация. Но большую часть информации о внешнем мире человек получает в виде изображения и звука. При этом более важным оказывается изображение. Помните пословицу: «Лучше один раз увидеть, чем сто раз услышать». Поэтому сегодня компьютеры начинают все активнее работать с изображением и звуком. Способы кодирования такой информации будут обсуждаться в следующем параграфе.

§ 9. ДВОИЧНОЕ КОДИРОВАНИЕ ИНФОРМАЦИИ

В параграфе 3 при записи чисел основание системы счисления указывалось внизу после числа:

$$18_{10} = 10010_2 = 12_{16}.$$

В этом параграфе встречаются только десятичные, двоичные и шестнадцатеричные числа, и различаться они будут по-другому. Все числа, набранные обычным шрифтом, будут считаться десятичными. Для двоичной записи будут использоваться курсивные цифры *0* и *1*, а для шестнадцатеричной — шрифт «пишущей машинки»: 0123456789ABCDEF. Например: 18 =*10010*= 12.

9.1. Почему все-таки двоичное кодирование?

Для кодирования информации нужен набор элементарных символов, из которых будут строиться последовательности символов — сообщения. Наименьшее количество таких элементарных символов — два. Кодирование с помощью всего двух символов называется ***двоичным кодированием***. Эти два символа принято обозначать *0* и *1* и называть двоичными цифрами, или ***битами*** (*англ.* bit — binary digit — двоичная цифра).

Таким образом, двоичное кодирование — это кодирование с минимально возможным числом элементарных символов, кодирование самыми простыми средствами. Тем оно и замечательно с теоретической точки зрения.

Инженеров двоичное кодирование информации привлекает тем, что легко реализуется технически. Электронные схемы для обработки двоичных кодов должны находиться только в одном из двух состояний: *есть сигнал/нет сигнала* или *высокое напряжение/низкое напряжение*. А раз состояний только два, то их легко различать, схему легко переключать из одного состояния в другое.

Еще двоичное кодирование нравится инженерам тем, что оно автоматически дает способ кодировки чисел в двоичной системе счисления. В такой системе особенно легко выполнять арифметические операции. Таблица сложения выглядит следующим образом:

$$0+0=0 \quad 1+0=1$$
$$0+1=1 \quad 1+1=10$$

Таблица умножения еще проще:

$$0\times0=0 \quad 1\times0=0$$
$$0\times1=0 \quad 1\times1=1$$

То, что числа в двоичной системе счисления получаются слишком длинными, инженеров не очень беспокоит. В компьютере легче иметь дело с большим числом простых элементов, чем с небольшим числом сложных.

При хранении информации в памяти компьютера каждый бит хранится в одном ***разряде*** памяти, а разряды объединяются в ***ячейки памяти*** фиксированного размера.

Ячейки размером 8, 16 и 32 разряда носят специальные названия *байт*, *слово* и *двойное слово*, и фактически эти три размера являются стандартом для современных персональных компьютеров.

9.2. Двоичное кодирование чисел

Над числами, размещенными в байтах, словах и двойных словах, компьютер проводит арифметические действия особенно быстро. Но не всякое число может быть записано в один байт после перевода его в двоичную систему. Например, 1000 не удастся поместить в байт, поскольку его двоичная запись содержит 10 цифр (*1111101000*), что больше размера байта. Число 9876543210 не поместится даже в двойное слово. Большие числа можно размещать в большем числе байтов, но операции над ними придется сводить к операциям над байтами, словами и двойными словами.

Если на бумаге выписаны несколько целых двоичных чисел подряд, то кроме двоичных цифр *0* и *1* приходится еще использовать знак «–» для отрицательных чисел и пробел для отделения чисел друг от друга. Причем числа могут состоять из любого числа двоичных цифр. Но при двоичном кодировании на компьютере в ячейки памяти нельзя записать ничего, кроме двоичных цифр, и нужно как-то договориться о кодировании отрицательных чисел и чисел, которые не занимают целиком одну ячейку.

Кодирование нуля и положительных чисел

Число ноль записывается одной цифрой *0*. Как же записать ноль в байте, слове или двойном слове? Очень просто, его нужно записать в виде последовательности из 8, 16 или 32 нулей. Вообще, при записи чисел в позиционной системе счисления слева от числа можно дописывать любое число нулей. Обычно такие нули называют ***незначащими***, поскольку они и в самом деле ничего не означают. Например, число 1965 можно записать как 00001965. Это свойство позиционной записи используется в компьютерах — положительные числа дополняются слева нулями до размера ячейки.

Кодирование отрицательных чисел

Для записи отрицательных чисел чаще всего используется ***дополнительный код***. Покажем, что это такое, на примере записи числа в 8-разрядную ячейку — байт. Запись в ячейки другого размера проводится аналогично.

Подсчитаем вначале, сколько комбинаций нулей и единиц можно записать в 8-разрядную ячейку.

В первом разряде может быть *0* или *1* — два варианта. Каждый из них можно скомбинировать с *0* или *1* во втором разряде — итого для двух разрядов возможны 4 варианта. Каждый из этих вариантов, в свою очередь, можно скомбинировать с *0* или *1* в третьем разряде — итого для трех разрядов возможно $8 = 2^3$ вариантов. Для восьми разрядов возможно $256 = 2^8$ вариантов. Из этих 256 комбинаций одна должна кодировать ноль, примерно половина оставшихся — положительные числа, а остальные комбинации — отрицательные числа.

Разумеется, ноль будет кодироваться комбинацией из 8 цифр *0*: *00000000*. Положительные числа от 1 до 127 будем кодировать

как обычно, в двоичной системе, дописывая слева незначащие нули:

1	*0000 0001*
2	*0000 0010*
3	*0000 0011*
...	...
126	*0111 1110*
127	*0111 1111*

С отрицательными числами поступим так. Пусть надо записать в байт (т.е. в 8-разрядную ячейку) число -5. Прибавим к этому числу $256 = 2^8$ и запишем в ячейку полученное положительное число: *1111 1011*.

Поскольку $-5+256 > 127$, в старшем разряде полученного числа будет единица. Эта единица и будет сигнализировать о том, что число отрицательное. Таким образом удастся закодировать числа -1, -2, -3,... -127, -128. Каждое из них после прибавления 256 будет больше 127, т.е. будет иметь *1* в старшем разряде.

Таким образом, мы научились записывать в байт ноль, 127 положительных чисел и 128 отрицательных чисел. Название *дополнительный код* означает, что -5 записывается как $256 - 5$, т.е. как *дополнение* к 5 до 256.

Единица в старшем разряде дополнительного кода означает, что число отрицательное. Поэтому старший разряд ячейки обычно называют *знаковым*.

Вопрос. Верно ли, что *1000 0001* кодирует отрицательное число -1?

Ответ. Эта комбинация действительно кодирует отрицательное число. Но не -1, а -127, так как $-127 + 256 = 129 = 128 + 1$, что в двоичной системе записывается как *1000 0001*.

Вопрос. Как записать в байт в дополнительном коде числа 128 и -129?

Ответ. Ни одно из этих чисел записать в байт в дополнительном коде нельзя. Записать в байт можно только числа от -128 до 127.

Диапазоны чисел, которые можно записывать в ячейки разных размеров, следующие:

Байт

от	$-128 = -2^7$	*1000 0000*
до	$127 = 2^7 - 1$	*0111 1111*

Слово

от	$-32768 = -2^{15}$	*1000 0000 0000 0000*
до	$32767 = 2^{15} - 1$	*0111 1111 1111 1111*

Двойное слово

от	$-2147483648 = -2^{31}$	*1000000000000000000...0000*
до	$2147483647 = 2^{31} - 1$	*0111111111111111111...1111*

32 разряда

Если в процессе вычислений на компьютере получается число, которое нельзя записать в предназначенную для него ячейку, то говорят, что возникло ***переполнение***.

Знаковые и беззнаковые ячейки

В некоторых случаях заранее известно, что кодируемые числа всегда неотрицательны. Тогда можно использовать ***беззнаковые ячейки***, в которые записываются только неотрицательные числа. При этом старший разряд теряет смысл знакового, а максимальное положительное число, которое можно записать в такую ячейку, становится примерно в два раза больше. В байт, например, записываются числа от 0 до 255.

Упражнения:

22. Записать в 8-разрядную знаковую ячейку следующие числа: 0, 1, 20, 64, -1, -2, -127.

23. Можно ли записать в 8-разрядную знаковую ячейку числа: 100, 200, -100, -200? Тот же вопрос для беззнаковой ячейки.

9.3. Кодирование текста

Если мы научились кодировать числа, ничего не стоит закодировать и текст. Для этого достаточно как-нибудь перенумеровать все буквы, цифры, знаки препинания и другие используемые при письме символы. Теоретически порядок нумерации совершенно не важен, но на практике он должен быть «известен» компьютеру, чтобы обеспечить правильную передачу информации человеку и другим компьютерам.

В большинстве современных компьютеров для хранения символа отводится 8-разрядная ячейка (байт). В байт можно записать всего 256 различных чисел — это позволяет закодировать 256 разных символов. Соответствие символов и их кодов при данном способе кодировки задается с помощью таблицы, в которой для каждого кода указывается соответствующий символ.

В России есть несколько распространенных способов кодировки. На компьютерах IBM PC обычно используется кодировка с неуклюжим названием ***альтернативная модифицированная***. Таблица для этой кодировки приведена на следующей странице.

20		30	0	40	@	50	P	60	`	70	p	80	А	90	Р	A0	а	E0	р
21	!	31	1	41	A	51	Q	61	a	71	q	81	Б	91	С	A1	б	E1	с
22	"	32	2	42	B	52	R	62	b	72	r	82	В	92	Т	A2	в	E2	т
23	#	33	3	43	C	53	S	63	c	73	s	83	Г	93	У	A3	г	E3	у
24	$	34	4	44	D	54	T	64	d	74	t	84	Д	94	Ф	A4	д	E4	ф
25	%	35	5	45	E	55	U	65	e	75	u	85	Е	95	Х	A5	е	E5	х
26	&	36	6	46	F	56	V	66	f	76	v	86	Ж	96	Ц	A6	ж	E6	ц
27	'	37	7	47	G	57	W	67	g	77	w	87	З	97	Ч	A7	з	E7	ч
28	(	38	8	48	H	58	X	68	h	78	x	88	И	98	Ш	A8	и	E8	ш
29	)	39	9	49	I	59	Y	69	i	79	y	89	Й	99	Щ	A9	й	E9	щ
2A	*	3A	:	4A	J	5A	Z	6A	j	7A	z	8A	К	9A	Ъ	AA	к	EA	ъ
2B	+	3B	;	4B	K	5B	[	6B	k	7B	{	8B	Л	9B	Ы	AB	л	EB	ы
2C	,	3C	<	4C	L	5C	\	6C	l	7C	|	8C	М	9C	Ь	AC	м	EC	ь
2D	-	3D	=	4D	M	5D	]	6D	m	7D	}	8D	Н	9D	Э	AD	н	ED	э
2E	.	3E	>	4E	N	5E	^	6E	n	7E	~	8E	О	9E	Ю	AE	о	EE	ю
2F	/	3F	?	4F	O	5F	_	6F	o			8F	П	9F	Я	AF	п	EF	я

Не пугайтесь, для большинства работ на компьютере эта таблица не нужна, и заучивать ее не придется.

Коду 20 в этой таблице соответствует *пробел* — пустой промежуток шириной в один символ, который можно использовать для разделения слов.

> Пусть вас не удивляет, что, говоря о двоичном кодировании, мы приводим шестнадцатеричные коды символов. В пункте 3.17 уже говорилось, что двоичная и шестнадцатеричная системы счисления являются родственными. Шестнадцатеричная система значительно удобнее для человека — для записи любого значения от 0 до 255 требуется не 8, а всего две «цифры». С другой стороны, переход к двоичным кодам может быть легко выполнен в уме.

Хотя в России используются несколько систем кодировки, все эти системы одинаково кодируют латинские буквы, цифры, знаки препинания и другие специальные знаки, скажем: +, – и т.п. Дело в том, что кодирование этих символов описывается стандартом США под названием ASCII, что означает *американский стандартный код для обмена информацией*. Этот стандарт в наши дни фактически стал мировым стандартом.

Упражнения:

24. Закодировать при помощи приведенной выше таблицы:

а) номер своего класса;

б) свою фамилию;

в) строку «`f(x+1)`».

25. Раскодировать: `A4 AE 23`.

9.4. Азбука Морзе

Иногда в качестве примера двоичного кодирования приводится ***азбука Морзе***. На первый взгляд буквы и знаки препинания в азбуке Морзе кодируются при помощи последовательностей всего двух символов — точки и тире (короткого и длинного сигналов):

АА ·—	ИИ ··	РР ·–·	Ш ––––
ББ –···	ЙЙ ·–––	SC ···	ЩЩ ––·–
WB ·––	КК –·–	TT –	ХЬ –··–
ГГ ––·	ЛЛ ·–··	UУ ··–	ЫЫ –·––
ДД –··	ММ ––	FФ ··–·	Э ··–··
ЕЕ ·	NН –·	НХ ····	Ю ··––
VЖ ···–	ОО –––	СЦ –·–·	Я ·–·–
ЗЗ ––··	РП ·––·	Ч –––·	

Однако азбука Морзе не является на самом деле двоичным кодированием. В ней, кроме точек и тире, используется третий «символ» — пауза между буквами. Если записывать символы азбуки Морзе подряд без пауз, то сообщения нельзя будет однозначно раскодировать. Например, начало сообщения – – – – · – · – · может означать либо – – –– (ш) ·– (а) · – · (р), либо – – – (о) –· (н) ·– (а)...

Упражнения:

26. Придумайте способ кодирования последовательности точек и тире азбуки Морзе одним числом. Можно ли для решения этой задачи использовать двоичную систему счисления?

27. Придумайте два слова, которые при записи их азбукой Морзе без пробелов выглядят одинаково.

28. Закодируйте буквы от А до Я (без Е и Ъ) точками и тире так, чтобы их можно было передавать без промежутков.

9.5. Кодирование изображений

Практически все современные компьютеры являются цифровыми — они хорошо работают с числами, но не умеют обрабатывать непрерывные величины. К счастью, человеческий глаз можно обмануть: изображение, составленное из большого числа отдельных мелких деталей, воспринимается как непрерывное.

Разобьем картинку вертикальными и горизонтальными линиями на маленькие прямоугольники. Полученный двухмерный массив прямоугольников называется ***растром***, а сами прямоугольники — ***элементами растра***, или ***пикселами*** (это слово

произошло от английского picture's element — элемент картинки). Теперь осталось закодировать числами цвета каждого пиксела — и задача кодирования изображения будет решена: закодированные цвета пикселов, перечисленные по порядку (например, слева направо и сверху вниз), и будут кодировать картинку.

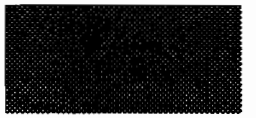
Разумеется, часть информации о картинке при таком кодировании потеряется. Потери будут тем меньше, чем мельче прямоугольники и чем точнее закодирован цвет каждого из них.

Осталось разобраться, как же кодировать цвет элемента изображения. Заметим, что, во-первых, в понятие цвет элемента включается и его яркость. Во-вторых, для единообразия говорят и о цветах черно-белого изображения. В этом случае цвет (оттенок серого цвета) просто сводится к яркости.

Кодирование черно-белых изображений

Яркость описывается одним числом. Для кодирования яркости пикселов отводятся ячейки фиксированного размера, чаще всего от 1 до 8 бит; черный цвет кодируется нулем, а чисто белый — максимальным числом N, которое может быть записано в ячейку. Для одноразрядной ячейки $N = 1$, а для 8-разрядной $N = 255$. Для практических приложений 8-разрядных ячеек вполне достаточно (человеческий глаз в состоянии различить не более одной-двух сотен разных оттенков серого цвета).

Если же по тем или иным причинам выбрано $N = 1$, т.е. каждый элемент изображения имеет либо черный, либо белый цвет, то полутона приходится имитировать. Например, серый прямоугольник справа состоит из чередующихся черных и белых пикселов. В общем случае, пусть какая-то часть картинки должна иметь серый цвет, средний между чисто черным и чисто белым. Тогда на этой части картинки из каждых четырех соседних пикселов сделаем два черными и два белыми. Глазу будет казаться, что эта часть картинки имеет серый цвет.



Кодирование цветных изображений — метод RGB

Для цветных изображений дело обстоит сложнее. Человеческий глаз различает огромное количество разных цветов и оттенков, которые не так просто закодировать одним числом. Однако все не так страшно.

Дело в том, что глаз воспринимает все цвета как сумму трех основных цветов — красного, зеленого и синего. Например, сиреневый цвет — это сумма красного и синего, желтый цвет — сумма красного и зеленого, оранжевый — тоже сумма красного и зеленого, но в другой пропорции. Поэтому можно закодировать цвет пиксела тремя числами — яркостью его красной, зеленой и синей составляющих.

Этот способ кодирования цветов называется RGB — по первым буквам английских слов Red, Green, Blue — красный, зеленый, синий.

Другие способы кодирования цветов

При рисовании на бумаге действуют другие законы (поскольку краски сами по себе не испускают света, а только поглощают некоторые цвета из падающего на них света). Если смешать красную и зеленую краски, то получится не желтый цвет, а коричневый. Поэтому на печатающих устройствах обычно используются в качестве основных голубой, сиреневый и желтый цвета (такой метод кодирования цвета называется CMY — Cyan, Magenta, Yellow). Красный цвет получается как сумма сиреневого и желтого, а зеленый — как сумма желтого и голубого.

Описанные выше методы кодирования при помощи сложения основных цветов просты в реализации, но работать с ними не очень удобно. Поэтому во многих программах обработки изображений используется более удобная для человека схема кодирования «цветовой тон/насыщенность/яркость». Она называется HSV (по первым буквам английских слов Hue, Saturation, Value). При этом цвет каждой точки также описывается тремя числами, но их значения уже не те, что в методах кодирования RGB и CMY.

Упражнения:

29. Предложите способ кодирования трех чисел (каждое из которых лежит в интервале от 0 до 7) одним числом. Каким будет алгоритм декодирования?

30. Пусть известно, что картинка состоит из непересекающихся прямоугольников разного размера и цвета со сторонами, параллельными краям картинки. Придумайте способ кодирования таких картинок на компьютере.

9.6. Кодирование звуков

Из курса физики известно, что звук представляет собой колебания воздуха. Амплитуда этого колебания непрерывно меняется со временем. По своей природе звук является непрерывным сигналом. Для кодирования звука надо этот непрерывный сигнал

превратить в последовательность нулей и единиц. Это можно сделать так.

С помощью микрофона звук можно превратить в колебания электрического тока. Будем измерять амплитуду колебаний через равные промежутки времени (на практике — несколько десятков тысяч раз в секунду). Каждое измерение сделаем с ограниченной точностью и запишем в двоичном виде. Этот процесс называется ***дискретизацией***.

Устройство для выполнения дискретизации называется аналого-цифровым преобразователем (АЦП). АЦП измеряет электрическое напряжение в каком-то диапазоне и выдает ответ в виде многоразрядного двоичного числа. Например, типичный 8-битовый АЦП преобразует напряжения в диапазоне [−500мВ, 500мВ] в 8-разрядные двоичные числа в диапазоне [−128, +127].

Воспроизведение закодированного таким образом звука производится при помощи цифро-аналогового преобразователя (ЦАП). Двоичные числа, кодирующие звук, подаются на вход ЦАП с точно такой же частотой, как и при дискретизации, и ЦАП преобразует их электрические напряжения обратно тому, как это делал АЦП. Например, двоичные числа из диапазона [−128, +127] преобразуются в напряжения из диапазона [−500мВ, 500мВ]. Полученный на выходе ЦАП ступенчатый сигнал сначала сглаживается с помощью аналогового фильтра, а затем преобразуется в звук с помощью усилителя и динамика.

При работе со стереозвуком все это проводится отдельно и независимо для левого и правого каналов.

На качество воспроизведения закодированного звука влияют в основном два параметра: частота дискретизации и ее ***разрешение*** — размер ячейки, отводимый под запись значения амплитуды.

Например, при записи на компакт-диски (CD) используются 16-разрядные значения, а частота дискретизации равна 44 032 Гц. Эти параметры обеспечивают превосходное качество звучания речи и музыки.

Выбор частоты дискретизации объясняется тем, что максимальная частота звука, который еще слышит человек, не превосходит 22 кГц. Колебание с частотой 22 кГц при дискретизации с той же частотой превратится в постоянный сигнал. Чтобы удержать при дискретизации информацию о колебании в 22 кГц, на каждом периоде должно записываться, по крайней мере, два значения. То есть нужна вдвое большая частота дискретизации, а именно 44 кГц. Эта частота обеспечивает запись любых слышимых человеком звуков. В тех случаях, когда столь высокое качество не требуется, можно использовать

меньшие частоты дискретизации: 11 кГц, 5,5 кГц и т.д. Чтобы первые частоты, получаемые последовательным делением исходной частоты вдвое, оказались целыми, удобно взять исходную частоту в виде произведения целого числа на степень двойки. Этим и объясняется выбор частоты $172 \times 2^8 = 44032$ Гц.

Однако во многих случаях качество CD не требуется. Для записи и передачи речи достаточна частота дискретизации 8 кГц. Несмотря на то, что все составляющие человеческого голоса с частотой свыше 4 кГц не могут быть зарегистрированы при такой частоте дискретизации, закодированную речь легко понять.

9.7. Цифровая обработка звука

Основное достоинство использования компьютеров в работе со звуком заключается в том, что закодированный звук можно не только хранить на компьютере, но и обрабатывать его. Конечно, сложные виды обработки требуют применения сложных алгоритмов, но некоторые изменения можно сделать очень просто. Например, можно легко модифицировать звук так, чтобы в начале записи он не возникал резко, а плавно нарастал.

```
алг нарастание звука в начале записи
 дано | в файле in.xxx записан звук с частотой
      | дискретизации 22016 Гц
 надо | в файле out.xxx записан тот же звук,
      | плавно нарастающий в первую секунду
нач цел i,x
| начать чтение("in.xxx")
| начать запись("out.xxx")
| i:=1
| нц пока не конец файла
| | ввод x
| | если i<22016 то x:=x*i/22016 все
| | вывод x
| | i:=i+1
| кц
| кончить запись
| кончить чтение
кон
```

При работе этого алгоритма первые 22 016 значений звука умножаются на множитель i/22016, который в течение первой секунды возрастает от 0 до 1. Следующие значения звука переписываются без изменений.

Упражнения:

31. В файле `in.xxx` записаны `N` значений дискретизации некоторого звука. Составьте алгоритм, который записывает в файл `out.xxx` тот же звук, плавно и равномерно затухающий до нуля от начала записи к ее концу.

32. В таблицах `зв1[1:N]` и `зв2[1:N]` записаны два сигнала. Как получить в таблице `зв3[1:N]` сигнал, являющийся их наложением?

9.8. Два способа кодирования музыки

С физической точки зрения музыка является, конечно, звуком. И, следовательно, она может кодироваться как всякий другой звук. Так что первый способ кодирования музыки состоит в том, чтобы забыть, что дело идет о музыке, и закодировать звук, как это было описано выше. Такой способ кодирования музыки имеет два недостатка.

Во-первых, для записи качественной дискретизации музыки требуется очень много места (позднее будет подсчитано, сколько именно). Во-вторых, такую запись трудно обрабатывать (конечно, можно без труда добавить плавное затухание или сделать наложение записей разных инструментов, но изменить тональность мелодии или тембр уже не так просто).

К счастью, человечество уже давно придумало способ компактной записи музыкальных произведений — нотную запись. Она-то и положена в основу второго способа кодирования музыки.

Посмотрим на нотную запись музыкального произведения с точки зрения исполняющего музыкальное произведение, скажем, с точки зрения пианиста. Нотная запись может рассматриваться как последовательность команд пианисту: нажать такую-то клавишу с такой-то силой и продолжать ее держать, нажать одновременно несколько клавиш, отпустить ранее удерживаемую клавишу, нажать педаль на такое-то время и т.п.

Если выписать все мыслимые команды пианисту, получится система команд воображаемого исполнителя Пианист. И фортепианное произведение можно будет записать как последовательность команд, которую должен выполнить этот исполнитель.

Аналогичную систему команд можно выписать для других музыкальных инструментов: гитары, трубы, барабана и т.д.

В 80-х годах нашего века появились электронные музыкальные инструменты — ***синтезаторы***, которые были способны воспроизводить звуки многих существующих музыкальных инструментов, а также и совсем новые для музыкантов звуки.

В 1983 г. ведущие производители электронных музыкальных синтезаторов и производители компьютеров договорились о системе команд универсального синтезатора, о том, какими электрическими сигналами будут подаваться такие команды, и даже о разъемах и кабелях, которые будут соединять компьютеры и синтезаторы. Это соглашение получило название *стандарт MIDI* (англ. Musical Instrument Digital Interface — описание цифрового музыкального инструмента). Этот стандарт дает и удобный способ кодирования музыки.

Запись музыкального произведения в формате MIDI есть не что иное, как программа игры на воображаемом музыкальном инструменте (он называется *синтезатором*). Такая запись состоит из последовательности закодированных *сообщений*, разделенных закодированными паузами. Сообщение может быть:

- командой синтезатору (нажать или отпустить определенную клавишу, изменить высоту или тембр звучания и т.д.);
- описанием параметров воспроизведения (например, значения силы давления на клавишу);
- управляющим сообщением (в их число входят, например, команда включения полифонического режима, синхронизирующее сообщение и т.д.).

При таком кодировании нельзя записать вокальные произведения, так как звуки, издаваемые певцом или хором, не входят в систему команд воображаемого синтезатора. В остальном такая система кодирования весьма удобна, так как:

- запись очень компактна и больше соответствует человеческому представлению о музыке;
- такие изменения, как смена инструмента или тональности мелодии, производятся очень легко;
- одну и ту же закодированную запись можно воспроизводить как на дорогом синтезаторе с очень высоким качеством звучания, так и на дешевой компьютерной плате (или даже на встроенном динамике).

9.9. Кодирование фильмов

Кодирование движущихся изображений кажется на первый взгляд очень сложной задачей. Однако, к счастью, человеческий глаз несовершенен. Для того чтобы создать у зрителя иллюзию движения, надо показывать ему быстро сменяющиеся картинки, на которых изображены последовательные фазы движения. Именно на этом принципе основаны кино и телевидение, на нем же основано и компьютерное кодирование фильмов.

Поскольку уже известны принципы кодирования отдельных кадров, ничего не стоит закодировать и последовательность таких кадров.

Что касается звукового сопровождения, то не имеет смысла придумывать какой-нибудь специальный способ кодирования. Можно просто записывать звук независимо от изображения (так, как это делается в кино).

При независимой записи изображения и звука важно добиться синхронности при воспроизведении. Для облегчения работы монтажера в кино применяется «хлопушка». Кроме того, что на ней написаны номера сцены и дубля, щелчок планкой позволяет легко совместить изображение и звук: момент щелчка хорошо виден на пленке.

Как правило, закодированный фильм в компьютере содержит заголовок с описанием параметров фильма, который включает:

- размер кадра в пикселах и количество используемых цветов;
- параметры звука (частота и разрешение);
- способ записи звука (раздельная запись к каждому кадру или непрерывная для всего фильма в целом).

После заголовка записывается последовательность закодированных картинок, соответствующих кадрам фильма, и закодированных звуковых фрагментов.

Упражнение:

33. Можно ли использовать второй способ кодирования музыки (по стандарту MIDI) в кодировании фильмов?

§ 10. ЕДИНИЦЫ ИЗМЕРЕНИЯ ИНФОРМАЦИИ

Ниже *количество информации* всюду будет рассматриваться в техническом смысле этого слова, как информационный объем сообщения или как объем памяти, необходимый для хранения сообщения без каких-либо изменений.

10.1. Биты и байты

В теории информации используется единица измерения количества информации, которая называется ***бит***. Информационный объем сообщения измеряется в битах и равен количеству двоичных цифр, которыми закодировано сообщение.

Однако в компьютерной практике слово *бит* используется также и как единица измерения объема памяти. По определению, ячейка памяти размером в один бит может находиться в двух состояниях и в нее может быть записана одна двоичная цифра. Восемь бит составляют ***байт***. В ячейку размером один байт можно записать восемь двоичных цифр (то есть всего $2^8 = 256$ различных значений).

Для измерения больших объемов памяти используются производные единицы — ***килобайт***, ***мегабайт*** и ***гигабайт***. В отличие от производных физических единиц, приставка *кило-* означает в компьютерной практике не 1000, а 1024 (это равно 2^{10}). Чтобы подчеркнуть это различие, в сокращенной записи используется не маленькая, а большая буква: километр записывается как «км», а вот килобайт — как «Кб». Мегабайт содержит 1024 килобайта или $1024 \times 1024 = 1\,048\,576$ байт и сокращенно записывается как «Мб». Гигабайт содержит 1024 мегабайта, т.е. около миллиарда байтов и сокращенно записывается как «Гб».

Различие между 1000 и 1024 довольно маленькое, и для проведения приближенных расчетов вполне можно полагать, что 1 Кб ≈ 1000 байт, 1 Мб $\approx 10^6$ байт и 1 Гб $\approx 10^9$ байт.

Многие прикидочные расчеты в информатике оперируют со степенями двойки. Для упрощения подобных расчетов полезно помнить, что $2^{10} \approx 10^3$. Если, например, нужно прикинуть, чему равно число 2^{32}, то по этой формуле получаем

$$2^{32} = 4 \times 2^{30} = 4 \times (2^{10})^3 \approx 4 \times (10^3)^3 = 4 \times 10^9.$$

10.2. Мегабайт — это сколько?

Чтобы не утонуть в море цифр и правильно представлять себе возможности современных компьютеров, важно иметь наглядные представления о том, сколько и какой информации способен вместить килобайт, мегабайт или гигабайт памяти. Следующие подсчеты заимствованы из школьного учебника А. Кушниренко и др. «Основы информатики и вычислительной техники».

При двоичном кодировании текста каждая буква, знак препинания, пробел — это 1 байт. На страницу обычного школьного учебника помещается чуть меньше 50 строк, в каждой строке — примерно 60 знаков (60 байт). Таким образом, полностью заполненная текстом страница учебника имеет информационный объем около 3000 байт (3 Кб). Средняя страница содержит около 2,5 Кб, а весь учебник — около 0,5 Мб текста. В одном номере четырехстраничной газеты — 150 Кб, а если собрать по одному номеру всех газет, выходящих в нашей стране, то в них будет уже ги-

габайт информации. Если человек говорит по 8 часов в день без перерыва, то за 70 лет жизни он наговорит около 10 гигабайт информации (это 5 млн страниц — стопка бумаги высотой 500 м).

Один черно-белый телевизионный кадр (при 32 градациях яркости каждой точки) содержит примерно 300 килобайт информации. Цветной кадр, образованный из трех кадров основных цветов (красный, синий, зеленый), содержит уже около мегабайта информации. А 1,5-часовой цветной телевизионный художественный фильм (при частоте 25 кадров в секунду) — 135 гигабайт.

Упражнения:

34. Вычислить, сколько байт информации занимает на компакт-диске одна секунда стереозаписи (частота 44 032 Гц, 16 бит на значение). Сколько занимает одна минута? Какова максимальная емкость диска (считая максимальную длительность равной 80 минутам)?

35. Оцените объем памяти, требуемый для хранения какого-нибудь школьного учебника. Сколько памяти надо для хранения всех учебников за все годы обучения? Сколько компакт-дисков потребуется для записи всех учебников?

36. Оцените, сколько места требуется для записи Большой Советской Энциклопедии.

§ 11. ПРОСТЕЙШИЕ СПОСОБЫ УПАКОВКИ И ШИФРОВАНИЯ ИНФОРМАЦИИ

Упаковка и ***шифрование*** информации (а точнее — сообщений, содержащих информацию) делаются с разными целями. Упаковка нужна для сокращения длины сообщения и уменьшения времени его передачи по каналам связи. Шифрование предназначено для обеспечения режима секретности при передаче сообщения. Между этими процессами, однако, есть и много общего. И тот и другой являются примерами процессов переработки информации. Часто эти процессы объединяются: упаковка одновременно обеспечивает и шифрование.

11.1. Как упаковать информацию

Все используемые методы упаковки информации можно разделить на два класса: упаковка без потери информации и упаковка с потерей.

В первом случае исходное сообщение можно точно восстановить по упакованному; во втором случае возможно внесение искажений (то есть распакованное сообщение будет отличаться от исходного). Обычно предполагается, что при упаковке с потерей информации искажения «малы» в том или ином смысле. Смысл этот уточняется для каждого конкретного алгоритма упаковки. Во многих случаях внесение искажений нежелательно или вообще недопустимо (в самом деле, достаточно изменить в программе всего один символ, чтобы она перестала работать). Однако при упаковке изображений или звука незначительные искажения допустимы, так как человек часто даже не в состоянии их заметить.

При упаковке кадров фильма можно позволить еще большие искажения, чем при упаковке статических картинок. Кадры меняются быстро, и зритель не успевает рассматривать их детально.

11.2. Упаковка без потери информации

На сегодняшний день разработано много алгоритмов упаковки информации без потерь, но практически все они основаны на одной из двух простых идей.

Первая идея — учет частот символов

Первая идея была впервые разработана Хафманом в 1952 г. и основана на том, что в обычном тексте частоты появления разных символов различны. При стандартном кодировании текста каждый символ кодируется одним байтом. Соответствие «один символ — один байт» упрощает обработку текста. Но перед длительным хранением (или перед передачей по каналам связи) можно позволить себе более сложную кодировку текста. При упаковке по ***методу Хафмана*** часто встречающиеся символы кодируются короткими последовательностями битов (короче 8), а более редкие — длинными (может быть, более 8 битов). В результате в среднем получается менее 8 битов на символ.

Рассмотрим простой пример. Предположим, что в тексте из 1000 байтов 50% пробелов. Закодируем этот текст, байт за байтом, одной длинной последовательностью нулей и единиц по такому правилу. Если в тексте встретился пробел, то допишем в конец последовательности *0*, а если встретился не пробел, то запишем в последовательность *1* и за ней — двоичный код этого символа, т.е. 1 + 8 нулей и единиц. Поскольку в тексте, по предположению, 500 пробелов, то на их кодировку тратится 500 раз по одному биту (заметная экономия по сравнению с кодировани-

ем каждого пробела 8 битами). А вот на каждый из 500 непробелов расходуется по 9 бит. Всего на весь текст будет потрачено $500 + 500 \times 9 = 5000$ бит, что значительно меньше, чем 8000 бит в исходном тексте.

Вторая идея — учет повторений

Вторая основная идея упаковки состоит в том, что в сообщениях часто встречаются несколько подряд идущих одинаковых байтов, а некоторые последовательности байтов повторяются многократно. При упаковке можно заменять такие места на специальные команды вида «повторить данный байт n раз» или «взять часть текста длиной k байтов, которая уже встречалась m байтов назад». При упаковке графической информации чаще встречается первая ситуация, а при упаковке текстов — вторая. (Две одинаковые буквы подряд в русском языке встречаются довольно редко, а повторяющиеся слоги и слова — довольно часто. Например, в данном абзаце сочетания букв «встреча» и «байт» встречаются 6 раз, сочетание «упаковк» — 5 раз, а сочетание «то» — 18 раз.)

11.3. Упаковка изображений без потери информации

Больше всего места требуется для хранения графической информации, поэтому задача ее упаковки наиболее важна. Графическая информация вообще очень редко хранится в компьютере в неупакованном виде.

Рассмотрим сначала один из простейших методов упаковки — ***RLE-кодирование*** (англ. Run-Length Encoding — кодирование путем учета числа повторений). Этот метод применим не только к изображениям, но и к произвольным сообщениям, и позволяет компактно кодировать длинные последовательности одинаковых байтов. Упакованная последовательность состоит из управляющих байтов, за каждым из которых следуют один или несколько байтов данных. Смысл управляющих байтов таков. Если старший (самый левый) бит управляющего байта равен *1*, то следующий байт данных надо при распаковке повторить несколько раз (сколько именно — записано в оставшихся семи битах управляющего байта). Например, управляющий байт *10000101* говорит, что следующий за ним байт нужно повторить 5 раз (так как двоичное число *101* равно 5). Если же старший бит управляющего байта равен *0*, то надо взять несколько следующих байтов данных

без всяких изменений. Сколько именно — тоже записано в оставшихся 7 битах. Например, управляющий байт *00000011* говорит, что следующие за ним 3 байта нужно взять без изменений.

Попробуем упаковать методом RLE последовательность байтов

11111111 11111111 11111111 11111111
11111111 11110000 00001111 11000011
10101010 10101010 10101010 10101010

В начале последовательности 5 раз подряд повторяется байт *11111111*. Чтобы закодировать эти 5 байтов, запишем сначала управляющий байт *10000101*, а затем сам повторяемый байт *11111111*. Выигрыш — 3 байта. Далее идут 3 разных байта *11110000 00001111 11000011*. Чтобы их закодировать, запишем управляющий байт *00000011* и затем эти 3 байта. Проигрыш — 1 байт. Далее в последовательности 4 раза повторяется байт *10101010*. Чтобы закодировать эти 4 байта, запишем управляющий байт *10000100* и повторяемый байт *10101010*. Выигрыш — 2 байта.

После кодировки получилась последовательность из 8 байтов:

10000101 11111111 00000011 11110000
00001111 11000011 10000100 10101010

В результате получается суммарный выигрыш в 4 байта, 12 байтов удалось упаковать в 8.

Упражнения:

37. Какова длина последовательности, после кодировки которой методом RLE получилось следующее:
11111111 11111111 11000000 00000001
00000010 11111111 00000000

38. Как надо поступить при RLE-кодировании, если количество одинаковых байтов больше 127 и не помещается в 7 разрядов?

39. Что будет, если в упакованном сообщении пропустить один байт?

11.4. Упаковка изображений с потерей информации

Методы упаковки картинок с потерей информации основываются на особенностях человеческого восприятия изображений. Например, для человеческого глаза информация о яркости более существенна, чем информация о цветовом тоне и насыщенности точки. Поэтому можно при упаковке выбросить данные о цвете

каждой второй точки изображения (сохранив только ее яркость), а при распаковке — брать вместо выброшенного цвет соседней точки. Распакованная картинка, конечно, будет отличаться от исходной, однако это отличие будет практически незаметно на глаз.

При таком методе упаковки экономия составляет менее 50%. Более сложные методы упаковки картинок позволяют добиться значительно лучших результатов. Например, алгоритм JPEG (названный так по названию разработавшей его группы — Joint Photographic Experts Group) способен упаковывать картинки в несколько десятков раз без заметной потери качества.

11.5. Компромисс между качеством упаковки и скоростью распаковки

За высокое качество упаковки, как правило, приходится платить большими затратами времени на распаковку. Чем изощреннее алгоритм упаковки, тем сложнее восстановить исходную информацию.

Алгоритмы, дающие очень хорошее качество упаковки, могут оказаться неприменимыми из-за слишком большого времени, требуемого на распаковку. Например, если время распаковки одного кадра фильма будет измеряться секундами, то смотреть такой фильм можно будет только после предварительной распаковки. Разработчики новых методов упаковки всегда ищут компромисс между качеством упаковки и скоростью распаковки.

Как правило, информацию можно «не торопясь» запаковать, а затем требуется многократно использовать. При этом время упаковки не так уж и важно. Но бывают случаи, когда информацию нужно паковать «на лету» или, как говорят, в ***режиме реального времени***. Такова , например, ситуация записи изображения с видеокамеры в память компьютера. В этом случае важна скорость алгоритмов упаковки.

11.6. Упаковка звука

Из всех видов информации, используемых в компьютерах, хуже всего поддается упаковке звук. Это связано с тем, что звуковые сигналы обладают малой избыточностью (в частности, в закодированных звуковых фрагментах редко появляются повторяющиеся последовательности байтов). Даже методы упаковки с потерей информации, как правило, не позволяют упаковать звук более чем в два раза без заметного снижения качества.

В два раза звук можно упаковать с помощью метода со странным названием ***компандирование***. Оно происходит от английского термина *compander*, что означает compressing/expanding coder/decoder. Этот метод основан на законе, открытом психологами:

если интенсивность раздражителя меняется в геометрической прогрессии, то интенсивность человеческого восприятия меняется в арифметической прогрессии.

Применительно к звуку это означает, что, если повышать громкость звука в 2, 4, 8 и т.д. раз, то человеческое ухо будет воспринимать это как линейное увеличение интенсивности. Отсюда следует, что изменение громкости от 1 до 2 столь же заметно человеку, как и изменение громкости от 100 до 200. А изменение громкости от 100 до 101 человеком практически не ощущается: младшие биты громкого звука менее существенны, чем те же биты для тихого звука.

Таким образом, фактически мы «слышим» логарифм громкости, а не саму громкость. (Разность $\log 2 - \log 1$ такая же, как разность $\log 200 - \log 100$, ибо обе они равны $\log 2$.)

Поэтому при компандировании значение амплитуды звука заменяется на логарифм этого значения. Полученные числа округляются и записываются в ячейки меньшего размера. Абсолютная величина амплитуды при 8-битном кодировании звука не превосходит 2^7, значит, логарифм по основанию 2 не превосходит 7 и может быть закодирован 3 двоичными разрядами. Еще один разряд уходит на знак амплитуды и получается 4 разряда на каждое значение амплитуды.

Практика показывает, что при компандировании 8-битового звука в 4-битовый происходит очень незначительное ухудшение качества звука. А такое кодирование сжимает информацию как раз вдвое.

11.7. Как спрятать информацию

В современном, сильно компьютеризованном обществе с необходимостью спрятать информацию сталкиваются не только шпионы. Передавать по глобальным сетям коммерческую или банковскую информацию можно только в зашифрованном виде.

В обыденной речи слова *кодирование* и *шифрование* часто смешиваются. Тем не менее смысл их различен: *кодирование* обычно означает перевод информации из одной формы представления в другую по некоторому известному всем алгоритму, а при *шифровании* алгоритм перевода держится в тайне.

Любопытно, что русское слово «шифр» похоже на слово «цифра». Это не случайно. В английском и французском языках одно из значений глагола «шифровать» — записывать цифрами. Так что «шифровать» означает «цифровать».

11.8. Шифрование простой подстановкой

Простейший способ спрятать информацию, известный задолго до появления компьютеров, — это заменять каждый символ на другой, пользуясь секретной таблицей. Например, в замечательном рассказе Эдгара По «Золотой жук» приведен такой пример зашифрованного текста:

53##†305)) 6*; 4826) 4#·) 4#); 806*; 48†8‡60)) 85; ;]8*; :#*8†83 (88) 5*†; 46(; 88*96*?; 8)*#(; 485); 5*†2:*#(; 4956*2(5*=4) 8‡8*; 406928 5);)6†8) 4##; 1(#9; 48081; 8:8#1; 48†85; 4) 485 †528806*81(#9; 48; (88; 4(#?34; 48)4#; 161; :188; #?;

Буквы латинского алфавита (сообщение, как и рассказ, записано по-английски) заменены некоторыми значками. На первый взгляд эта запись выглядит очень загадочно. Однако Вильям Легран, главный герой рассказа, утверждает: «Прямо скажу, если текст зашифрован без грубых ошибок и документ в приличной сохранности, я больше ни в чем не нуждаюсь; последующие трудности для меня просто не существуют».

Быть может, в этом утверждении есть некоторое преувеличение, но в целом оно соответствует действительности. Запись, полученную при помощи простой подстановки, действительно несложно расшифровать (если в нашем распоряжении есть достаточно длинный кусок зашифрованного текста). Для расшифровки надо воспользоваться тем, что в обычных текстах разные буквы встречаются с разными частотами.

Например, в текстах на русском языке буквы встречаются в среднем с частотами, приведенными в таблице:

о	0.090	в	0.038	з	0.016	ж	0.007
е,ё	0.072	л	0.035	ы	0.016	ш	0.006
а	0.062	к	0.028	б	0.014	ю	0.006
и	0.062	м	0.026	ь,ъ	0.014	ц	0.004
н	0.053	д	0.025	г	0.013	щ	0.003
т	0.053	п	0.023	ч	0.012	э	0.003
с	0.045	у	0.021	й	0.010	ф	0.002
р	0.040	я	0.018	х	0.009		

пробелы и знаки препинания 0.174

Это означает, что среди 1000 букв текста в среднем будет 174 пробела и знака препинания, 90 букв *о*, 72 буквы *е* и т.д. А вот букв *ф* в среднем встретится только 2.

Конечно, вовсе не обязательно, что самой редкой буквой любого текста будет буква *ф*. Скорее всего, в газетной статье под названием «Финские фальшивомонетчики на фондовом рынке Филиппин» буква *ф* будет встречаться чаще обычного.

Но в типичном тексте достаточной длины частоты первых нескольких символов будут близки к частотам первых символов приведенной выше таблицы.

11.9. Пример расшифровки текста, зашифрованного простой подстановкой

Пусть у нас есть достаточно длинный текст на русском языке, зашифрованный простой подстановкой:

ЦЗСЯЩ*Щ*ЯЙБЦЕРЯТЫЙЧГЕРЯЭХБЖЬШЮЗНЩБЯГЫЯЖЛФЧЮФЯОБГЫЦЗН
ЗГЯФЖБВБЧИЯЙЫА*ЯОБЙЧБЖЬШЬЯШФТЛЫЯЖЕЩФЦБЧИЯГЫЯЦЗН
ЫНЗЯЭХЬЦЫХЯЩХФНЬЦЯГБФЮБГЗЯЯАЗВЫЯЦЗСЯЯЮБЮБ*ЯЙЮФЮБ
ЙЯАЗШИГЕЦЯЙЬЦЫЧИЯЬЯЩЫГИЯЬЯГЗТИГЫЯЗЧРЗЩ*ЯГЬЯЛБНФЯЭХЗТИ
ЮБЮЗЫЯГЬОШЗЫЯЮЗЖБХЙЧЖЗЭЗШФВЬЖЗНЗЯОБАБЖШ*ЧИ
ЫЦФЯЭЗЩФЛЮЬЯЭЗЭХБЖШ*ЧИЭЫТБШИГЗЯЭЗЩГЗЙЬЧИЯШЫЮБХЙЧЖЗ
ЖОЩЕРБЧИЯЬЯЩФЦБЧИЯЭХЗЯЙЫА*ЯЮЗНЩБЯВЫЯТЫХЧЯЖЗОИЦЫЧЯЧЫА*
ЧБЮЯЩФЦБШЯЦЗШЗЩЗСЯЭЗЖЫЙБШЫЧ*ЯЖЯЭЕШЬЯГБЯЭЗТЧЗЖЕР
ЖЙЫЖЕЛГЫСЯЖЗШЫКЯОШЖЫЙБГБЙШЫЩГЬЮЯЖЙЫРЯЙЖЗЬРЯХЗЩГЕР
ЩХФОИ*ЯШКЩЦЬШЕЯЬЯХФЙШБГБЙЯНЫХЗЫЦЯЦЗЫНЗЯХЗЩБГБ
АЬОЯЭХЫЩЬЙШЗЖЬСЯЙЫСЯВЫЯТБЙЭЗОЖЗШИЧЫЯЭЗОГБЮЗЦЬЧИЯЖБЙ
ЗГЫНЬГЯЯЩЗАХЕСЯЦЗСЯЭХЬ*ЧЫШИХЗЩЬЙЙ*ЯГБЯАХЫНБРЯГЫЖЕ
НЩЫЯЯЦЗВЫЧЯАЕЧИЯЯХЗЩЬЬЙИЯЖЕЬЬЯАЬЙЧБШЬЯЯЦЗСЯТЬЧБЧЫШИ
ЧБЦЯГЫЮЗНЩБЯНФИ*ШЯЬ*ГЗЯЖХЫЩЫГЯЙЫЖЫХЯЩИ*ЯЦЫГ*

Поступим так. Подсчитаем частоты всех символов текста. Самый частый заменим всюду на пробел (в приведенном тексте самый частый символ — *я*). Второй по частоте заменим на букву *о*, третий — на *е*, четвертый — на *а*, пятый — на *и*, а шестой — на *н*. Остальные символы заменим подчеркиваниями.

В приведенном тексте несколько сот символов и все эти подсчеты и замены вполне посильно провести вручную. Но лучше написать небольшую программу, обрабатывающую таким образом любой текст.

«Поиграть» с этой программой можно, запустив гипертекст к главе 3.

Вот результат работы этой программы. Для удобства он напечатан рядом с зашифрованным текстом:

```
ЦЗСЯЩ*Щ*ЯЙБЦЕРЯТЫЙЧГЕРЯЭХБЖЬЫ    _о_ ___ _а__ _е_н_ __а_и_
ЮЗНЩБЯГЫЯЖЛФЧЮФЯОБГЫЦЗН          _о_а не _____ _ане_о_
ЗГЯФЖБВБЧИЯЙЫА*ЯОБЙЧБЖЬЫ         он __а_а__ _е__ _а__а_и_
ЬЯЮФТЛЫЯЖЕЩФЦБЧИЯГЫЯЦЗН          и ___е ____а__ не _о_
ЫНЗЯЭХЬЦЫХЯЩХФНЬЦЯГБФЮБ          е_о __и_е_ ___и_ на__а
ГЗЯЯАЗВЫЯЦЗСЯЯЮБЮБ*ЯЙЮФЮБ        но _о_е _о_ _а_а_ ___а
ЙЯАЗЮИГЕЦЯЙЬЫЧИЯЬЯЩЫГИЯЬЯГЗТИ    _ _о_н__ _и_е__ и _ен_ и но__
ГЫЯЗЧРЗЩ*ЯГЬЯЛБНФЯЭХЗТИ          не о__о__ ни _а__ __о__
ЮБЮЗЫЯГЪЮЮЗЫЯЮЗЖБХЙЧЖЗ           _а_ое ни__ое _о_а____о
ЭЗЮФВЬЖЗНЗЯОБАБЖЮ*ЧИ             _о___и_о_о _а_а_____
ЫЦФЯЭЗЩФЛЮЬЯЭЗЭХБЖЮ*ЧИ           е__ _о____и _о__а____
ЭЫТБЮИГЗЯЭЗЩГЗЙЬЧИЯЮЫЮБХЙЧЖЗ     _е_а_но _о_но_и__ _е_а____о
ЖОЩЕРБЧИЯЬЯЩФЦБЧИЯЭХЗЯЙЫА*       _____а__ и ___а__ __о _е__
ЯЮЗНЩБЯВЫЯТЫХЧЯЖЗОИЦЫЧЯЧЫА*     _о__а _е _е__ _о___е_ _е__
ЧБЮЯЩФЦБЮЯЦЗЮЗЩЗСЯЭЗЖЫЙБ         _а_ ___а_ _о_о_о_ _о_е_а
ЫЫЧ*ЯЖЯЭЕЮЬЯГБЯЭЗТЧЗЖЕР          _е__ _ __и на _о__о___
ЖЙЫЖЕЛГЫСЯЖЗЮЫКЯОЫЖЫЙБ           __е___не_ _о_е_ _е_е_а
ГБЙЮЫЩГЬЮЯЖЙЫРЯЙЖЗЬРЯХЗЩГЕР      на__е_ни_ __е_ __ои_ _о_н__
ЩХФОИ*ЯЮКЩЦЬЮЕЯЬЯХФЙЮБГБ        _____ ____и_ и ____ана
ЙЯНЫХЗЫЦЯЦЗЫНЗЯХЗЦБГБ           _ _е_ое_ _ое_о _о_ана
АЫОЯЭХЫЩЬЙЮЗЖЬСЯЙЫСЯВЫЯТБЙ      _е_ __е_и_о_и_ _е_ _е _а_
ЭЗОЖЗЮИЧЫЯЭЗОГБЮЗЦЬЧИЯЖБЙ       _о__о___е _о_на_о_и__ _а_
ЗГЫНЬГЯЯЩЗАХЕСЯЦЗСЯЭХЬ*ЧЫЮИ     оне_ин _о____ _о_ __и__е_
ХЗЩЬЙЙ*ЯГБЯАХЫНБРЯГЫЖЕ          _о_и___ на __е_а_ не__
НЩЫЯЯЦЗВЫЧЯЯЕЧИЯЯЯХЗЩЫЬЙИЯЯЖЕ  __е _о_е_ ____ _о_и_и__ __
ЬЫЬЯАЫЬЙЧБЮЬЯЯЦЗСЯТЬЧБЧЫЮИ      и_и __и__а_и _о_ _и_а_е__
ЧБЦЯГЫЮЗНЩБЯЮЙЮ*ЫЯЬ*           _а_ не_о__а ____ и _
ГЗЯЖХЫЩЫГЯЙЫЖЫХЯЩЮ*ЯЦЫГ*        но __е_ен _е_е_ ___ _ен_
```

Похоже, что пробел и первые 5 самых частых букв угаданы верно. Среди расшифрованных слов видны несколько часто встречающихся слов, составленных из тех 5 букв, которые мы пытались угадать («и», «ни», «не» «но», «он»). Теперь найдем слова, в которых не расшифрована всего лишь одна буква:

```
ЫНЗ  е_о
ЬЫЬ  и_и
```

Первое слово, скорее всего, расшифровывается как «его». Значит, Н расшифровывается как г. Второе слово, скорее всего, расшифровывается как «или». Значит, Ы расшифровывается как *л*. Подставим две угаданные буквы в текст:

```
ЦЗСЯЩ*Щ*ЯЙБЦЕРЯТЫЙЧГЕРЯЭХБЖЬЫ    _о_ ___ _а__ _е_н_ __а_ил
ЮЗНЩБЯГЫЯЖЛФЧЮФЯОБГЫЦЗН          _ог_а не _____ _ане_ог
ЗГЯФЖБВБЧИЯЙЫА*ЯОБЙЧБЖЬЫ         он __а_а__ _е__ _а__а_ил
ЬЯЮФТЛЫЯЖЕЩФЦБЧИЯГЫЯЦЗН          и л___е ____а__ не _ог
ЫНЗЯЭХЫЦЫХЯЩХФНЬЦЯГБФЮБ          его __и_е_ ___ги_ на__а
ГЗЯЯАЗВЫЯЦЗСЯЯЮБЮБ*ЯЙЮФЮБ        но _о_е _о_ _а_а_ ___а
ЙЯАЗЮИГЕЦЯЙЬЫЧИЯЬЯЩЫГИЯЬЯГЗТИ    _ _ол_н__ _и_е__ и _ен_ и но__
ГЫЯЗЧРЗЩ*ЯГЬЯЛБНФЯЭХЗТИ          не о__о__ ни _аг_ __о__
ЮБЮЗЫЯГЪЮЮЗЫЯЮЗЖБХЙЧЖЗ           _а_ое ни__ое _о_а____о
ЭЗЮФВЬЖЗНЗЯОБАБЖЮ*ЧИ             _ол_и_ого _а_а_л___
ЫЦФЯЭЗЩФЛЮЬЯЭЗЭХБЖЮ*ЧИ           е__ _о____и _о__а_л___
ЭЫТБЮИГЗЯЭЗЩГЗЙЬЧИЯЮЫЮБХЙЧЖЗ     _е_ал_но _о_но_и__ ле_а____о
ЖОЩЕРБЧИЯЬЯЩФЦБЧИЯЭХЗЯЙЫА*       _____а__ и ___а__ __о _е__
ЯЮЗНЩБЯВЫЯТЫХЧЯЖЗОИЦЫЧЯЧЫА*     _ог_а _е _е__ _о___е_ _е__
ЧБЮЯЩФЦБЮЯЦЗЮЗЩЗСЯЭЗЖЫЙБ         _а_ ___ал _оло_о_ _о_е_а
ЫЫЧ*ЯЖЯЭЕЮЬЯГБЯЭЗТЧЗЖЕР          ле__ _ __ли на _о__о___
ЖЙЫЖЕЛГЫСЯЖЗЮЫКЯОЫЖЫЙБ           __е___не_ _оле_ _е_е_а
ГБЙЮЫЩГЬЮЯЖЙЫРЯЙЖЗЬРЯХЗЩГЕР      на_ле_ни_ __е_ __ои_ _о_н__
ЩХФОИ*ЯЮКЩЦЬЮЕЯЬЯХФЙЮБГБ        _____ л__ил_ и ___лана
ЙЯНЫХЗЫЦЯЦЗЫНЗЯХЗЦБГБ           _ ге_ое_ _оего _о_ана
АЫОЯЭХЫЩЬЙЮЗЖЬСЯЙЫСЯВЫЯТБЙ      _е_ __е_и_ло_и_ _е_ _е _а_
ЭЗОЖЗЮИЧЫЯЭЗОГБЮЗЦЬЧИЯЖБЙ       _о__ол__е _о_на_о_и__ _а_
ЗГЫНЬГЯЯЩЗАХЕСЯЦЗСЯЭХЬ*ЧЫЮИ     онегин _о____ _о_ __и__ел_
ХЗЩЬЙЙ*ЯГБЯАХЫНБРЯГЫЖЕ          _о_ил__ на __ега_ не__
НЩЫЯЯЦЗВЫЧЯЯЕЧИЯЯЯХЗЩЫЬЙИЯЯЖЕ  г_е _о_е_ ____ _о_или__ __
ЬЫЬЯАЫЬЙЧБЮЬЯЯЦЗСЯТЬЧБЧЫЮИ      или _ли__али _о_ _и_а_ел_
ЧБЦЯГЫЮЗНЩБЯЮЙЮ*ЫЯЬ*           _а_ не_о_а г_л_л и _
ГЗЯЖХЫЩЫГЯЙЫЖЫХЯЩЮ*ЯЦЫГ*        но __е_ен _е_е_ _л_ _ен_
```

Теперь снова найдем слова, в которых не расшифрована только одна буква:

```
НЩЫ    г_е
ЦЗЫНЗ  _оего
```

Первое из них расшифровывается как «где», а второе, скорее всего, как «моего». Впрочем, одно из уже полностью расшифрованных слов — онегин — подсказывает, что пора взять с полки «Евгения Онегина» и закончить расшифровку.

11.10. Шифрование с открытым ключом

Широкое распространение в современных коммерческих системах получил метод шифрования ***с открытым ключом***. Суть этого метода заключается в следующем.

Ключ состоит из пары очень больших целых чисел. Алгоритм шифрования зависит от ключа как от параметра. Ниже результат шифрования текста T с ключом K будет записываться как $T \star K$.

Специальный алгоритм дешифровки при шифровании с открытым ключом не нужен. Для расшифровки нужно выполнить алгоритм шифрования с другим ключом, который называется ***обратным*** к исходному. Если ключ K' обратен к K, то для любого

текста T выполняется $T \star K \star K' = T$. Если $T1 = T \star K$, то $T1 \star K'$ даст исходное сообщение. Для дальнейшего очень существенно свойство взаимной обратности ключей: если K' обратен к K, то K обратен к K'.

Как применить все это на практике? Есть инструкция, по которой каждый желающий может самостоятельно, без посторонней помощи, придумать (а точнее, изготовить их за пару часов на своем персональном компьютере) два взаимно обратных ключа $Л$ и $П$. Ключ $Л$ держится в секрете (он называется закрытым или *личным*), а ключ $П$ сообщается всем желающим (он называется открытым, или *публичным*).

Предположим, что Старбанк уже давно пользуется системой шифрования с открытым ключом. Его публичный ключ всем известен, он сообщается по телефону всем желающим. Пусть вновь открытый Новбанк хочет обмениваться информацией со Старбанком. Директор Новбанка раздобывает инструкцию, запирается в своем кабинете и изготавливает два шифровальных ключа: $Л_н$ и $П_н$. Ключ $П_н$ немедленно публикуется в финансовой газете. Дискета с ключом $Л_н$ прячется в сейф директора. Эти два ключа взаимно обратны. Если текст T зашифровать ключом $Л_н$ и опубликовать в газете, то любой желающий его расшифрует, зашифровав его ключом $П_н$, опубликованным ранее в финансовой газете. Это записывается так:

T — исходный текст,

$T \star Л_н$ — зашифрованный текст и

$T \star Л_н \star П_н$ — дважды зашифрованный текст.

Поскольку ключи $П_н$ и $Л_н$ взаимно обратны, дважды зашифрованный текст совпадает с исходным.

Тот факт, что текст $T \star Л_н$ после шифрования ключом $П_н$ оказывается осмысленным текстом, доказывает, что этот текст действительно изготовлен в Новбанке (или, что ключ $Л_н$ стал кому-то известен).

Предположим теперь, что директор Старбанка хочет передать директору Новбанка секретную информацию. Как ему поступить? Очень просто. Зашифровать ее ключом $П_н$ и доставить в Новбанк, совершенно не заботясь о секретности. Расшифровать ее сможет лишь тот, кто знает ключ $Л_н$, т.е. только директор Новбанка.

Теперь представим себе, что секретное сообщение из Старбанка в Новбанк гласит: «Перечислите с нашего счета №12345678 в Вашем банке 1 000 000 (один миллион) долларов на счет №987654321 в Зарубежбанке. Директор Старбанка Комиссаров». Вряд ли директор Новбанка начнет немедленно выполнять подобный приказ. Шифрование обеспечивает секретность этого приказа, но не его подлинность. Подобный приказ может послать кто угодно, например владелец счета в Зарубежбанке.

Чтобы гарантировать не только секретность, но и подлинность сообщения, директор Старбанка должен поступить так. Исходный текст T он сначала шифрует своим личным секретным ключом $Л_с$, это можно обозначить как $T \star Л_с$. То, что получилось, он повторно шифрует ключом $П_н$ — этот ключ он взял в газете (или позвонил в Новбанк и узнал у секретарши).

Таким образом, директор Новбанка Иванов получит сообщение вида

$$T \star Л_с \star П_н.$$

Сначала он шифрует полученное своим личным ключом $Л_н$, взяв его из сейфа. То, что получилось, Иванов еще раз шифрует всем известным публичным ключом Старбанка $П_с$. В результате 4 шифрования подряд дадут

$$T \star Л_с \star П_н \star Л_н \star П_с,$$

но этот текст совпадает с T, так как шифрование ключом $Л_н$ уничтожает действие шифрования ключом $П_н$, а затем шифрование ключом $П_с$ уничтожает действие шифрования ключом $Л_с$.

Теперь, прочтя расшифрованное сообщение, директор Новбанка может быть уверен, что оно поступило из Старбанка. Никто другой не смог бы послать сообщение, выдерживающее расшифровку ключом $П_с$.

Алгоритмы шифрования в коммерческих системах построены так, что их практически невозможно раскрыть, не зная ключа. Не вдаваясь в подробности, скажем, что для взламывания шифра нужно разложить на сомножители очень большое целое число (около 400 десятичных цифр). Если такое число имеет всего два примерно равных простых сомножителя, то задача их нахождения не по зубам даже самым быстрым современным компьютерам.

§ 12. ПАМЯТЬ НАШЕГО КОМПЬЮТЕРА

Перефразируя известное выражение, можно сказать, что

> «лучше купить дешевый компьютер с большой и быстрой памятью, чем дорогой компьютер с маленькой и медленной памятью»

Но чудес не бывает, и большая память стоит дороже, чем маленькая, а быстрая — дороже, чем медленная. Поэтому и производителям компьютеров, и покупателям приходится искать компромисс

между ценой, объемом и скоростью доступа к памяти компьютера, стараясь скомбинировать разные виды памяти так, чтобы компьютер работал оптимально.

Да и на уже укомплектованном компьютере организовать хранение информации можно по-разному. Поэтому каждый владелец должен иметь хотя бы общее представление о том, как устроен компьютер и его память, а точнее *памяти* (во множественном числе), поскольку есть несколько принципиально разных видов памяти.

12.1. Принципы устройства памяти компьютера

Начнем с краткого обзора основных компонентов ЭВМ. При всем разнообразии этих компонентов их можно разделить на 4 основные категории: ***процессор***, ***оперативная память***, ***внешняя память*** и прочие ***внешние устройства***. Последние позволяют компьютеру обмениваться информацией с человеком и другими компьютерами, управлять технологическими процессами и т.п.

Компоненты компьютера обмениваются информацией между собой, используя электрические сигналы, передаваемые по проводам.

Минимальный элемент памяти (бит) способен хранить минимально возможный объем информации — одну двоичную цифру. Здесь не обсуждается, как этот минимальный элемент устроен с точки зрения радиотехники. Быстродействие ЭВМ хотя и опирается на быстродействие этих элементарных радиотехнических элементов, но во многом зависит от общей организации памяти ЭВМ.

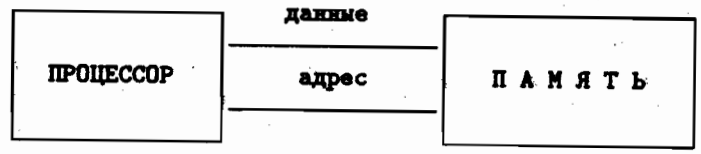
Адреса байтов

Для того чтобы отдельные компоненты ЭВМ работали слаженно, они должны подчиняться некоторым общим правилам.

Первое правило относится к памяти. С точки зрения современного процессора бит настолько маленькая единица, что неразумно давать ей отдельное имя. Поэтому биты в памяти любого вида объединяются в ***байты*** — восьмерки битов. Каждый байт является, так сказать, «самостоятельным лицом», он имеет свое личное ***имя***. Зная имя байта N, можно совершать две основные операции: прочитать *нечто* из байта с именем N и записать *нечто* в байт с именем N. Принято для именования байтов использовать неотрицательные целые числа и говорить о номерах или ***адресах*** байтов.

Взаимодействие процессора и памяти

С памятью могут работать разные компоненты ЭВМ, но чаще всего с нею взаимодействует процессор. Как же процессор может прочитать *нечто* из байта с *адресом* N или записать *нечто* в байт с *адресом* N? Во-первых, от процессора к памяти должен поступить адрес. Во-вторых, сам байт информации должен быть передан от процессора к памяти (при записи) или от памяти к процессору (при чтении). Все это должно передаваться по проводам, и провода эти разбиты на две части, два пучка, называемые ***шинами***:



Одна часть проводов называется ***шина адреса***, другая — ***шина данных***. Адрес байта передается по шине адреса, а сам байт (в том или ином направлении) — по шине данных. Как правило, эти процессы происходят одновременно.

Число проводов в шине данных называется ***разрядностью*** шины. Обычно разрядность равна 8, 16, 32 или 64 (вспомним, что 1 байт = 8 бит). Если шина 8-разрядная, то за один раз можно передать 1 байт. Если шина данных 16-разрядная, то за один раз можно передать два байта. Однако по шине адреса в этот момент передан только один адрес N. Как быть? Да просто работать с двумя адресами N и $N+1$. Аналогично, если шина 32-разрядная, то нужно работать с четырьмя последовательными адресами N, $N+1$, $N+2$ и $N+3$.

Таким образом, точнее было бы сказать, что шина адреса задает адрес байта памяти или адреса нескольких последовательных байтов. Шина адреса также имеет некоторую разрядность. Чем больше эта разрядность, тем больше адресов можно указать, пользуясь шиной. Если шина, например, 20-разрядная, то максимально возможное количество адресов равно $2^{20} = 1\,048\,576$. Таким образом, максимально возможный объем памяти, обслуживаемой шиной, равен 1 мегабайту.

Можно ли при 20-разрядной шине адреса работать с памятью размером больше 1 Мб? Можно, если разбить адрес на две части и передавать их по шине в два приема. Такой метод называется ***мультиплексированием*** и позволяет иметь большее количество адресов за счет увеличения времени на передачу адреса.

Что еще надо для работы с памятью?

Достаточно ли описанных выше проводов для связи процессора с памятью? Очевидно, нет. Например, как память может понять, когда передаваемый по шине байт надо записать, а когда прочесть?

Можно протянуть еще один провод от процессора к памяти. Пусть передача 0 по этому проводу означает, что байт надо прочесть из памяти, а передача 1 — что надо записать в память.

Можно представлять себе, что низкое напряжение (около 0 вольт) означает 0, а высокое (большее некоторого порогового значения V) означает 1. Обычно единице соответствует напряжение около 5 вольт. Важно отметить, что по проводу нельзя передать никакого «промежуточного» значения — любой сигнал будет восприниматься либо как 0 (если напряжение меньше V), либо как 1 (в противном случае).

Однако при этом остается другая проблема. В любой момент времени на шинах находится некоторый адрес и какие-то данные, так как каждый бит равен либо 0, либо 1. Как же память может определить, что надо выполнять очередное требование процессора?

Поэтому обычно адресная шина и шина данных дополняются еще двумя проводами: один называется *запрос чтения* (передача по нему бита 1 означает указание памяти прочесть байт), а другой — *запрос записи* (здесь передача бита 1 означает указание записать байт). Передача единицы сразу по обоим проводам запрещена.

Реально для работы потребуется еще один провод — память должна уметь сообщать, что она выполнила запрос процессора и готова к принятию следующего запроса.

От чего зависит производительность компьютера

Как видно, проводов для связи процессора и памяти требуется довольно много (а ведь перечислены только самые главные). В современных компьютерах для управления сигналами на этих проводах используются специальные электронные схемы. Вместе их можно считать отдельной частью ЭВМ (ее называют ***магистралью*** или ***шиной***, без указания слов *адрес* или *память*). С учетом этого общая схема взаимодействия процессора и памяти будет выглядеть так, как показано на рис. 7.

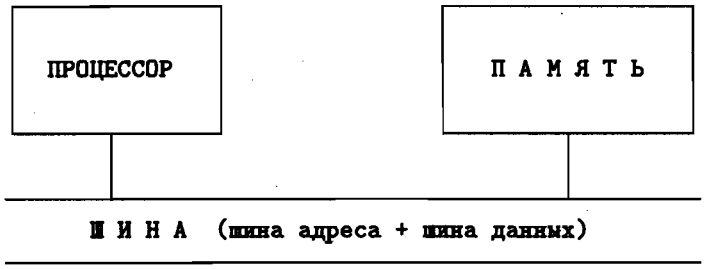


Рис. 7. Схема взаимодействия процессора и памяти

Говорят, что процессор и память ***обслуживаются*** шиной. Шина может обслуживать и другие компоненты ЭВМ. Основная задача шины (кроме собственно передачи данных) — согласование скоростей работы обслуживаемых ею устройств. Для этого шина вырабатывает собственные сигналы. Благодаря этим сигналам процессор может понять: готова ли шина передать или принять информацию? Важно, чтобы ***производительности*** шины и всех подсоединенных к ней устройств были согласованы. Неразумно иметь быстрый процессор и медленную память или быстрые процессор и память, но медленную шину.

12.2. Внешние запоминающие устройства

Оперативная память компьютера должна работать быстро и потому достаточно дорога. Она не может вместить громадные объемы информации, перерабатываемой компьютером. Кроме того, ее содержимое теряется при выключении компьютера. По этим причинам компьютеру необходима ***внешняя*** память. Внешняя память также может быть устроена по-разному.

В одну группу попадают так называемые устройства с ***произвольным доступом***, в другую — устройства с ***последовательным доступом***. Типичный пример устройства с последовательным доступом — магнитная лента. Данные можно читать или записывать только последовательно. А если порядок нарушается, то нужно долго ждать, пока лента будет перемотана на нужное место. Устройства с произвольным доступом позволяют получить доступ к произвольной порции данных примерно за одно и то же ***время доступа***.

Вот основные устройства с произвольным доступом:

Дискеты — маленькие магнитные диски, упакованные в пластиковый «конверт», гибкий у $5\frac{1}{4}$-дюймовых (130 мм) дискет и жесткий — у $3\frac{1}{2}$-дюймовых (90 мм). Данные хранятся на концентрических магнитных дорожках, расположенных на поверхности диска, как правило, с двух сторон. Устройство, которое читает и пишет на дискеты, называется ***дисководом для гибких магнитных дисков***. Внутри дисковода находятся две магнитные головки. Они могут перемещаться радиально от границы диска к его центру шаговым двигателем. Сама дискета вращается с небольшой скоростью. В итоге чтение или запись можно произвести в любом месте дискеты, подождав не более продолжительности одного оборота.

Максимальная емкость $5\frac{1}{4}$-дюймовой дискеты — 1.2 мегабайт. На $3\frac{1}{2}$-дюймовые дискеты, несмотря на их меньшие размеры, помещается больше информации — 1.44 Мб. Кроме того, такие дискеты лучше защищены от механических повреждений и легко помещаются в карман пиджака (правда, и стоят они дороже).

Скорость передачи данных при работе с дискетой невелика — примерно несколько килобайт в секунду. Среднее время доступа — 250 мс. В настоящее время существуют дискеты и дисководы, разработанные фирмой Toshiba, у которых емкость дискеты увеличена в два раза и доведена до 2.88 Мб.

Винчестеры — жесткие несъемные магнитные диски. Уже давно на «больших» ЭВМ применялись съемные магнитные диски. Переход к несъемному магнитному носителю (диск конструктивно объединен в едином блоке с дисководом) дал возможность существенно повысить скорость вращения диска и плотность записи на диск. Емкость современных винчестеров варьируется от сотен мегабайт до нескольких гигабайт. Скорость передачи данных достигает 1.5 мегабайт в секунду. А среднее время доступа около 10 мс. На современных компьютерах это основной вид внешней памяти.

> Легенда гласит, что первое устройство такого типа состояло из двух дисков по 30 мегабайт в каждом. Суммарная емкость такой памяти обозначалась 30/30. Это обозначение совпало с обозначением модели популярного винчестерского охотничьего ружья, откуда и возникло название «винчестер».

Оптические диски — получили распространение вслед за бытовыми оптическими аудиокомпакт-дисками (лазерными дисками

или CD). Устройство лазерного проигрывателя механически аналогично устройству дисковода для гибких дисков, однако при чтении используется лазер, а дорожка с информацией — единственная спиральная (как на виниловой пластинке). Лазерные диски имеют те же форматы, что и аудио-диски. Более того, компьютерные устройства для чтения с компакт-дисков позволяют воспроизводить на компьютере через звуковую плату стандартный аудио компакт-диск. К недостаткам можно отнести невозможность перезаписывать компакт-диски и медленную скорость устройства. Скорость передачи данных составляет в среднем 150—300 килобайт в секунду (на устройствах фирмы NEC до 600—900 килобайт в секунду), а среднее время доступа — 200—300 мс (как у дисководов для гибких дисков). К достоинствам относятся дешевизна самого диска и большой объем записанной информации: 500—600 мегабайт.

Последние новинки — оптические записываемые диски, магнито-оптические и магнитные диски большой емкости, дисковод для дискет на $3\frac{1}{2}$дюйма, который может писать на специальные дискеты емкостью до 150 мегабайт, магнитные съемные карты и т.д. и т.п. Прогресс не остановишь!

Устройство с последовательным доступом практически только одно:

Магнитные ленты — достаточно медленные устройства последовательного доступа, хотя и большой емкости. Современные устройства для работы с магнитными лентами — ***стримеры*** — имеют увеличенную скорость записи, примерно 4—5 мегабайт в минуту. Дешевизна магнитных лент позволяет использовать стримеры для организации долгосрочного хранения информации. Емкость одной кассеты стримера меняется в тех же пределах, что и емкость винчестера. В последнее время появились устройства, позволяющие записывать цифровую информацию на обычные видеокассеты (на одну кассету удается записать около 2 Гб информации).

12.3. Что такое 100 мегабайт?

В 100 мегабайтах памяти можно хранить:

- 50 000 страниц текста или около 150 романов;
- свыше 150 цветных слайдов высочайшего качества;
- аудиозапись 1,5-часовой речи;

- 10-минутный музыкальный фрагмент качества CD-стерео;
- 15-секундный фильм высокого качества записи;
- 1000-летний протокол операций с банковским счетом крупной фирмы.

Стоимость устройства, способного хранить 100 Мб информации, в настоящее время ниже ста долларов. Стоимость аренды пространства для хранения 100 Мб в компьютерных сетях составляет около ста долларов в год. Можно ожидать, что через несколько лет это же будет верно после замены Мб на Гб.

12.4. Кэш нам поможет

Быстрее всего процессор может записать или прочесть данные из своей сверхбыстрой памяти (регистров). Далее, в порядке убывания скорости доступа, идут основная память и внешняя память. Если бы вся память была сверхбыстрой, то это сильно повысило бы производительность компьютера, но быстрая память дорога. И на помощь приходит ***кэш-память***. Это быстрая память небольшого объема, связанная с основной памятью следующим способом.

Когда процессор записывает в основную память байт, этот байт (и его адрес в основной памяти) заносится одновременно и в кэш-память. Всякий раз, когда процессор намерен прочитать некоторый байт, сначала проводится анализ: есть ли байт с этим адресом в быстрой кэш-памяти? Если он там есть, то происходит чтение из кэш-памяти. Если нет, то байт копируется из основной памяти в кэш-память и одновременно передается процессору.

> Что же произойдет в тот момент, когда вся кэш-память будет заполнена? Существуют разные методы очистки кэш-памяти. Например, из нее можно удалять данные, записанные первыми, или те, к которым было меньше всего обращений.

В результате в каждый момент времени наиболее часто используемая часть основной памяти уже оказывается скопированной в кэш-память и становится быстродоступной. Применение кэш-памяти во многих случаях повышает производительность компьютера в несколько раз.

Принцип кэш-памяти может быть с успехом использован во всех случаях, когда есть относительно небольшая быстрая память и относительно большая, но медленная. Например, тот же принцип можно применить к винчестеру и оперативной памяти.

12.5. Структура хранения информации в памяти

Основная память компьютера с логической точки зрения организована весьма примитивно. Чтобы получить байт из памяти, надо знать его адрес. Все байты в некотором смысле одинаковы, равноправны. Хотя в основной (оперативной) памяти компьютера можно хранить любую информацию (например, закодированные тексты, изображения, числа и т.п), основной единицей информации является байт произвольных данных.

Точно так же можно хранить произвольную информацию во внешней памяти, например на винчестере. Однако человеку неудобно работать с информацией, обращаясь непосредственно к байтам по их адресам. Такой способ годится для процессора, но не для человека. Первым шагом к более «человеческой» организации информации на внешних носителях были ***файлы*** и ***файловые системы***.

Файл — это именованная единица информации, имеющая внутреннюю структуру. Файловая система позволяет работать с файлами безотносительно к их содержимому, примерно так же, как библиотекарь работает с книгами, не раскрывая их. Вот несколько аналогий между операциями файловой системы и операциями над книгами в библиотеке:

Книжная полка	*Файловая система*
Снять книгу с полки, изменить страницы книги (добавить новые, удалить старые, переписать или дописать старые) и поставить на место	Модифицировать файл при помощи программы (например, текстового редактора)
Сделать копию книги при помощи ксерокса и поставить ее на полку	Скопировать файл
Снять книгу с полки и выбросить	Удалить файл
Снять книгу с полки и поставить в другое место на полку	Переместить файл в рамках носителя (винчестера)
Снять книгу с полки и поставить на другую полку	Переместить файл на другой носитель (винчестер)

Файловую структуру организации данных имеют не только винчестеры, но и оптические компакт-диски, дискеты и пр. Одна-

ко надо отметить, что различные устройства даже на одном компьютере могут использовать различные файловые системы.

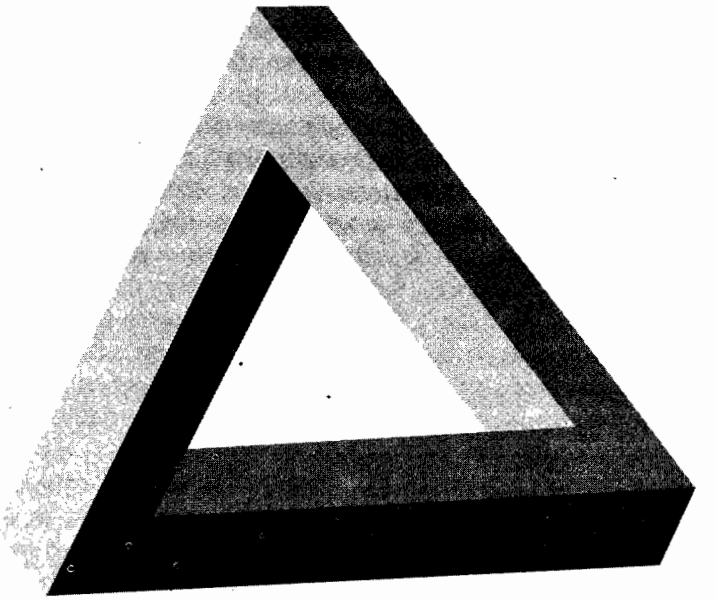
Для простейших работ на компьютере (например, для подготовки текстов и организации личного архива) понятий *файл* и *файловая система* достаточно. Для более сложных работ, связанных с большими объемами информации или с часто меняющимися данными, нужны более сложные структуры, примерно так относящиеся к файлам, как файлы относятся к байтам. Примером могут служить ***базы данных*** и ***информационные системы***. Они опираются на файловую систему, но дают возможность пользователю оперировать не с понятием *файл*, а с понятиями более высокого уровня.

ИНФОРМАЦИОННАЯ КУЛЬТУРА

ИНФОРМАЦИОННЫЕ МОДЕЛИ

КЛАСС

10



Глава 1 ПЕРЕДАЧА ИНФОРМАЦИИ

§ 1. ПЕРЕДАЧА ИНФОРМАЦИИ МЕЖДУ ЛЮДЬМИ

Всю информацию об окружающем мире человек получает с помощью пяти чувств — вкуса, обоняния, осязания, слуха и зрения. Значение каждого из них связано с количеством информации, которое человек получает с их помощью. Потеря обоняния (например, при простуде) не слишком страшна в отличие от потери слуха или зрения. Осязание более важно; для слепых, к примеру, оно является единственным способом чтения книг. Большую часть информации человек получает с помощью слуха и зрения, причем зрение неизмеримо важнее слуха (лучше один раз увидеть, чем сто раз услышать).

1.1. Передача информации и частоты колебаний

Звуки, которые мы слышим, представляют собой колебания воздуха. Частоты, воспринимаемые слухом, лежат в диапазоне от 20 до 20 000 герц, но речь почти полностью укладывается в полосу частот от 100 до 3000 герц. Именно такой диапазон частот передается по телефону. Видимый свет также представляет собой колебания, только электромагнитные. Частоты их неизмеримо выше и составляют миллиарды герц.

Чем больше частота колебаний, с помощью которых передается информация, тем большее количество информации можно передать в единицу времени. Это утверждение имеет точное математическое обоснование, а инженерам-практикам оно было известно уже с появлением радио. Подтверждения можно видеть и в природе: например, дельфины, летучие мыши, а также некоторые виды ночных птиц большую часть информации получают с помощью слуха, частично заменяющего им зрение. При этом все они используют для эхолокации и для общения друг с другом ультразвук, т.е. звук очень высокой частоты.

Радио и телевидение в настоящее время используют для передачи информации радиоволны, т.е. электромагнитные колебания. Чем меньше длина волны (чем больше несущая частота), тем

большего качества можно достичь при передаче. Обычные радиопрограммы передаются на длинных, средних и коротких волнах, но высококачественная стереофоническая музыка может быть передана только на ультракоротких волнах (длины волн порядка нескольких метров). Передача изображения — телевидение — требует еще более высоких частот, длины волн уже измеряются метрами и дециметрами.

В нашей стране длина волны для первого канала телевидения равна 6.03 метра, частота 49.75 МГц, для 12-го канала — 1.35 метра и 223.25 МГц, каналы с 13-го по 99-й используют дециметровый диапазон. В современных компьютерных сетях, а также в проектах цифрового телевидения будущего в качестве несущей используется свет видимого диапазона, генерируемый микролазером. Средой передачи служит оптоволоконный кабель.

Упражнение:

1. Для передачи телефонного разговора требуется диапазон частот от 0 до 3000 Гц, т.е. так называемая ширина канала должна составлять 3 килогерца. Для надежного разделения каналов ширина полосы частот между ними должна быть не менее 2 кГц. Сколько телефонных разговоров можно одновременно передавать в полосе частот между первым и вторым каналами телевидения от 49 до 59 МГц?

1.2. Передача информации на большое расстояние

Непосредственное общение людей друг с другом возможно лишь на небольшом расстоянии, поэтому на любом уровне развития общества существовали многообразные способы передачи данных на расстоянии. Это костры, тамтамы, морская флажковая азбука, семафоры на железных дорогах и т.п. Отметим характерные особенности любой передачи данных на большое расстояние:

- на большое расстояние данные передаются по цепочке, через ряд промежуточных участников передачи (промежуточных станций, ретрансляторов и т.п.);
- всякая такая передача должна быть подчинена четким, заранее установленным правилам. Должны быть заранее определены виды сигналов, смысл каждого из них, действия, которые надо совершать при успешном приеме сообщения или при необходимости повторной передачи (обычно устанавливается какой-либо способ подтверждения приема или запроса на повторную передачу). В мире компьютеров набор таких правил называется *протоколом передачи*;
- передачи бывают двусторонними (разговор на большом расстоянии) либо односторонними. В последнем случае передача может быть *широковещательной* — адресованной сразу большому числу участников.

В современном мире самые распространенные способы передачи информации на расстоянии — это телеграф, телефон, радио и телевидение. Приставка *теле-* означает «дальний» или «удаленный» (*телеграф* = письмо на расстоянии, *телефон* = звук на расстоянии, *телевидение* = изображение на расстоянии). И хотя эти способы передачи информации появились не так давно — в XIX—XX столетиях, — представить без них современную цивилизацию невозможно.

Упражнение:

2. Привести примеры передачи информации на расстоянии, отличные от перечисленных в тексте.

1.3. Почта

Почта — один из самых древних способов передачи информации. По-видимому, письма существуют столько же времени, сколько и письменность, а письменность в той или иной форме есть у любых племен и народов.

Первоначально письма доставлялись специальными людьми — гонцами либо передавались, например, с попутным кораблем. При достаточном уровне развития общества формируется государственная система доставки писем — почта. Почта существовала уже в Древнем Риме. В Европе почта впервые появилась во Франции при Людовике XI. Услугами почты могут пользоваться более или менее широкие общественные группы, начиная от аристократии в Древнем Риме и кончая всеми людьми в современном обществе.

Отметим характерные особенности писем во все времена у любых народов. Письмо обязательно содержит адрес человека, которому оно предназначено, и обратный адрес. Письмо имеет конверт, на котором, кроме адресов, может записываться дополнительная служебная информация (номер почтового отделения, даты отправки и приема, категория письма и т.п.). Если даже конверта нет, то все равно адреса и другая служебная информация тем или иным способом отделены от содержания письма.

Важное отличие писем от других способов передачи информации на расстоянии (телефона, телевидения) состоит в том, что

получатель письма не обязательно должен присутствовать в тот момент, когда письмо доставляется по назначению. Письмо обычно сохраняется в почтовом ящике до тех пор, пока получатель не заберет его оттуда.

Таким образом, письмо — это передача данных не только на расстоянии, но еще и во времени. Можно, например, оставить записку на рабочем столе человека, зная, что он появится здесь на следующий день и прочтет ее.

Упражнение:

3. Когда на Руси впервые появилась почта и как она выглядела?

§ 2. ПЕРЕДАЧА ИНФОРМАЦИИ МЕЖДУ КОМПЬЮТЕРАМИ

История развития компьютерных сетей насчитывает уже несколько десятилетий. Например, проект DARPA, который привел к созданию самой крупной из существующих сейчас общемировых сетей — сети Internet, был начат Министерством обороны США в 1969 г. Но широкое распространение компьютерные сети получили в 80-х годах, с появлением персональных компьютеров и лавинообразным ростом числа самых различных компьютеров.

Хотя компьютерные сети появились всего двадцать–тридцать лет назад, мы являемся свидетелями того, что они вторгаются в нашу жизнь, постепенно становясь ее неотъемлемой принадлежностью.

Уже сейчас есть сферы человеческой деятельности, которые не могут существовать без компьютерных сетей. Работа над большими научными проектами, работа банков, крупных библиотек и вообще работа с большими объемами информации невозможна без компьютерных сетей; сети используются при управлении крупными автоматизированными производствами, газопроводами, электростанциями — всюду, где требуется слаженная работа многообразного сложного оборудования, расположенного на больших расстояниях.

Сети — идеальное средство быстрого обмена научной информацией. К примеру, в настоящее время в мире почти не печатаются научные журналы по физике; вместо них издаются электронные журналы, распространяемые по всемирной сети.

2.1. Среда передачи

Для передачи данных компьютеры используют самые разнообразные физические каналы, которые обычно называются ***средой передачи***. Традиционная среда передачи — это электрический кабель, состоящий из одного или нескольких металлических проводников. Компьютеры могут передавать информацию с помощью радио, непосредственно друг другу или через ретрансляторы, спутники связи и т.п. На небольшом расстоянии, в пределах одной комнаты, компьютеры могут связываться между собой с помощью инфракрасных лучей. Наконец, наиболее современная и перспективная среда передачи — это оптоволоконный кабель, по которому передаются световые сигналы, генерируемые микролазером.

Компьютеры могут использовать для передачи данных обычную телефонную сеть. Такой способ, правда, имеет серьезные недостатки и используется только при отсутствии других каналов передачи.

2.2. Понятие протокола

Фундаментальным понятием в области передачи данных является понятие ***протокола***. Любая передача данных должна подчиняться четко установленным правилам, которые заранее известны всем участникам передачи и строго соблюдаются ими. В самом общем виде ***протокол*** — это набор соглашений о взаимодействиях. Такое значение слова *протокол* несколько отличается от более распространенного значения «последовательная запись происходивших событий», например «протокол допроса», но именно оно используется в области передачи данных.

С протоколами взаимодействия мы сталкиваемся и помимо компьютеров. Особенно важно четко описать взаимодействие в тех областях, где от быстроты и слаженности совместных действий зависит очень многое, например у военных. Недаром имеется четкая форма отдачи приказов и ответов на них. Другой пример — авиадиспетчеры переговариваются с пилотами самолетов только с помощью жестко установленного набора фраз (включающего около пятидесяти фраз), напечатанного в специальной книге. В ней четко указано, что означает каждая фраза, а также перечислены возможные ответы и действия, которые следует выполнять. Использовать во время переговоров другие фразы нельзя, потому что не гарантировано, что они будут пониматься однозначно, а это чревато самыми серьезными последствиями.

Протоколами можно считать правила банковских операций, правила движения (на суше, на воде и в воздухе) и т.п.

Упражнение:

4. Привести примеры протоколов из жизни.

2.3. Протоколы в компьютерных сетях

Взаимодействие между компьютерами всегда устроено очень сложно, и для его описания используют несколько *уровней протоколов*: физический, канальный, сетевой, транспортный, сеансовый и т.д. Работа протокола каждого уровня описывается в терминах протокола более низкого уровня. Такая модель позволяет изменить реализацию протокола низкого уровня, не меняя протоколы верхних уровней.

Например, если необходимо передавать информацию по проводам, то сначала следует договориться о том, какие напряжения будут соответствовать нулям и единицам при двоичном кодировании информации. После этого при описании протокола передачи одного байта можно совсем не употреблять слова *напряжение*, а говорить примерно следующее: «Для посылки байта со значением $i_1i_2i_3i_4i_5i_6i_7i_8$ надо посылать сигнал *1* в течение 150 мксек, потом — сигнал i_1 в течение 100 мксек, потом — сигнал i_2 и т.д.». Описание протокола передачи файлов будет в этом случае использовать только посылку байтов и уже не будет содержать интервалов времени.

Теперь, если нужно передавать информацию не по проводам, а по радио, то, конечно, придется изменить протокол самого нижнего уровня. Можно, например, обозначать единицу сигналом с длиной волны 10,1 метра, а нуль — сигналом с длиной волны 10,2 метра. Однако протоколы более высокого уровня можно не менять — они никак не зависят от способа кодирования нулей и единиц!

Упражнение:

5. Какой протокол надо изменить при изменении скорости передачи данных?

2.4. Какие бывают компьютерные сети

Сети можно разделить на классы по их величине. Небольшие быстрые сети в пределах, как правило, одного здания, объединяющие до сотни компьютеров, называются ***локальными*** (LAN — Local Area Network). Без сомнения, это самый важный класс сетей. Сети между учреждениями в пределах города, объединяющие несколько локальных сетей, расположенных в разных зданиях, называются ***городскими*** (MAN — Metropolitan Area Network). Наконец, ***всемирные*** сети объединяют сотни тысяч компьютеров во всех странах мира (WAN — Worldwide Area Network). Наиболее известная всемирная сеть называется Internet, существуют и другие всемирные сети — Decnet, Usenet, FIDO и т.д. В нашей стране непосредственным продолжением сети Internet является сеть Relcom.

Первые общемировые сети были созданы военными (сеть ARPANET, 1969 г.). Очень быстро, однако, они перестали быть секретными и стали широко использоваться в научных исследованиях. Вначале сети объединяли крупнейшие научные центры, университеты, физические лаборатории и т.п. Затем они быстро завоевали сферу образования и бизнеса. Чрезвычайную популярность приобрела электронная почта, а также другие услуги, предоставляемые сетями, — электронные доски объявлений, журналы, бесплатное распространение открытых компьютерных программ и т.п. В настоящее время всемирные сети доступны любому человеку, имеющему компьютер и модем, который за небольшую плату может подключаться к сети прямо из дома через телефонную линию.

Одноранговые сети и сети типа «клиент-сервер»

Кроме величины, сети можно разделить на одноранговые и сети типа «клиент-сервер».

Одноранговая сеть объединяет равноправные компьютеры, каждый из которых может использовать ресурсы другого. В одноранговой сети пользователю, работающему за любым компьютером, доступны ресурсы всех других компьютеров сети. Например, сидя за одним компьютером, можно редактировать файлы, расположенные на другом компьютере, печатать их на принтере, подключенном к третьему, и запускать программы на четвертом. При этом практически нет разницы, работает ли пользователь с файлами, расположенными на локальном диске (т.е. на том же самом компьютере) или на сетевом. Все диски других компьютеров «видны» так же, как если бы они были расположены на компьютере, за которым работает пользователь.

Недостатки таких сетей — это продолжение их достоинств. В сетях с большим количеством пользователей обычно нежелательно, чтобы все пользователи получали доступ ко всем компьютерам сети. Поэтому одноранговые сети больше всего подходят для

небольших групп, работающих над одним проектом. Недаром одна из популярных сетевых операционных систем такого типа называется Windows for Work Groups.

Операционная система в одноранговой сети должна обеспечивать параллельную работу нескольких процессов на одном компьютере. Идеальный пример такой операционной системы — Unix. Встроенная поддержка одноранговой сети имеется также в операционных системах Windows NT и Windows 95. Одноранговые сети существуют и в системе MS DOS — это LANtastic, NetWare Lite, NetWare Personal и др. Правда, такие сети в системе DOS менее надежны, потому что DOS — не многозадачная система.

Наиболее популярны ***сети типа «клиент-сервер»***. *Сервер* — это компьютер, который предоставляет в сеть свои ресурсы или обеспечивает выполнение определенных услуг. Самый распространенный тип сервера — это файловый сервер, обеспечивающий хранение и использование большого числа файлов. Бывают серверы печати, почтовые серверы, серверы баз данных и в самом общем случае — серверы приложений, на которых можно запускать любые программы. Часто сервер совмещает несколько функций — например, файловый и почтовый сервер одновременно.

Клиент — это компьютер (или, более правильно, программа), который пользуется услугами сервера. Как правило, к клиенту нет доступа из сети: он может только брать, но ничего не отдает. Клиент может не опасаться компьютерных вирусов или взлома защиты, если, конечно, не будет переписывать из сети сомнительные программы.

Всякий сервер работает одновременно с большим количеством клиентов, поэтому нежелательно, чтобы при этом сервер использовался и как обычный компьютер — например, для отладки программы. Любой сбой, «зависание» при работе сервера может иметь катастрофические последствия для пользователей. В идеале на сервере не должно быть клавиши выключения питания (чтобы невозможно было случайно нажать на нее), клавиатура и экран тоже не обязательны. Доступ к серверу ограничивается небольшим кругом лиц — обычно это администратор и оператор сети. Сервер, как правило, работает круглосуточно.

2.5. Сеть Internet

Internet — самая крупная из существующих на сегодняшний день в мире всемирная компьютерная сеть, объединяющая более 300 000 компьютеров во всех развитых странах мира. Без сомнения, по своему значению Internet — сеть номер один. Название *Internet* отражает тот факт, что эта сеть представляет собой множество разнородных сетей, объединенных в одну общую «интерсеть».

Internet была основана в 1969 г. и называлась тогда ARPANET. Первоначально она использовала выделенные телефонные линии. В настоящее время в Internet используются все существующие виды передачи, причем основу составляют оптоволоконные линии и спутниковые системы. В основном в Internet объединяются компьютеры с операционной системой Unix, хотя возможно подключение компьютеров и с другими операционными системами.

2.6. Электронная почта

Спектр услуг, предоставляемых сетью Internet, весьма широк, но, пожалуй, самая популярная из них — это электронная почта. С появлением Internet люди стали гораздо больше писать друг другу и почти перестали пользоваться услугами обычной почты. Электронное письмо доходит до адресата в считанные минуты (если не секунды), причем процедуры отправки писем, чтения полученных писем и ответа на них крайне просты (не надо искать конверт или почтовый ящик). Ответ на полученное письмо потребует лишь нескольких секунд, если он короткий, а так чаще всего и бывает. В ответное письмо можно включить цитаты из полученного письма, используя специальный текстовый редактор. Можно даже настроить машину так, чтобы она автоматически отвечала на приходящие письма и сообщала, к примеру, что вы уехали в отпуск и вернетесь к такому-то числу.

Адреса компьютеров в Internet состоят из четырех байтов, которые обычно изображаются в виде четырех десятичных чисел, разделенных точками, например, `129.144.50.56` (в качестве примеров даются несуществующие адреса!). Такие адреса, однако, неудобны для людей, поэтому существует параллельная система символических адресов. Они организованы в виде иерархической структуры. Вся сеть делится на несколько областей (например, для России область называется `ru` или `su`). Области верхнего уровня делятся на подобласти, те, в свою очередь, на подобласти еще более низкого уровня, и т.д.

В примере на рисунке 1 компьютер с именем `hobbiton` входит в область `earth` и подобласть `shire`. Компьютер с именем `rivendell` входит в область `earth`. Полный адрес компьютера `hobbiton` равен `hobbiton.shire.earth`, компьютера `rivendell` — `rivendell.earth`.

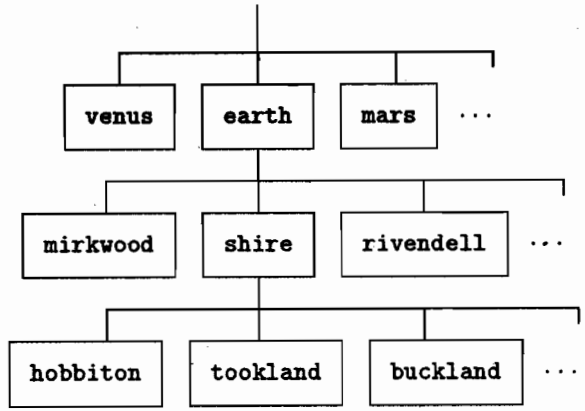


Рис. 1. Иерархическая структура Internet

Пользователь frodo компьютера hobbiton имеет адрес

frodo@hobbiton.shire.earth

пользователь bilbo компьютера rivendell — адрес

bilbo@rivendell.earth

Имя пользователя отделяется от адреса компьютера значком @ (читается «эт коммерческое»). Запись адреса соответствует западноевропейской: сперва указывается имя адресата, затем дом, улица и город, название страны записывается последним.

Еще один пример адреса в Internet:

aivanov@abcd.msk.ru

Здесь имя пользователя — aivanov, адрес компьютера — abcd.msk.ru. Компьютер находится в России (ru), в Москве (msk); имя компьютера — abcd.

Упражнение:

6. Приведите примеры известных вам адресов в Internet.

2.7. Сравнение почтовой и компьютерной сети

Имеется глубокая аналогия между работой обычной почты и работой компьютерной сети. Дело в том, что современные компьютеры *никогда не передают информацию в виде непрерывного потока*. Информация передается небольшими порциями ограниченной длины — так называемыми ***пакетами***. Каждый пакет, кроме передаваемых данных, обязательно содержит служебный заголовок, в котором указаны адрес назначения и адрес источника (обратный адрес), а также та или иная служебная информация (она определяется используемым протоколом передачи). Пакеты аналогичны письмам, заголовки пакетов — конвертам с адресами и специальными пометками.

Человек, передающий письмо на почту, обычно не интересуется, по какому маршруту оно будет следовать к пункту назначения. Точно так же маршрут пакета в компьютерной сети заранее не фиксирован. Так же как письмо по пути к адресату может доставляться различными видами транспорта и проходить через различные почтовые учреждения, так и пакет, переданный компьютером, может проходить через разные промежуточные компьютеры и даже разнородные сети, прежде чем достигнет пункта назначения.

Очень важно также, что перед опусканием письма в почтовый ящик не нужно выполнять никаких предварительных действий (в отличие от телефона, где нужно набрать номер и, самое главное, дождаться, пока телефон, по которому вы звоните, будет свободен). Точно так же в компьютерной сети пакет передается в сеть практически мгновенно, без всяких предварительных процедур установки соединения. Компьютерная сеть, в отличие от телефона, никогда не бывает занята; при сильной загрузке просто увеличивается время доставки пакетов.

Упражнение:

7. Пропускная способность сети типа Ethernet составляет 10 мегабит/сек, размер пакета равен 1024 байта. Сколько пакетов в секунду можно передать по сети?

2.8. Проект WWW

Самым интересным и удобным видом сервиса, появившимся в последнее время в Internet, является World Wide Web (дословно «Всемирная паутина»). Существует несколько программ для работы с WWW: Mosaic, Netscape, Hot Java и др. Все они позволяют просматривать (и прослушивать!) гипертексты, расположенные на WWW-серверах по всему миру. ***Гипертекст*** — это текст, который как бы является проводником в мире информации. По нему можно путешествовать с помощью графического терминала и мыши. Гипертекст подобен меню и содержит выделенные места, «нажав» на которые с помощью мыши пользователь попадает либо в другое место того же самого гипертекста, либо в другой гипертекст. Но, в отличие от обычных гипертекстов, расположенных на одном компьютере, в WWW новый гипертекст может быть расположен на другом компьютере в другой части света. Попав

в другой гипертекст, можно затем вернуться обратно либо продолжать путешествие все дальше и дальше. Подобным образом можно получить информацию, относящуюся практически к любой сфере жизни. Кроме текстов, можно просматривать на экране изображения и даже прослушивать звук. Нужную информацию можно скопировать на свой компьютер. WWW как бы превращает весь мир в единое информационное пространство. Отметим, что большинство программ для работы с WWW распространяются свободно. Это соответствует традиции открытых систем и свободного обмена информацией, широко поддерживаемой в сети Internet.

§ 3. ЧТО ТАКОЕ МОДЕМ

Модем предназначен для передачи данных между компьютерами по обычной телефонной сети. Информация хранится и передается компьютерами в цифровом виде (т.е. в виде последовательности нулей и единиц), однако существующие телефонные линии могут передавать только аналоговые сигналы (попросту говоря, звук). Поэтому приходится сначала преобразовывать информацию в звуковые колебания, кодируя нули и единицы тонами разной частоты или фазы, затем передавать ее в такой форме по телефонной линии и на другом конце линии выполнять обратное преобразование.

Этим и занимается модем, объединяющий в себе два логических устройства: *модулятор*, т.е. преобразователь из цифровой в аналоговую форму, и *демодулятор* — обратный преобразователь.

Такой способ передачи цифровой информации не очень удачен. Тенденция развития современных средств телекоммуникации как раз обратная: представлять аналоговую информацию (звук, изображение) в цифровом виде. Это позволяет почти полностью избавиться от помех, увеличить скорость и гибкость передачи, выполнять обработку изображений и т.д. Сравните качество звучания грампластинки, на которой звук записан в аналоговом виде, и лазерного диска, хранящего цифровую информацию. Скорость и надежность передачи по модему невысоки (максимальная скорость — 19 600 бит в секунду) и совершенно неудовлетворительны для современных компьютерных сетей.

Тем не менее огромным достоинством модема является то, что он позволяет использовать уже существующие телефонные линии, не дожидаясь создания инфраструктуры компьютерных сетей, и уже сейчас пользоваться некоторыми услугами всемирных компьютерных сетей.

3.1. Что позволяет делать модем

Не выходя из дому, можно получить пришедшую электронную почту или отправить письмо в любую часть света. Можно переписать к себе нужный файл, например текст статьи, лежащий в компьютере вашего коллеги в другой стране. Можно войти в любой Unix-компьютер, подключенный к сети Internet, и поработать на нем, не выходя из дому, даже если этот компьютер расположен на противоположной части земного шара (нужно только иметь входное имя на этом компьютере и знать пароль). Причем такое подключение к глобальной сети стоит совсем недорого и вполне доступно всем, кому это необходимо.

3.2. Как работает модем

Модем представляет нули и единицы в виде колебаний звуковой частоты. Возможны несколько способов такого представления — амплитудная, частотная, фазовая модуляции. Современные модемы используют фазовую модуляцию — резко меняют фазу синусоидального сигнала при фиксированной частоте.

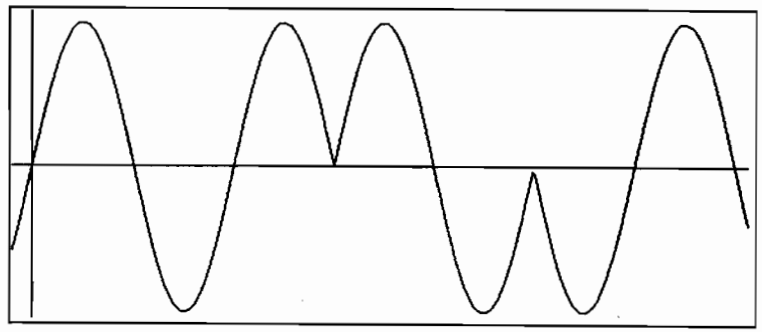


Рис. 2. Фазовая модуляция

Информация передается модемом в виде последовательности байтов — групп по 8 бит. К каждому байту добавляется один или несколько служебных битов (старт-бит, стоп-бит). Байты могут передаваться по отдельности либо объединяться в пакеты, причем для каждого пакета передается контрольная сумма. При несовпадении контрольной суммы передача пакета повторяется.

При установке соединения два модема автоматически договариваются между собой о максимально возможной для обоих скорости передачи и выборе коммуникационного протокола (со сжатием или без сжатия данных, с коррекцией или без коррекции

ошибок). Более скоростной модем может связываться с медленным, установив уменьшенную скорость передачи.

Дешевые модемы обеспечивают скорость передачи 2400 бит/с, дорогие — до 19 600 бит/с. Более высокая скорость в существующих телефонных сетях теоретически недостижима, поскольку в качестве несущей может использоваться только звуковая частота, которая недостаточно велика.

3.3. Как программировать модем

Чуть-чуть упрощая реальную ситуацию, можно считать, что модем устроен следующим образом. В компьютере имеется коммуникационный порт, через который можно передавать байты в модем и принимать байты от модема. (У компьютера IBM PC четыре коммуникационных порта, модем подключается к одному из них.) Можно считать, что передаваемые и принимаемые байты образуют два независимых потока. В любой момент можно и передавать байты через порт, и считывать байты из порта.

Модем может находиться в одном из двух режимов. В *режиме команд* модем интерпретирует передаваемые ему через порт байты как команды. Эти команды либо меняют внутреннее состояние модема (содержимое его внутренних регистров), либо заставляют модем выполнить некоторое действие. Любая команда (кроме команд `A\` и `+++`) начинается с двух символов `AT` и заканчивается символом перевода строки. Например, команда `AT L3` заставляет модем установить максимальную громкость его встроенного динамика. Команда `AT S0=0` запрещает модему самому «снимать телефонную трубку», т.е. отвечать на приходящие звонки. Команда `AT S0=2` предписывает модему снимать трубку после двух звонков. Таких AT-команд несколько десятков; все общение компьютера с модемом осуществляется с их помощью. Наиболее часто используемая команда — это команда набора телефонного номера:

```
AT DP 123-45-67
```

`D` — от слова *Dial*; буква `P` используется при импульсном наборе номера (англ. *Pulse*), принятом в нашей стране (при тональном наборе используется префикс `DT`). Важно отметить, что в режиме команд модем ничего не пересылает по телефонной линии другому модему.

В *режиме данных* модем пересылает все байты, передаваемые ему компьютером через коммуникационный порт, другому модему. Способы пересылки зависят от протокола общения двух модемов, выбор которого определяется конструкцией модемов, их

текущим состоянием, качеством телефонной линии и т.п. Выбор происходит на аппаратном уровне в момент соединения модемов. Выбор протокола — это внутреннее дело двух модемов. С точки зрения коммуникационной программы от этого выбора ничего не зависит (разве что скорость и надежность передачи).

Для перевода модема из режима команд в режим данных имеются специальные команды (`ATO`, `ATA` и `ATD`). В них, однако, обычно нет необходимости: после набора телефонного номера и успешного соединения модем автоматически переходит в режим данных.

Для перехода из режима данных в режим команд нужно:
1) выдержать паузу: в течение так называемого «защитного времени» (обычно это одна секунда) ничего не передавать модему;
2) передать модему последовательность `+++` (три знака плюс);
3) снова выдержать паузу.

После этого модем переходит в режим команд.

Коды результата

В режиме команд модем сообщает компьютеру результат выполнения каждой команды. Он передает компьютеру через коммуникационный порт текстовую строку последовательно по одному символу. Строка заканчивается символом новой строки (символом возврата каретки). Коды результата передаются от модема компьютеру точно так же, как и данные, принимаемые от удаленного модема, однако обычно путаницы не возникает благодаря соглашениям о передаче (т.е. протоколам высокого уровня). Вот некоторые коды результатов:

`OK`	– команда успешно выполнилась
`ERROR`	– неправильная команда
`CONNECT 1200`	– модем соединился на скорости 1200 бит/с
`CONNECT 2400`	– модем соединился на скорости 2400 бит/с
`BUSY`	– занято (короткие гудки)
`NO CARRIER`	– ошибка соединения или потеря несущей
`RING`	– обнаружен звонок
`NO DIALTONE`	– нет непрерывного гудка

3.4. Система предписаний исполнителя Модем

Наш исполнитель Модем умеет делать следующие действия:

`начать работу с модемом (арг цел порт)`

Эта команда служит для установления связи между компью-

тером и модемом. Значение аргумента порт (номер порта модема) должно быть от 1 до 4.

передать байт (арг сим байт)

Команда для передачи данных модему. Указанный байт просто передается через коммуникационный порт.

передать строку (арг лит строка)

Строка передается последовательно по одному символу. В конце добавляется символ перевода строки (символ *возврат каретки*).

принять байт (рез сим байт, рез лог успех)

Байт принимается с ожиданием. На выходе переменная успех принимает значение да, если байт успешно принят, и нет в противном случае (например, когда во время ожидания байта связь прервалась).

принять строку (рез лит стр, рез лог успех)

Строка принимается по одному байту, пока не встретится символ новой строки (символ *возврата каретки*).

цел состояние модема()

Функция возвращает целое число, которое кодирует информацию о состоянии модема следующим образом:

0 **нет связи** — модем находится в командном режиме, и в выходном буфере нет символа; либо в режиме данных связь прервалась;

1 **ожидание байта** — выдается в режиме данных, когда есть связь с удаленным компьютером и модем ждет очередного символа (но выходной буфер модема пуст);

2 **есть байт** — модем готов передать очередной байт компьютеру.

закончить работу с модемом

Эту команду надо отдать в конце выполнения программы.

3.5. Пример: пересылка файла с помощью модема

В качестве простейшего примера передачи данных рассмотрим программу пересылки текстового файла с одного компьютера на другой с помощью модема. Пусть один выделенный компьютер (обозначим его через *A*) обслуживает другие компьютеры, телефонные номера которых зарегистрированы в нем. По запросам других компьютеров он рассылает им некоторый текстовый файл. Это может быть текст телеконференции, новости, исходный текст общедоступной программы, текущий курс доллара и т.п.

Для запроса файла компьютер *B* должен соединиться с *A* через модем и передать ему свой телефонный номер. После этого сеанс связи заканчивается — оба модема «вешают трубки». Компьютер *A* проверяет, значится ли принятый телефонный номер в его списке, и если это так, то в соответствии с очередностью запросов через некоторое время звонит компьютеру *B*. В случае, если компьютер *B* готов в этот момент принять файл, компьютер *A* пересылает ему файл.

Приведем две программы, которые запускаются на компьютере *B*. Первая программа с помощью модема звонит компьютеру *A* и передает ему телефонный номер компьютера *B*. Вторая программа ждет, пока позвонит компьютер *A*, и принимает от него файл. Названия программ — Заказать файл и Принять файл. Программы используют базовый исполнитель Модем.

Первая программа для исполнителя Модем

Первая программа дозванивается до компьютера, рассылающего файлы по запросам, и передает ему свой номер телефона. После этого программа разрывает связь («вешает трубку») и заканчивает работу.

```
цел порт; порт:= 2          | порт модема
лит нашN; нашN:="123-45-67"| наш телефон

алг Заказать файл (арг лит N)
 дано | N - номер телефона, куда звонить
 надо
нач лог успех
| начать работу(порт,успех)
| если не успех
| | то вывод "Ошибка при инициализации модема"
| | иначе
| | нц 10 раз
| | | набрать номер(N,успех)
| | | если успех
| | | | то передать строку(нашN)
| | | | выход
| | | все
| | кц
| | повесить трубку
| все
| закончить работу с модемом
кон
```

Эта программа использует три вспомогательных алгоритма. В программе анализируются результаты их выполнения. В случае успешной установки связи модему передается строка, содержащая наш телефонный номер (и модем пересылает ее по телефонной линии). В случае неуспеха делается следующая попытка дозвониться (всего делается не более десяти попыток).

Первый вспомогательный алгоритм начинает работу с модемом и устанавливает необходимые для работы параметры:

```
алг начать работу (арг цел порт,рез лог успех)
нач лит результат
| начать работу с модемом(порт)
| передать строку("AT E0Q0X4B1&D2&C1")
| принять строку(результат,успех)
| успех:=успех и (результат = "OK")
кон
```

После начала работы модему передается командная строка

"AT E0Q0X4B1&D2&C1"

которая устанавливает режим работы модема. Эта командная строка содержит несколько элементарных команд. Например, символы E0 отключают «эхо» при посылке команд в модем (т.е. модем не возвращает символы, которые передаются в него), а символы Q0X4 задают набор кодов результата, которые будет посылать модем.

При получении командной строки модем выполняет команду и всегда возвращает результат в виде текстовой строки. Ее надо принять из модема, проанализировать и в зависимости от результата предпринять те или иные действия.

Второй вспомогательный алгоритм командует модему набрать номер:

```
алг набрать номер (арг лит N, рез лог успех)
нач лит команда,результат
| команда:="AT DP "+N | команда звонка
|                     | по указанному телефону
| передать строку(команда)
| принять строку(результат,успех)
| вывод результат
| успех:=успех и (результат[1:7]="CONNECT")
кон
```

После приема командной строки "AT DP 987-65-43" от компьютера модем звонит по номеру 987-65-43 и в случае успешного соединения выдает текстовую строку, которая начинается со слова

CONNECT. После этого модем переходит в режим данных, в котором он автоматически пересылает все байты, передаваемые ему компьютером, удаленному модему.

Последний вспомогательный алгоритм приказывает модему повесить трубку:

```
алг повесить трубку
нач лит результат, лог успех
| пауза(2)                  | перейти
| передать строку("+++")    | в командный
| пауза(2)                  | режим
| передать строку("AT H")
| принять строку(результат,успех)
| вывод результат
кон
```

После успешной передачи в течение двух секунд модему ничего не передается. Затем модему передается последовательность "+++" (три символа плюс) и снова выдерживается пауза. Это приводит к тому, что модем переходит в командный режим. Затем модему передается команда "AT H", заставляющая его разорвать соединение («повесить трубку»). После этого программа завершает работу.

Вторая программа для исполнителя Модем

Вторая программа ожидает звонка от компьютера, который должен передать файл. В случае звонка модем самостоятельно «поднимает трубку» и устанавливает связь с удаленным модемом. Затем принимаются все байты, пока удаленный модем не разорвет соединение. Принятые байты по мере их поступления записываются в файл RECEIVE.TXT.

```
цел порт; порт:=2 | номер порта модема

алг Принять файл
нач лог успех, цел статус, сим байт
| начать работу(порт,успех)
| если успех то команда("AT S0=1",успех) все
| если не успех
| | то вывод "Ошибка при инициализации модема"
| | иначе
| |  ждать звонка
| |  начать запись("RECEIVE.TXT")
| |  нц
| |  | статус:=состояние модема()
```

```
        если статус = 2 | т.е. есть байт
          то принять байт(байт,успех)
          вывод байт
        все
      кц при статус = 0
      кончить запись
      повесить трубку
    все
    закончить работу с модемом
кон
```

Эта программа, кроме приведенных выше вспомогательных алгоритмов начать работу и повесить трубку, использует еще два. Первый просто передает команду модему и проверяет результат ее выполнения:

```
алг команда (арг лит X, рез лог успех)
нач лит результат
  передать строку(X)
  принять строку(результат,успех)
  успех:=успех и (результат = "OK")
кон
```

Следующий вспомогательный алгоритм ждет поступления звонка:

```
алг ждать звонка
нач лит результат, лог успех
  нц
    если состояние модема() = 2
    то
      принять строку(результат,успех)
      вывод результат
      если успех и результат[1:7]="CONNECT"
      то выход
      все
    все
  кц
кон
```

Глава 2 ИНФОРМАЦИОННЫЕ МОДЕЛИ

В этой главе мы познакомимся с понятием информационной модели и моделированием действительности на ЭВМ.

Обрабатываемая на ЭВМ информация должна быть представлена в памяти компьютера в виде значений величин используемого языка программирования. Мы используем школьный алгоритмический язык и потому будем кодировать информацию с помощью величин этого языка.

Набор величин, содержащий всю необходимую информацию об исследуемых объектах и процессах, в информатике называется ***информационной моделью***.

Информационная модель содержит не всю информацию о моделируемых явлениях, а только ту ее часть, которая нужна для рассматриваемых задач. То, что не нужно для решения поставленных задач, при моделировании отбрасывается. Если круг решаемых задач расширяется, то приходится расширять и модель, включать в нее больше информации. Даже когда круг решаемых задач фиксирован, информационную модель можно строить многими разными способами.

§ 4. ПРОСТЕЙШИЙ ПРИМЕР ИНФОРМАЦИОННОЙ МОДЕЛИ

Пусть в кинозале 28 рядов по 20 мест, а цена билета зависит только от номера ряда. Предположим, что мы собираемся с помощью компьютера организовать предварительную продажу билетов на один-единственный сеанс. Какая информация о кинозале должна войти в информационную модель? Как организовать эту информацию?

Во-первых, нам понадобится информация о том, на какие места в зале билеты уже проданы, а на какие — еще нет. При ручной продаже билета перед кассиром обычно лежит **таблица**, в которой изображены все места в кинозале, и по мере продажи билетов кассир делает отметки — ставит «крестики» — в этой таблице. Примерно такая же таблица используется в информационной модели.

4.1. Информация о проданных местах

Введем в кинозале координаты: номер места i и номер ряда j. Договоримся обозначать проданное место числом 1, а непроданное — числом 0. Чтобы запомнить информацию о всех проданных и непроданных местах, будем хранить в памяти ЭВМ прямоугольную таблицу целтаб a[1:20,1:28], в которой:

$$a[i,j] = 1 \Leftrightarrow \text{место продано}$$

$$a[i,j] = 0 \Leftrightarrow \text{место не продано}$$

Как организовать с помощью этой таблицы хранение всей нужной информации о проданных и непроданных местах?

Очень просто. Перед началом продажи надо отметить в таблице, что все места пока свободны, т.е. таблицу надо заполнить нулями:

```
нц для j от 1 до 28    | для каждого ряда
| нц для i от 1 до 20  | для каждого места в ряду
| | a[i,j]:=0          | запомнить, что свободно
| кц
кц
```

а при продаже каждого билета надо занести в таблицу на соответствующее место единичку. Например, при продаже билета на место 1 в ряду 28 нужно выполнить команду

```
a[1,28]:=1
```

С помощью такой таблицы можно получать информацию о ходе продажи билетов. Например, можно подсчитать число n уже проданных билетов:

```
n:=0
нц для j от 1 до 28
| нц для i от 1 до 20
| | если a[i,j]=1 то n:=n+1 все
| кц
кц
```

4.2. Информация о ценах

Но подсчитать стоимость уже проданных билетов не удастся, пока мы не включим в модель информацию о ценах на билеты. Поскольку, по предположению, все места в ряду имеют одинаковую цену, эту информацию можно организовать в виде линейной таблицы целтаб цена[1:28]:

цена[j] = цена билета в j-м ряду в рублях

Предположим, что стоимость билета зависит от номера ряда так:

ряды от 1 до 6 — 2 тыс. р.
ряды от 7 до 20 — 3,5 тыс. р.
ряды от 21 до 28 — 2,5 тыс. р.

Перед началом продажи эту информацию нужно занести в таблицу:

```
нц для j от 1 до 28
| если  1<=j<= 6 то цена[j]:=2000 все
| если  7<=j<=20 то цена[j]:=3500 все
| если 21<=j<=28 то цена[j]:=2500 все
кц
```

Теперь в модели уже достаточно информации для подсчета выручки s в любой момент продажи. Выручка складывается из цен всех проданных мест:

```
s:=0
нц для j от 1 до 28    | для каждого ряда
| нц для i от 1 до 20  | для каждого места в ряду
| | если a[i,j]=1      | если место продано
| | то s:=s+цена[j]    | то учесть его цену
| | все
| кц
кц
```

4.3. Другая модель кинозала

При построении модели всегда есть возможность свободного выбора. Модель кинозала можно было организовать и по-другому. Например, мы условились, что в таблице a[1:20,1:28] первый индекс означает номер места в ряду, а второй — номер ряда. Вместо этого можно было бы сделать наоборот.

Таблицу цена [1:28] мы выбрали целочисленной и заносили в нее цену в рублях. Вместо этого можно было бы сделать таблицу вещественной и хранить цену в тысячах рублей:

```
нц для j от 1 до 28
| если  1<=j<= 6 то цена[j]:=2.0 все
| если  7<=j<=20 то цена[j]:=3.5 все
| если 21<=j<=28 то цена[j]:=2.5 все
кц
```

При таком подходе цены выглядели бы более наглядно.

Наконец, все места в кинозале можно было бы пронумеровать подряд (их всего 20 × 28 = 560) и вместо прямоугольной ввести линейную таблицу a[1:560]. Но такой подход доставил бы массу неприятностей.

Во-первых, в реальном кинозале никакой сквозной нумерации мест нет, а места задаются номером ряда и номером кресла в ряду. Поэтому пришлось бы постоянно преобразовывать информацию из одной формы в другую, что неудобно и повышает вероятность ошибок. Во-вторых, алгоритмы получились бы сложнее. Попробуйте составить алгоритм подсчета выручки по таблицам a[1:560] и цена[1:28], и вы поймете, что такая модель была бы очень неудобна.

4.4. Алгоритмы тоже можно написать по-другому

Когда модель уже выбрана, алгоритмы извлечения информации из модели тоже можно написать по-другому. Например, если в таблице a[1:20,1:28] проданным местам соответствуют единицы, а непроданным — нули, то количество проданных мест n можно подсчитать просто как сумму всех элементов таблицы a[1:20,1:28]:

```
n:=0
нц для j от 1 до 28
| нц для i от 1 до 20
| | n:=n+a[i,j]
| кц
кц
```

А выручку s можно подсчитать, используя следующий факт. Для проданного места i,j произведение a[i,j]*цена[j] равно 1*цена[j], а для непроданного — 0*цена[j], т.е. в любом случае произведение равно вкладу данного места в общую выручку. Алгоритм подсчета выручки может выглядеть так:

```
s:=0
нц для j от 1 до 28       | для каждого ряда
| нц для i от 1 до 20     | для каждого места в ряду
| | s:=s+a[i,j]*цена[j]   | учесть вклад места
| кц
кц
```

4.5. Какую информацию учитывает модель

Описанная модель кинозала весьма схематична — она рассчитана только на один сеанс, не учитывает возможности бронирования мест, продажи абонементов, скидок в цене пенсионерам и т.п. В модели также не учитывается, например, количество входов и выходов из кинозала, наличие или отсутствие проходов в зале и пр.

В одних случаях такая информация важна, в других нет. Например, для подсчета выручки или числа проданных билетов информация о проходах не нужна. Но если требуется найти два места рядом, информация о проходах становится существенной.

Если каждый ряд кресел «непрерывен», не разделяется ни одним проходом, то найти два свободных места можно так:

```
нашли:="нет"; i:=1; j:=1
нц пока нашли="нет" и j<=28
| если a[i,j]=0 и a[i+1,j]=0
| | то нашли:="да"
| | иначе i:=i+1
| | если i=20 то j:=j+1; i:=1 все
| все
кц
```

Значение величины лит нашли дает информацию о результатах поиска. Если нашли="да", то два места i и i+1 в j-м ряду одновременно свободны. Если же после завершения цикла нашли="нет", то в зале уже нет двух свободных мест рядом.

В этом алгоритме перебор мест производится так. Сначала рассматриваются места 1 и 2 в ряду 1. Затем i увеличивается на 1, и рассматриваются места 2 и 3 в том же ряду и т.д. Если в первом ряду двух свободных мест рядом нет, то после рассмотрения пары 19, 20 величина i будет увеличена до 20 и сразу после этого номер ряда будет увеличен на 1, а номер места станет равным 1, т.е. все будет готово для анализа мест 1 и 2 во втором ряду.

Если же и в последнем ряду с номером 28 двух свободных мест не найдется, то величина j станет равна 29, что приведет к завершению цикла.

4.6. Расширение модели информацией о проходах

Если в зале есть проходы, разделяющие ряды, то этот алгоритм может найти два места с соседними номерами, которые, увы, окажутся разделены проходом. Никакая модификация алгоритма поиска соседних мест тут не спасет. Дело в том, что в нашей модели информации о расположении проходов просто нет. А значит, никакими самыми хитроумными алгоритмами ее отту-

да извлечь нельзя. Так что если мы расширяем круг задач, которые обслуживает модель, задачей поиска двух соседних свободных мест, то придется расширить модель.

Предположим, что в кинозале есть три части, разделенные двумя проходами:

левая часть	— места с номерами от 1 до 6;
центральная часть	— места с номерами от 7 до 14;
правая часть	— места с номерами от 15 до 20.

Тогда места 6 и 7 в любом ряду уже не будут соседними, равно как и места 14 и 15. Для хранения информации о расположении проходов введем две величины

```
цел i1 | номер последнего места в левой части зала
цел i2 | номер последнего места в центральной части зала
```

и присвоим им значения 6 и 14 соответственно.

Теперь в нашей модели достаточно информации для нахождения соседних мест. При поиске соседних свободных мест нужно исключить в предыдущем алгоритме случаи, когда в паре найденных свободных мест i,i+1 место i оказывается последним в левой или центральной части. А это случается, только если i=i1 или i=i2:

```
нашли:="нет"; i:=1; j:=1
нц пока нашли="нет" и j<=28
| если a[i,j]=0 и a[i+1,j]=0 и i<>i1 и i<>i2
| | то нашли:="да"
| | иначе i:=i+1
| |  если i=20 то j:=j+1; i:=1 все
| все
кц
```

Таким образом, после расширения модели оказалось возможным решать задачу о поиске соседних свободных мест. (Это неудивительно, так как ради этой задачи модель и расширялась.)

4.7. Другие расширения модели — билеты со скидкой

Предположим теперь, что в кинотеатр поступило распоряжение об улучшении обслуживания пенсионеров: билеты им теперь продаются без очереди и со скидкой 50%.

Нужно ли что-нибудь менять в самой модели и в алгоритмах, работающих с этой моделью? Разумеется, обслуживание пенсионеров без очереди никак ни на чем не скажется. Ведь очередь

покупателей организуется не компьютером и никакого отношения к нашей компьютерной модели и алгоритмам не имеет.

А вот продажа билетов со скидкой 50% затрагивает и модель и алгоритмы. На подсчете числа свободных мест или на поиске свободного места скидка никак не сказывается. Но при подсчете выручки скидку придется учитывать. Значит, в модель должна быть добавлена информация, позволяющая правильно подсчитать выручку.

Сделать это можно многими разными способами.

Способ 1. Введем величину цел скидка, равную суммарной сумме скидок в рублях, сделанной при продаже билетов. Перед началом продажи билетов запоминается, что скидка равна нулю, а при продаже одного билета на место i,j со скидкой выполняются команды:

```
a[i,j]:=1; скидка:=скидка+0.5*цена[j]
```

Теперь для подсчета выручки s нужно из подсчитанной полной цены вычесть скидку, которая была сделана. Недостаток этого способа в том, что в модели не сохраняется информация о том, на какие места проданы билеты со скидкой. Такая информация может понадобиться при проверке добросовестности кассира. Попробуем включить в модель эту информацию.

Способ 2. Изменим правила хранения информации в таблице a[1:20,1:28]. Будем по-прежнему считать, что непроданным местам соответствуют нули, а вот для проданного места будем заносить в таблицу не 1, а процент оплаты стоимости билета:

a[i,j] = 100, если билет оплачен на 100%,
a[i,j] = 50, если билет оплачен на 50%.

После такой переделки понадобится изменить все алгоритмы. Например, подсчет числа проданных мест теперь будет выглядеть так:

```
n:=0
нц для j от 1 до 28
| нц для i от 1 до 20
| | если a[i,j]>0 то n:=n+1 все
| кц
кц
```

Теперь проданному месту в таблице соответствует не 1, а 100 или 50, так что нужно подсчитывать число положительных элементов таблицы.

Алгоритм подсчета выручки тоже придется изменить:

```
s:=0
нц для j от 1 до 28
| нц для i от 1 до 20
| | s:=s+a[i,j]*цена[j]/100
| кц
кц
```

Если a[i,j]=100, то выражение a[i,j]*цена[j]/100 равно полной цене места, а если a[i,j]=50, то это выражение равно половине цены места.

Достоинства этого способа в том, что при необходимости он позволяет варьировать размеры скидки или продавать билеты с разными скидками. Например, детям до 12 лет — со скидкой 70%. Но, увы, и у этого способа есть недостатки: если потребуется какой-то категории покупателей выдавать бесплатный билет, нельзя будет это сделать, пользуясь нашей моделью. При скидке 100% в таблицу a[1:20,1:28] по нашим правилам нужно занести 0 процентов стоимости билета, а такое место нельзя отличить от свободного.

4.8. Окончательная версия модели кинозала

В разработке компьютерной системы обычно принимают участие заказчик и исполнитель. Задача исполнителя — выполнить требования заказчика. Однако на практике заказчик не в состоянии сразу сформулировать все требования к компьютерной системе. Заказчик выдает свои требования в процессе разработки, а затем и в процессе эксплуатации системы.

Опытный исполнитель заранее предугадывает возможные требования заказчика и старается построить информационную модель так, чтобы по возможности в процессе развития системы модель только расширялась, а ее готовые части и работающие с ними алгоритмы менялись как можно меньше.

Так что на практике информационная модель всегда составляется с некоторым «запасом».

Попробуем составить «с запасом» модель кинозала. Ниже будем называть ее модель М1.

Прежде всего введем две величины imax и jmax:

```
цел imax | количество мест в ряду
цел jmax | количество рядов в зале
```


где imax=20 и jmax=28. Это позволит в дальнейшем при необходимости легко использовать модель для прямоугольных залов других размеров.

Далее введем таблицу

```
цел таб a[1:imax,1:jmax]
```

для хранения информации о местах. Какая информация будет храниться и в каком виде, обсудим позже, пока только договоримся, что свободному месту в кинозале будет соответствовать ноль в этой таблице. Перед началом продажи в эту таблицу нужно занести информацию о том, что все места свободны (a[i,j]=0 ⇔ место i ряда j не продано):

```
нц для j от 1 до jmax    | для каждого ряда
| нц для i от 1 до imax  | для каждого места в ряду
| | a[i,j]:=0            | запомнить, что свободно
| кц
кц
```

Как и раньше, для хранения информации о проходах введем величины

```
цел i1 | номер последнего места в левой части зала
цел i2 | номер последнего места в центральной части зала
```

где i1=6 и i2=14.

Как и раньше, будем предполагать, что полная цена места зависит только от номера ряда j:

```
цел таб цена[1:jmax]
| цена[j] = цена билета в j-м ряду в рублях
```

Причем

```
для рядов с номерами от    1 до j1   цена равна c1
для рядов с номерами от j1+1 до j2   цена равна c2
для рядов с номерами от j2+1 до jmax цена равна c3
```

где j1=6, j2=20, c1=2000, c2=3500, c3=2500.

Перед началом продажи в эту таблицу нужно занести информацию о ценах:

```
нц для j от 1 до jmax
| если  1<=j<=j1   то цена[j]:=c1 все
| если j1< j<=j2   то цена[j]:=c2 все
| если j2< j<=jmax то цена[j]:=c3 все
кц
```

Теперь осталось решить, как будут кодироваться в таблице a[1:imax,1:jmax] билеты, проданные за полную цену; билеты, проданные с той или иной скидкой, и т.д. Сделаем это так:

a[i,j]=0 ⇔ не продано
a[i,j]=1 ⇔ продано за полную стоимость
a[i,j]=2 ⇔ продано пенсионеру за 50% стоимости
a[i,j]=3 ⇔ продано за 0% стоимости (бесплатно)
a[i,j]=4 ⇔ забронировано, но не оплачено

Значения 5, 6, 7, 8 и 9 оставим в резерве. Если заказчик введет какие-то новые виды бронирования или продажи билетов, эти значения будут использованы. Это и есть «запас» исполнителя для возможного расширения модели. Единственное, что осталось, — более удобно организовать информацию, необходимую для подсчета выручки. Для этого введем таблицу

```
целтаб ст[0:9]
```

в которую будем записывать процент уплаты стоимости места каждого вида. Первые элементы этой таблицы: ст[0]=0, ст[1]=100, ст[2]=50, ст[3]=0, ст[4]=0; остальные элементы будут заполняться по мере ввода новых видов услуг.

При использовании такой модели выручку s можно подсчитать так:

```
s:=0
нц для j от 1 до 28
| нц для i от 1 до 20
| | проц:=ст[a[i,j]]
| | s:=s+проц*цена[j]/100
| кц
кц
```

В этом алгоритме проц равно проценту от стоимости полного билета, заплаченному за место i в ряду j, а выражение проц*цена[j]/100 равно сумме (в рублях), заплаченной за указанное место.

Упражнения:

8. Запишите на алгоритмическом языке (для модели М1):
   а) продажу места за полную цену;
   б) продажу места за 50% цены;
   в) бронирование места i в ряду j;
   г) разбронирование места i в ряду j;
   д) разбронирование всех ранее забронированных мест;
   е) поиск одного свободного места как можно ближе к экрану;

   ж) поиск одного свободного места как можно дальше от экрана;
   з) поиск двух соседних свободных мест;
   и) поиск трех соседних свободных мест.
9. Измените модель М1, считая, что места в центре зала стоят дороже, чем места по краям. Запишите на алгоритмическом языке подсчет выручки в измененной модели.
10. Измените модель М1, считая, что в зале проходит 6 сеансов в день: 4 по одной цене и 2 — по другой, а билеты продают только на завтра.
11. В рамках модели М1 напишите алгоритм подсчета выручки, считая, что цена места номер 13 в 13-м ряду равна 20% суммы, указанной в таблице цена[1:jmax], а цена всех остальных мест в 13-м ряду, а также всех 13-х мест в других рядах равна 50% от суммы, указанной в таблице. (Пенсионер и за эти места заплатит вдвое меньше, чем другие зрители.)
12. Составьте информационную модель кинозала, в котором ряды имеют разное количество мест.

§ 5. ИНФОРМАЦИОННАЯ МОДЕЛЬ ТРАНСПОРТНОЙ СЕТИ

Пусть в некоторой стране 100 городов и между некоторыми из них летают самолеты. Авиатрассы проложены так, что из любого города можно перелететь в любой другой (возможно, с пересадками). В каждом городе только один аэропорт, так что нет рейсов из города в тот же самый город.

Построим информационную модель, используя которую можно отвечать на вопросы, как перелететь из одного города в другой, каково минимальное число пересадок и какова минимальная длина пути при таком полете. Эту модель назовем М2:

```
цел n; n:=100 |  количество городов
литтаб назв[1:n]
литтаб прям[1:n,1:n]
вещтаб расст[1:n,1:n]
```

причем

назв[i] – название i-го города
прям[i,j]="да" ⇔ между городами i и j есть прямой авиарейс
расст[i,j] – расстояние между городами i и j

Для любого i от 1 до n выполнено равенство прям[i,i]="нет", так как нет рейсов из города в тот же самый город.

В таблице расст в этой модели хранятся расстояния только между такими городами i и j, для которых прям[i,j]="да".

Используя модель М2, можно, например, заставить ЭВМ проанализировать, как добраться из города i1 в город i2, совершив не более одной пересадки. Ниже используются величины цел p, лит найден (пункт пересадки):

```
вывод "Из города",назв[i1],"в город",назв[i2],нс
если прям[i1,i2]="да"
| то вывод "есть прямой рейс"
| иначе
|  p:=0; найден :="нет"
|  нц пока p<n и найден = "нет"
|  | p:=p+1
|  | если прям[i1,p]="да" и прям[p,i2]="да"
|  | | то найден:="да"
|  | все
|  кц
|  если найден="да"
|  | то
|  |  вывод "есть маршрут с пересадкой в",назв[p]
|  | иначе
|  |  вывод "с одной пересадкой добраться нельзя"
|  | все
| все
```

Предположим, что назв[1]="Севгор", назв[50]="Центгор" и назв[100]="Южгор", причем

прям[1,50]="да", прям[50,100]="да",

а прям[1,100]="нет".

Это означает, что из Центгор есть прямые рейсы в Севгор и Южгор, а вот из Севгор в Южгор прямого рейса нет.

Если запустить алгоритм для i1=1 и i2=50, то будет напечатано:

```
Из города Севгор в город Центгор
есть прямой рейс
```

А если мы запустим алгоритм для i1=1 и i2=100, то p будет последовательно принимать значения 1, 2, ... до тех пор, пока не встретится город, соединенный прямыми рейсами с Севгор и Южгор одновременно. Если первый такой номер равен 50, то будет напечатано:

```
Из города Севгор в город Южгор
есть маршрут с одной пересадкой в Центгор
```


Упражнения:

13. В рамках модели М2 запишите на алгоритмическом языке фрагмент алгоритма, который выводит на экран:
   а) все маршруты из города i1 в город i2 ровно с одной пересадкой и длину каждого из этих маршрутов;
   б) все маршруты из i1 в i2 с одной пересадкой в порядке возрастания длины маршрута;
   в) кратчайший маршрут из i1 в i2 с одной пересадкой;
   г) сообщение о том, можно ли добраться из i1 в i2 с двумя пересадками.

14. Дано, что линейная таблица цел таб p[1:m] содержит номера всех городов, в которые можно добраться из i1, делая не более двух пересадок. В рамках модели М2 запишите фрагмент алгоритма, который выводит на экран сообщение о том, можно ли добраться из i1 в i2, делая не более трех пересадок.

§ 6. ИНФОРМАЦИОННЫЕ МОДЕЛИ ГЕОМЕТРИЧЕСКОЙ ИНФОРМАЦИИ

Основы числового кодирования геометрической информации заложил великий французский ученый XVII в. Рене Декарт: любую точку плоскости можно задать парой чисел — ее ***декартовыми координатами***. А от числового кодирования точек нетрудно перейти к кодированию более сложных геометрических объектов, например фигур школьной планиметрии. Любая компьютерная система, позволяющая проводить геометрические операции, базируется на моделях подобного типа.

Для одного и того же геометрического объекта можно придумать много моделей. Для одних операций над объектами будут удобны одни модели, а для других — другие. Поэтому полезно для данного объекта заготовить несколько моделей и при необходимости переходить от одной к другой. Вот несколько моделей для хорошо известных геометрических объектов на плоскости. Большинство из этих моделей используется в системе ПланиМир, о которой пойдет речь ниже.

М3. Точка.

```
вещ x,y | координаты точки
```

М4. Окружность.

```
вещ r   | радиус окружности
вещ a,b | координаты центра
```

М5. Прямая.

```
вещ x1,y1 | координаты двух
вещ x2,y2 |  точек на прямой
```

М6. Луч.

```
вещ x1,y1 | координаты начала луча
вещ x2,y2 | координаты точки на луче
```

М7. Отрезок.

```
вещ x1,y1 | координаты начала отрезка
вещ x2,y2 | координаты конца отрезка
```

М8. Треугольник.

```
вещ x1,y1 | координаты
вещ x2,y2 |  вершин
вещ x3,y3 |   треугольника
```

М9. n-угольник.

```
вещтаб x[1:n] | (x[i],y[i]) — координаты i-й вершины
вещтаб y[1:n] |
```

М10. Прямоугольник со сторонами, параллельными осям координат.

```
вещ xmin | Точка (x,y) принадлежит
вещ xmax | прямоугольнику ⇔
вещ ymin |   xmin ≤ x ≤ xmax
вещ ymax |   ymin ≤ y ≤ ymax
```

М11. Квадрат.

```
вещ a,b | центр квадрата
вещ u,v | координаты вектора из центра
        | к одной из вершин квадрата
```

Пользуясь этими моделями, можно переводить геометрические понятия на алгоритмический язык:

Длина отрезка (модель М7):

```
(x1-x2)**2 + (y1-y2)**2
```

Точка (модель М3) внутри окружности (модель М4):

```
(x-a)**2 + (y-b)**2 < r**2
```

Квадрат длины диагонали n-угольника (модель М9):

```
(x[i]-x[j])**2+(y[i]-y[j])**2
```


Периметр n-угольника (модель М9):

```
p:=0
нц для i от 1 до n-1
| p:=p+длина(i,i+1)
кц
p:=p+длина(n,1)
```

Окружность М4 внутри прямоугольника М10:

```
xmin<a-r и a+r<xmax и ymin<b-r и b+r<ymax
```

Площадь прямоугольника (модель М10):

```
(xmax-xmin)*(ymax-ymin)
```

Площадь квадрата (модель М11):

```
2*(u**2+v**2)
```

Один и тот же геометрический объект можно кодировать по-разному. Например, отрезок можно задать не так, как в М7, а совсем по-другому:

М12. Отрезок.

```
вещ a,b | координаты середины отрезка
вещ l   | длина отрезка
вещ ф   | угол наклона отрезка к оси Ox
```

Наконец, даже одни и те же числа можно представить разными величинами алгоритмического языка, например, для задания точки можно использовать не две отдельные числовые величины, а таблицу

```
вещтаб x[1:2] | точка с координатами (x[1],x[2])
```

Упражнения:

15. Закодируйте величинами алгоритмического языка следующие геометрические объекты на плоскости:
а) угол;
б) пару параллельных прямых;
в) произвольный прямоугольник.

16. Составьте алгоритм `алг вещ S (арг вещ x1, y1,x2,y2,x3,y3)`, вычисляющий площадь треугольника по формуле Герона. Используя его как вспомогательный, напишите фрагменты алгоритмов, которые вычисляют:
а) высоту треугольника М8, опущенную из данной вершины;
б) расстояние от точки М3 до прямой М5;
в) касается ли окружность М4 прямой М5.

17. Напишите формулы перехода от модели отрезка М7 к модели М12 и обратно.

18. Напишите формулы, выражающие координаты всех 4 вершин квадрата через величины модели М11.

19. Запишите на алгоритмическом языке условие «отрезок М7 внутри окружности М4».

20. Даны два прямоугольника со сторонами, параллельными осям координат (модель М10): xmin1, xmax1, ymin1, ymax1 и xmin2, xmax2, ymin2, ymax2. Запишите фрагмент алгоритма, который выводит на экран:
    а) площадь пересечения;
    б) периметр объединения этих прямоугольников.

21. Закодируйте величинами алгоритмического языка следующие геометрические объекты в пространстве:
    а) точка;
    б) сфера;
    в) прямая;
    г) плоскость;
    д) треугольная пирамида;
    е) параллелепипед;
    ж) прямоугольный параллелепипед с ребрами, параллельными осям координат.

§ 7. ИНФОРМАЦИОННАЯ МОДЕЛЬ ОБСТАНОВКИ НА ПОЛЕ РОБОТА

Информационная модель обстановки на поле Робота должна содержать всю информацию, необходимую для выполнения команд Робота. Эту модель мы назовем М14. Напомним, что поле Робота имеет размер: 15 клеток в ширину и 9 клеток в высоту. На этом поле можно ввести координаты, перенумеровав столбцы слева направо от i=1 до i=15, а ряды — сверху вниз от j=1 до j=9. Пользуясь этими координатами, можно задать положение Робота на поле, а также информацию о расположении на поле стен и закрашенных клеток, температуре и радиации следующим образом:

```
цел x | 1 ≤ x ≤ 15, x-координата Робота (номер столбца)
цел y | 1 ≤ y ≤ 9, y-координата Робота (номер строки)
литтаб ниже[1:15,1:9], левее[1:15,1:9]
литтаб выше[1:15,1:9], правее[1:15,1:9]
литтаб закр[1:15,1:9]
вещтаб a[1:15,1:9] | уровень радиации и
вещтаб t[1:15,1:9] | температура в клетке
```


При этом литерные таблицы определены следующим образом:

```
ниже[i,j]="да"   ⇔ ниже стена
выше[i,j]="да"   ⇔ выше стена
левее[i,j]="да"  ⇔ левее стена
правее[i,j]="да" ⇔ правее стена
закр[i,j]="да"   ⇔ клетка закрашена
```

В этой модели есть подводные камни. Информацию в таблицах закр, a и t можно задавать независимо. А вот информация в остальных четырех таблицах должна задаваться согласованно. Если, например, ниже[i,j]="да", то автоматически и выше[i,j+1]="да". Дело в том, что если ниже клетки i,j есть стена, то выше клетки i,j+1 должна быть та же стена.

Пользуясь моделью М14, можно имитировать любые действия по управлению Роботом в прямоугольнике размером 15 на 9:

Управление Роботом	Имитация управления
вправо	x:=x+1
вверх	y:=y-1
закрасить	закр[x,y]:="да"
нц n раз | если температура>100 |то s:=s+радиация все | вниз кц	нц n раз | если t[x,y]>100 |то s:=s+a[x,y] все | y:=y+1 кц
если сверху стена |то влево все	если выше[x,y]="да" |то x:=x-1 все
в левом верхнем углу	x=1 и y=1
в правом нижнем углу	x=15 и y=9

Аналогичным образом задаются информационные модели лабиринтов в различных компьютерных играх. Мы разберем одну такую игру ниже (§ 12).

Упражнение:

22. Как уже говорилось, информация о расположении стен, хранящаяся в таблицах **выше**, **ниже**, **правее** и **левее** модели М14, избыточна. Например, если `правее[1,1]="да"` (т.е. между 1-й и 2-й клетками первого ряда стена), то `левее[2,1]` тоже обязательно равно "да". Измените модель М14 так, чтобы информация о стенах кодировалась более экономно, без дублирования.

§ 8. ИНФОРМАЦИОННАЯ МОДЕЛЬ РИСУНКА НА ПОЛЕ ЧЕРТЕЖНИКА

Чертежник не умеет рисовать ничего, кроме отрезков. Поэтому любой рисунок на поле Чертежника — это конечное число отрезков. Для задания каждого отрезка нужно задать две координаты его концов, т.е. для задания отрезка нужно задать 4 числа. Для задания n отрезков понадобится $4n$ чисел. Поэтому простейшая модель М15 рисунка на поле Чертежника, способная хранить информацию о любой картинке из 1000 или менее отрезков, может быть построена так:

```
цел n              | число отрезков, 0 ≤ n ≤ 1000;
вещтаб a[1:4000]   | координаты отрезков.
```

В этой модели первый отрезок задается четырьмя числами:

`a[1]` - x-координата начала отрезка,
`a[2]` - y-координата начала отрезка,
`a[3]` - x-координата конца отрезка,
`a[4]` - y-координата конца отрезка.

Аналогично задается второй отрезок и все следующие. Полезную информацию в этой модели хранят не все элементы таблицы `a[1:4000]`, а только первые $4n$.

На практике картинка часто состоит не из отдельных отрезков, а из ломаных — последовательностей отрезков, в которых каждый следующий начинается там, где кончается предыдущий. Для задания ломаной из n отрезков нужно задать всего $2n+2$ чисел, а вовсе не $4n$. Но модель М15 не позволяет использовать место в таблице `a[1:4000]` более экономно.

Рассмотрим поэтому модель М16, более подходящую для хранения ломаных. В этой модели для каждой ломаной из k звеньев будет храниться сначала количество звеньев — положительное

число, а затем $2k+2$ координат вершин ломаной. Данные о каждой ломаной будут записываться в таблицу

```
вещтаб a[1:4000]
```

Появление в качестве длины ломаной нуля будет сигнализировать о том, что картинка кончилась и остальные элементы таблицы не содержат полезной информации. Разберем, какая картинка задается таблицей, начало которой приведено ниже.

3.	0.	0.	0.	1.	1.	1.	1.	0.	1.	0.	0.5	1.	0.5	0.
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Первое число в таблице — 3. Это значит, что далее в таблице записана ломаная из трех звеньев, т.е. последовательно заданы 8 чисел — координаты четырех вершин ломаной: (0.,0.), (0.,1.), (1.,1.) и (1.,0.). Следующее число в таблице — 1. Это значит, что далее в таблице записана ломаная из одного звена с координатами вершин (0.,0.5) и (1.,0.5). Следующее число в таблице равно нулю. Это означает, что картинка кончилась и больше полезной информации в таблице нет.

Упражнения:

23. Какое количество отрезков можно гарантированно задать с помощью модели М16?

24. Задайте с помощью модели М16 картинку, изображающую
а) букву М;
б) слово МАМА.

25. Составьте алгоритм перевода информации из модели М16 в модель М15.

§ 9. ЗАДАЧИ НА ПОСТРОЕНИЕ ИНФОРМАЦИОННЫХ МОДЕЛЕЙ

26. Придумайте способ кодировки позиции в игре «крестики-нолики» на доске 10×10 клеток. Как подсчитать количество крестиков на доске? Как узнать, выиграли ли крестики?

27. Составьте информационную модель расписания уроков:
а) одного ученика;
б) одного класса (на некоторых уроках класс может делиться на группы);
в) всей школы.

Как подсчитать число уроков в среду (модели а и б)? Как узнать, сколько раз за неделю класс переходит из одной комнаты (кабинета) в другую (модель б)? Сколько учителей в школе (модель в)? Как определить, сколько уроков в неделю проводит данный учитель (модель в)?

28. На складе хранятся приборы 1000 наименований, не более 100 экземпляров каждого прибора. Приборы размещены в 50 помещениях, вмещающих до 2000 приборов каждое. Составьте информационную модель склада, позволяющую узнать, есть ли на складе данный прибор и в каком помещении его искать.
29. Составьте информационные модели:
    а) свидетельства о рождении;
    б) паспорта;
    в) свидетельства о браке;
    г) аттестата о среднем образовании;
    д) авиационного билета.
30. В камере хранения 100 ячеек. При выдаче багажа взимается плата 3000 р. за каждые полные или неполные сутки хранения. Придумайте информационную модель камеры хранения, позволяющую выяснять, есть ли свободные ячейки, и вычислять плату при выдаче багажа.
31. Предполагается создать информационную систему для предварительной продажи билетов в единственный 36-местный купейный вагон, следующий без остановок по маршруту Москва—Петушки. Задайте информационную модель вагона для такой системы. Составьте алгоритмы поиска
    а) свободного места;
    б) свободного нижнего места;
    в) двух свободных мест в одном купе.
32. Задайте информационную модель поезда из 14 одинаковых 36-местных купейных вагонов. Составьте алгоритмы поиска двух свободных мест в одном вагоне, четырех свободных мест в одном или в двух соседних вагонах.
33. Придумайте информационную модель поезда, в котором первые 8 вагонов — купейные, 9-й — вагон-ресторан, а вагоны с 10-го по 14-й — плацкартные. Пусть ваш класс в полном составе собрался поехать на экскурсию. Сформулируйте требования, которым, по вашему мнению, должны удовлетворять места в поезде для такой поездки (например, «все места в одном вагоне», «не более чем в двух соседних вагонах», «рядом с рестораном» и т.п.). Составьте алгоритм, который определит, есть ли в поезде нужное количество мест, удовлетворяющих вашим требованиям.

Глава 3 КАК УСТРОЕНЫ КОМПЬЮТЕРНЫЕ ПРОГРАММЫ

§ 10. «ЗАПИСНАЯ КНИЖКА». ПРИМЕР БАЗЫ ДАННЫХ

В этом параграфе рассмотрен пример работы с ***базой данных***. Речь идет об электронной телефонной книжке. Подобно обычной телефонной книжке, в нее заносятся имена и фамилии друзей и знакомых, а также их телефоны и дни рождения. Тем не менее на примере электронной записной книжки можно продемонстрировать все существенные черты современных баз данных.

10.1. Что такое база данных?

Прежде всего, база данных — это хранилище информации, из которого многие пользователи могут черпать нужные им сведения. Кроме того, говоря о базе данных, обычно подразумевают, что информация хранится в электронном виде и доступ к ней осуществляется средствами компьютерной обработки. К современным базам данных предъявляются следующие требования:

1. Возможность внесения и чтения информации.
2. Возможность работы с большими объемами данных.
3. Скорость поиска.

Люди, которые работают с информацией, содержащейся в базе данных, вправе рассчитывать на то, что они могут быстро получить интересующие их сведения. В некоторых случаях задержки при исполнении запроса могут сделать бессмысленной работу с базой. Например, брокер на фондовой бирже должен получать информацию о текущих котировках акций в течение нескольких секунд. Иначе ему эта информация не нужна вообще.

4. Целостность данных.

Говоря о целостности данных, обычно имеют в виду, что данные, хранящиеся в разных частях базы данных, не противоречат друг другу. Другими словами, можно задать условия, которым

должны удовлетворять данные. Например, рассмотрим базу данных, которая содержит сведения о всех работниках предприятия, их заработной плате и о том, кто является начальником каждого сотрудника. В этой базе данных может быть задано условие, что зарплата каждого сотрудника не превосходит заработной платы его начальника. Другой пример целостности данных в той же базе: для каждого человека должна быть указана его заработная плата.

5. Защита данных из базы от разрушения.

Здесь имеется в виду, во-первых, нарушение целостности. Во-вторых, потеря необходимой информации, например стирание нужных данных из памяти компьютера. Данные могут разрушиться при сбоях электропитания или в результате ошибки в программе, обрабатывающей эти данные. В самом неприятном случае это может быть полное уничтожение данных. Современные информационные системы несут такой объем ценной информации, что разрушение данных может привести к большим и часто невосполнимым потерям человеческого труда. Поэтому базы данных должны быть защищены от разрушений и снабжены средствами восстановления данных, если разрушение все-таки произошло.

6. Защита данных от несанкционированного доступа.

Современные информационные системы содержат огромное количество информации, и к этой информации одновременно обращаются много людей. Информация, которая хранится в базах данных, используется для принятия важных решений в различных областях человеческой деятельности, в том числе для анализа экономических, социальных тенденций и т.п. Люди, пользующиеся этой информацией, должны быть уверены в ее достоверности. Поэтому информация должна быть надежно защищена от постороннего вмешательства. Круг людей, которые могут вносить изменения в данные, должен быть ограничен небольшим количеством компетентных и заслуживающих доверия людей.

Упражнения:

34. С какой скоростью должен выполняться поиск нужного телефона в электронной записной книжке, чтобы с ней было удобно работать?

35. Придумайте правило целостности для базы данных электронной записной книжки. Является ли высказывание «Каждый человек должен иметь только один телефонный номер» правилом целостности?

10.2. Системы управления базами данных

Ответственность за соблюдение требований, изложенных выше, принимают на себя программные комплексы, называемые ***системами управления базами данных*** (СУБД). СУБД обыкновенно предоставляют пользователям средства доступа к информации, возможность вносить изменения и получать нужные данные.

Место СУБД при работе с базами данных находится между пользователем — потребителем информации и базой данных — хранилищем информации. Другими словами, пользователь работает с СУБД, а СУБД обращается непосредственно к данным:

Рассмотрим подробнее те средства, которые используются в СУБД для реализации ее основных функций — занесения информации в базу данных и извлечения ее оттуда.

Большинство современных систем управления базами данных позволяет представлять данные в виде таблиц. Это название отражает аналогию между обычной таблицей и базой данных. Из каких компонентов состоит обычная таблица? В ней есть строки, каждая строка разбита на колонки, и колонки имеют заголовок. Например, список сотрудников можно представить следующей таблицей:

Фамилия	Имя	Должность
Коробейников	Иван	директор
Верхушкина	Анастасия	бухгалтер

Говоря о таблице базы данных, обычно говорят, что таблица состоит из ***записей***, каждая запись состоит из ***полей***, каждое поле имеет ***название*** и ***значение***. Ниже приведен словарик, показывающий связь этих понятий с обычными таблицами:

Обычная таблица	Таблица базы данных
строка	запись
столбец	поле

Обычная таблица	Таблица базы данных
заголовок столбца	название поля
содержимое столбца в данной строке	значение поля в данной записи

Основные операции, которые выполняют СУБД при работе с таблицами, таковы:

- добавить запись в таблицу;
- удалить запись из таблицы;
- изменить имеющуюся запись;
- найти запись по значениям нескольких полей.

Далее, СУБД позволяют создавать новые таблицы и удалять имеющиеся. Кроме того, многие СУБД позволяют двигаться по таблице вперед и назад, считывая записи по одной в том порядке, в котором они занесены в таблицу.

Здесь приведены только самые общие сведения об устройстве и принципах работы систем управления базами данных. Реальные системы обладают большим количеством дополнительных возможностей и функций, которые позволяют эффективно работать с базами данных.

Упражнение:

36. Придумайте таблицу для хранения телефонов из записной книжки. Как называются поля этой таблицы?

10.3. Исполнитель DB — учебная система управления базами данных

Основные команды, нужные для взаимодействия с базой данных, реализованы в исполнителе DB системы КуМир. Исполнитель DB работает с таблицами, у которых значениями полей могут быть текстовые строки длиной до 256 символов. Такие поля мы будем называть ***литерными полями***.

Система КуМир вместе с исполнителем DB представляет простейшую систему управления базами данных. Несмотря на свою простоту, она предоставляет все необходимые возможности для написания программ, которые занимаются обслуживанием базы данных.

Для того чтобы начать работу с таблицей, нужно выполнить команду

начать работу (арг лит F, арг цел N) — начинает работу с таблицей с именем F, в которой N литерных полей.

После выполнения этой команды все следующие команды исполнителя DB будут относиться к этой таблице до тех пор, пока не будет исполнена команда начать работу для другой таблицы.

Команда начать работу может также использоваться для создания таблицы: если при выполнении этой команды таблицы с указанным именем не существовало, то она автоматически создается.

В каждый момент времени одна из записей таблицы является текущей. Это означает, что команды редактирования и удаления относятся именно к этой записи. Текущая запись остается неопределенной до выполнения первой команды поиска или добавления записи.

Для добавления новой записи используется команда

добавить (арг лит таб A[1:N]) — добавляет в таблицу запись со значениями полей из A.

Следующие три функции предназначены для поиска записей в таблице. Они возвращают логическое значение да, если поиск был успешен, и нет — в противном случае. При успешном поиске найденная запись становится текущей, при неуспешном — текущая запись не определена.

лог начать поиск (арг литтаб A[1:N], рез литтаб B[1:N]) — найти первую запись с полями, значения которых заданы в литерном массиве A (пустая строка обозначает, что значение может быть любым). Если поиск успешен, то значения полей найденной записи помещаются в литерный массив B.

лог следующий (рез литтаб B[1:N]) — найти следующую запись с полями, значения которых заданы в предыдущей команде начать поиск. Если поиск успешен, то значения полей найденной записи помещаются в B.

лог предыдущий (рез литтаб B[1:N]) — найти предыдущую запись с полями, значения которых заданы в команде начать поиск. Если поиск успешен, то значения полей найденной записи помещаются в B.

Для удаления и изменения текущей записи предназначены команды:

удалить — удалить текущую запись;

редактировать (арг литтаб A[1:N]) — заменить текущую запись записью с полями, значения которых находятся в литерном массиве A (пустая строка обозначает, что значение поля должно остаться без изменения).

10.4. База данных «Записная книжка»

База данных «Записная книжка» хранит данные об именах, телефонах и днях рождения различных людей. Эта база данных состоит из одной таблицы с именем NOTEBOOK.DB, которая имеет пять полей с названиями "Фамилия", "Имя", "Отчество", "Дата рождения" и "Телефон".

Мы рассмотрим программу, которая обслуживает эту базу данных. Эта программа выполняется следующим образом. Сначала на экран выводится меню:

```
(1 - Добавить, 2 - Найти, 3 - Выйти) ?> _
```

Если выбрать "Добавить" (т.е. нажать 1), то программа спрашивает данные о человеке. В ответ нужно ввести по очереди данные о фамилии, имени, отчестве, телефоне и дне рождения. После этого автоматически добавится новая запись с этими данными.

Если выбрать "Найти" (нажать клавишу 2), то программа спрашивает данные о человеке и выполняет поиск. После выбора "Выйти" работа программы завершается.

После успешного поиска на экране появляется запись, удовлетворяющая указанным требованиям, и следующее меню:

```
1 - следующий     4 - редактировать
2 - предыдущий    5 - закончить
3 - удалить       ?>
```

По команде "Следующая запись" появляется следующая запись, удовлетворяющая заданным условиям, или главное меню, если таковая отсутствует.

По команде "Предыдущая запись" появляется предыдущая запись, удовлетворяющая заданным условиям, или главное меню, если таковая отсутствует.

По команде "Удалить" текущая запись удаляется.

По команде "Редактировать" вас просят заново ввести поля записи. Если вы вводите пустую строку (т.е. просто нажимаете на клавишу Enter), ничего не введя), это означает, что данное поле записи останется неизменным.

По команде "Завершить" вы возвращаетесь к главному меню.

Вот полный текст программы в КуМире:

```
лит таб A[1:5]

алг Записная книжка
нач цел к, лог усп, лит таб B[1:5], C[1:5]
| A[1] := 'Фамилия'
| A[2] := 'Имя'
| A[3] := 'Отчество'
| A[4] := 'Дата рождения'
| A[5] := 'Телефон'
| начать работу ("NOTEBOOK.DB",5); к := 0
| нц пока к <> 3
|  вывод '(1 - Добавить, 2 - Найти, 3 - Выйти) ?>'
|  ввод к; вывод нс
|  выбор
|   при к = 1: ввод_данных(B); добавить(B)
|   при к = 2: ввод_данных(B)
|    усп := начать поиск(B,C); к := 0
|    нц пока усп и к <> 5
|     вывод_данных(C)
|     вывод '1 - следующий   4 - редактировать',нс
|     вывод '2 - предыдущий  5 - закончить',нс
|     вывод '3 - удалить     ?>'
|     ввод к; вывод нс
|     выбор
|      при к = 1: усп := следующий(C)
|      при к = 2: усп := предыдущий(C)
|      при к = 3: удалить; к := 5
|      при к = 4: ввод_данных(B)
|       редактировать(B); к := 5
|     все
|    кц
|  все
| кц
кон

алг ввод_данных (рез лит таб B[1:5])
нач цел i
| нц для i от 1 до 5
|  вывод A[i],':'; ввод B[i]; вывод нс
| кц
| вывод '- - - - - - - - - - - - - - - - - - - - -',нс
кон

алг вывод_данных (арг лит таб C[1:5])
нач цел i
| нц для i от 1 до 5
|  вывод A[i],':',C[i],нс
| кц
| вывод '- - - - - - - - - - - - - - - - - - - - -',нс
кон
```

Упражнение:

37. Написать программу, которая позволяет просматривать данные о всех людях, у которых сегодня день рождения.

§ 11. РЕАЛИЗАЦИЯ ОЧЕНЬ ПРОСТОГО ГРАФИЧЕСКОГО РЕДАКТОРА

11.1. Зачем нужен графический редактор?

Машинная графика — построение изображений и манипулирование ими при помощи компьютеров — очень важна для многих приложений. Созданные компьютером изображения и мультипликация используются при создании кинофильмов и телешоу (вспомните «Терминатор-2» или «Газонокосильщик»), рекламы и видеоигр. Так как машинная графика не ограничена физическими законами, то созданные компьютером объекты могут свободно менять цвета, трансформироваться, превращаться в другие объекты, двигаться с невообразимой скоростью.

Машинная графика используется при конструировании объектов всех видов — от зданий и самолетов до одежды и игрушек. Конструкция может быть изображена на экране компьютера, проверена, подвергнута различным испытаниям, моделируемым на компьютере.

Другим применением машинных графических средств является построение графиков и диаграмм для финансовых, статистических отчетов и тому подобного. Компьютеры можно запрограммировать так, чтобы при вводе новых данных менялось их графическое представление.

«Я думаю, что чертежи — очень полезное средство против неопределенности слов».

(Лейбниц, 1646—1716 гг.)

Графика играет важную роль в обучающих программах: при изучении геометрии (например, для иллюстрации понятия параллельного переноса), физики (для демонстрации действий над векторами), географии (для развития навыков работы с картой). Более сложные программы, которые обеспечивают получение графических изображений, близких к реальности, используются, например, для тренировки пилотов совершать посадку в различных условиях.

Вообще, везде, где появляется изображение (будь то телевидение, информация со спутников Земли или с медицинского зонда),

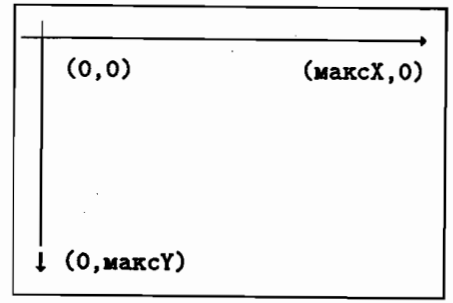


Рис. 3. Координаты на экране

оно может быть представлено в цифровой форме. А значит, и обработано компьютером.

Очень важная современная сфера применения компьютеров — издательская деятельность. С помощью ***издательских систем*** (компьютерных комплексов, включающих в себя возможности по обработке текстов и графики) газеты, журналы, книги верстаются в окончательном виде прямо на компьютерах, и оттуда напрямую попадают на печатные станки.

В этом параграфе обсуждается задача реализации простейшего ***графического редактора*** — программы для построения и манипулирования изображением на компьютере.

Запустите гипертекст и посмотрите, как работает графический редактор.

11.2. Исполнитель ГраТекс

Для реализации графического редактора потребуются команды рисования на экране. Вообще говоря, можно пользоваться единственной базовой командой *нарисовать точку* (в самом деле, любое изображение на экране компьютера состоит из отдельных точек разных цветов). Однако проще использовать исполнитель КуМира ***ГраТекс***.

При работе с ГраТексом на экране компьютера вводятся целочисленные координаты (рис. 3), где `максX` и `максY` — константы ГраТекса, определяющие размер экрана в пикселах соответственно по горизонтали и вертикали.

В ГраТексе важную роль играет понятие ***текущей позиции*** — аналог положения курсора в текстовых редакторах. Вывод ***графических примитивов*** ведется с использованием текущей пози-

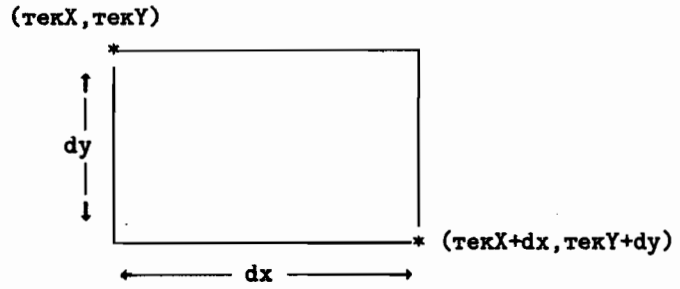


Рис. 4. Рисование прямоугольников в ГраТексе

ции. Графические примитивы — базовые графические объекты, которые «умеет» изображать сам ГраТекс. К таким примитивам относятся точка, прямая, прямоугольник, круг, эллипс.

Мы будем использовать в графическом редакторе *монохромный* видеорежим, в котором рисование выполняется одним цветом. В этом режиме у точки на экране может быть всего два цвета: цвет изображения (белый) и цвет фона (черный).

Команды исполнителя ГраТекс

видео(арг цел r) — установить графический режим с номером r; для выбора стандартного графического режима надо указать r = –1.

цел текX, текY — координаты текущей позиции.

поз(арг цел x,y) — сместить текущую позицию в точку (x,y).

цвет(арг цел c) — установить цвет рисования; значение аргумента c = 0 означает черный цвет, а c = 1 — белый.

точка(арг цел x,y) — изобразить точку (x,y).

линия(арг цел x,y) — нарисовать отрезок из текущей позиции в точку с координатами (x,y).

вектор(арг цел dx,dy) — нарисовать отрезок из текущей позиции в точку с координатами (текX+dx,текY+dy).

контур(арг цел dx,dy) — нарисовать прямоугольник с левым верхним углом в текущей позиции и длинами сторон dx и dy; правый нижний угол будет в точке с координатами (текX+dx,текY+dy) — см. рис. 4.

блок(арг цел dx,dy) — нарисовать закрашенный прямоугольник длинами сторон dx и dy из текущей позиции.

эллипс(арг цел a,b) — нарисовать эллипс с центром в текущей позиции и полуосями a и b.

овал(арг цел a,b) — нарисовать закрашенный эллипс с центром в текущей позиции и полуосями a и b.

окружность(арг цел r) — нарисовать окружность с центром в текущей позиции и радиусом r.

круг(арг цел r) — нарисовать закрашенную окружность с центром в текущей позиции и радиусом r.

закраска — закрасить замкнутую фигуру (текущая позиция должна находиться внутри этой фигуры).

Логические операции при рисовании

Рисование всегда ведется с использованием ***текущей логической операции***. Для ее установки используется переменная ГраТекса сим лог_оп, которая может принимать одно из следующих четырех значений:

```
лог_оп:='i'   – прямая запись
лог_оп:='&'   – логическое И
лог_оп:='|'   – логическое ИЛИ
лог_оп:='^'   – исключающее ИЛИ
```

Для объяснения сущности логической операции легче всего воспользоваться таблицей, показывающей, как изменится состояние точки при рисовании с использованием той или иной операции:

Старый цвет	Цвет, которым рисуем	Цвет после рисования с операцией			
		i	&	|	^
черный	черный	черный	черный	черный	черный
белый	черный	черный	черный	белый	белый
черный	белый	белый	черный	белый	белый
белый	белый	белый	белый	белый	черный

Из таблицы видно, что при рисовании белым цветом с использованием логической операции «исключающее ИЛИ» цвет точек на экране изменяется на противоположный. Поэтому двойное рисование в этом режиме оставляет изображение неизменным (черный → белый → черный или белый → черный → белый). Именно на этом факте и будет основана работа исполнителя Курсор (см. пункт 11.4).

11.3. Исполнитель Курсор — описание

В графическом редакторе совершенно необходим *курсор*, отмечающий на экране, какая позиция является текущей (то есть в какой позиции экрана будет производиться рисование). Важная деталь курсора — он, хотя и должен изображаться на экране, сам не является частью рисуемой картинки. Курсор можно представлять как какой-то шаблон, двигающийся *над* изображением.

Команды исполнителя Курсор:

курсор_начать (арг цел x,y) — начать работу с исполнителем Курсор и задать начальное положение курсора в точке с координатами (x,y).

курсор_тип (арг лит a) — задать тип курсора. В графическом редакторе будут использоваться два типа курсора: "обычн" (обыкновенный курсор в виде крестика) и "линия" (курсор, используемый при рисовании линий).

курсор_поз (арг цел x,y) — переместить курсор в позицию (x,y).

курсор_спрятать — сделать курсор невидимым;

курсор_показать — показать спрятанный курсор.

11.4. Исполнитель Курсор — реализация

В КуМире нет готового исполнителя Курсор, поэтому придется его написать.

Реализацию исполнителя будем вести «сверху вниз», то есть вначале составим алгоритмы, реализующие команды исполнителя, а потом — вспомогательные алгоритмы, которые понадобятся при работе.

Глобальные переменные:

```
цел курсX,курсY  – текущее положение
цел тип          – и тип курсора
```

Команды исполнителя:

```
алг курсор_начать (арг цел x,y)
нач
| курсX:=x  | запомним текущую позицию
| курсX:=y
| _изо(x,y) | изобразим курсор в точке (x,y)
кон
```

Используемый здесь алгоритм _изо изображает курсор в указанной позиции. Он будет реализован позже.


```
алг курсор_тип (арг лит a)
нач
| _удалить(курсX,курсY)  | удалим курсор
| выбор
| | при a="обычн": тип:=0 | выберем новый
| | при a="линия": тип:=1 | тип курсора
| все
| _изо(курсX,курсY)      | выведем курсор
кон                      | нового типа
```

Алгоритм _удалить удаляет курсор, находящийся в указанной позиции. При этом изображение на экране должно стать таким же, как было до использования _изо.

```
алг курсор_поз (арг цел x,y)
нач
| _удалить(курсX,курсY)  | удалим старый
| _изо(x,y)              | выведем «новый»
| курсX:=x
| курсY:=y               | сделаем текущей «новую» позицию
кон
алг курсор_спрятать
нач
| _удалить(курсX,курсY);
кон
алг курсор_показать
нач
| _изо(курсX,курсY)
кон
```

Реализация вспомогательных алгоритмов

Алгоритм рисования курсора должен запомнить старое значение логической операции. Потом, если курсор «обыкновенный», надо установить логическую операцию «исключающее ИЛИ» и нарисовать «крестик» размером 6 × 6 с центром в (x,y). В конце надо восстановить значение логической операции:

```
алг _изо (арг цел x,y)
нач сим s
| s:=лог_оп | сохраним старое значение
| если тип=0
| то лог_оп:='^'
|   поз(x,y-3);вектор(0,6)
```

```
    | поз(x-3,y);вектор(6,0)
  все
  лог_оп:=s
кон
```

Для удаления курсора надо всего лишь нарисовать курсор в этом месте еще раз. При этом старое состояние, бывшее до изображения курсора, полностью восстановится:

```
алг _удалить (арг цел x,y)
нач
| _изо(x,y)
кон
```

Запустите гипертекст и посмотрите, как работает исполнитель Курсор.

11.5. Ввод символов с клавиатуры

В предыдущем разделе был реализован исполнитель Курсор. Теперь все требуемые исполнители готовы и можно переходить к написанию самого редактора.

Программу вначале запишем на некотором «псевдоязыке», а затем, внося дополнения и исправления, получим работающий вариант программы. Первое приближение будет выглядеть так:

```
алг Графический Редактор
нач
| Описание переменных
| Инициализация переменных и окружения
| нц
| | Ввести символ
| | Выполнить операцию
| | Передвинуть курсор
| кц_при конец работы
кон
```

Для ввода символа с клавиатуры в КуМире используется команда a:=getkey(). В результате ее выполнения литерной величине a будет присвоено значение, равное символическому обозначению нажатой клавиши или аккорда.

Так как графический редактор должен выполнять несколько различных операций, надо воспользоваться командой выбор. Таким образом, в качестве второго приближения получается следующий алгоритм:


```
алг Графический Редактор
нач
| | Описание переменных
| лит a
| Инициализация переменных и окружения
| нц
| | a:=getkey() | Ввести символ
| | выбор
| | | при a="...": ...
| | | ...
| | все
| | Передвинуть курсор
| кц_при a="Esc ↑" | условие выхода из цикла
кон
```

«Каркас» программы готов. Теперь наша задача состоит в последовательном дописывании частей, реализующих те или иные функции редактора. Начать следует, несомненно, с команд перемещения курсора.

11.6. Изменение текущего положения

Для хранения текущего положения курсора введем глобальные переменные цел x,y.

Перемещение курсора осуществляется следующим образом. Сначала по введенному символу в алгоритме выч_смещ вычисляется смещение dx и dy. В этом алгоритме также проверяется, не выйдет ли курсор за границу экрана (если выйдет, то смещение делается нулевым). Затем изменяется текущая позиция и в нее перемещается курсор.

```
цел x,y

алг Графический Редактор
нач
| | Описание переменных
| лит a
| цел dx,dy
| | Инициализация переменных и окружения
| x:=0 | начальное положение - левый
| y:=0 | верхний угол экрана
| видео(-1)           | установим видеорежим
| курсор_начать(x,y) | начнем работу с Курсором
| курсор_тип("обыкн")
```

```
  нц
    a:=getkey()          | Ввести символ
    выч_смещ(a,dx,dy)    | Вычислить смещение
    выбор
      при a="...": ...
        ...
    все
    | Передвинуть курсор
    x:=x+dx;y:=y+dy | изменим текущую позицию
    курсор_поз(x,y) | передвинем курсор
  кц_при a="Esc ↑"  | условие выхода из цикла
кон
алг выч_смещ (арг лит a,рез цел dx,dy)
нач
  dx:=0 | это значение будет возвращено, если
  dy:=0 | мы не можем сместить курсор (например,
        | в случае выхода за границу экрана)
  выбор
    при a="↑": если y-1>=0     то dy:=-1 все
    при a="↓": если y+1<=макcY то dy:= 1 все
    при a="→": если x+1<=макcX то dx:= 1 все
    при a="←": если x-1>=0     то dx:=-1 все
    при a="Shift+↑": если y-5>=0     то dy:=-5 все
    при a="Shift+↓": если y+5<=макcY то dy:= 5 все
    при a="Shift+→": если x-5<=макcX то dx:= 5 все
    при a="Shift+←": если x+5>=0     то dx:=-5 все
  все
кон
```

Упражнение:

38. Опишите, что будет происходить при нажатии на клавиши Shift стрелка.

11.7. Научим компьютер рисовать

Пока наш графический редактор еще не умеет рисовать. Начнем с самого простого: с режима, когда курсор, перемещаясь по экрану, оставляет за собой след.

Переключение между режимом рисования и перемещения по экрану без следа будет происходить по нажатии клавиши пробел.

Рисовать будем так: если редактор находится в режиме рисования, то после вычисления смещения dx, dy изображается отрезок из текущего положения (x,y) в новое (x+dx,y+dy). Затем курсор перемещается.

Для того чтобы различать внутри программы режимы перемещения и рисования, введена переменная цел лин:

лин=0 — перемещение без следа,

лин=1 — перемещение со следом.

В начале работы программы мы находимся в режиме перемещения без следа, т.е. лин=0.

```
цел x,y; цел лин

алг Графический Редактор
нач
  | Описание переменных
  лит a
  цел dx,dy
  | Инициализация переменных и окружения
  x:=0 | начальное положение - левый
  y:=0 | верхний угол экрана
  лин:=0
  видео(-1)           | установим видеорежим
  курсор_начать(x,y) | начнем работу с Курсором
  курсор_тип("обычн")
  нц
    a:=getkey()         | Ввести символ
    выч_смещ(a,dx,dy) | Вычислить смещение
    выбор
      при a=" ":  | если нажат «пробел»
       лин:=1-лин | то изменим режим
       выбор      |    переключим тип курсора
         при лин=0: курсор_тип("обычн")
         при лин=1: курсор_тип("линия")
       все
      при a="...": ...
    все
    | Если режим рисования, то нарисуем линию
    если лин=1 то поз(x,y);вектор(dx,dy) все
    | Передвинуть курсор
    x:=x+dx;y:=y+dy | изменим текущую позицию
    курсор_поз(x,y) | передвинем курсор
  кц_при a="Esc ↑" | условие выхода из цикла
кон
```

Запустите гипертекст и посмотрите, как работает полученная программа.

11.8. Зачем на компьютере нужна «резиновая нить»?

Поработав с построенным графическим редактором, вы, наверное, заметили, что рисовать с его помощью можно, но не совсем удобно. Например, если провести горизонтальную или вертикальную прямую достаточно просто, то уже с наклонными линиями возникают сложности. Хотелось бы, чтобы сам редактор умел рисовать линии по заданным двум точкам.

Возможен такой вариант работы редактора: с помощью курсора отмечается одна точка, затем вторая, и после всего этого рисуется отрезок с указанными концами.

Это, конечно, удобнее, чем рисовать курсором по точкам, но и здесь есть свой недостаток: линию мы увидим только после того, как нарисуем ее, и если мы чуть-чуть промахнулись, то «подвинуть» ее уже будет невозможно.

Оптимальным было бы решение, если бы мы могли изменять положение линии на экране и, удостоверившись, что она поставлена на правильное место, оставить ее там.

Такой подход реализуется с помощью интерфейса ***«резиновой нити»***. Представьте себе, что у вас есть кусочек резиновой нити. Один конец вы закрепляете кнопкой, а другой начинаете перемещать. При этом нить растягивается, образуя отрезок между закрепленным концом и концом, который у вас в руке.

Реализуем этот метод на компьютере. Пусть «резиновая нить» вызывается нажатием F1, которое «закрепляет» один конец нити в текущей позиции курсора. Далее перемещение курсора «натягивает» нить в текущую позицию курсора. При этом в каждый момент времени на экране изображен отрезок от закрепленной точки до курсора. Потом можно нажать Enter, «закрепив» линию в текущем положении, или Esc для отмены рисования.

```
алг рез_линия
нач лит a, сим s
| цел x0,y0 | позиция «закрепленного» конца
| цел dx,dy | смещение
|
| x0:=x | «закрепим» конец в положении курсора
| y0:=y
| s:=лог_оп   | установим логическую операцию
| лог_оп:='^' |   исключающее ИЛИ
| курсор_спрятать
| поз(x0,y0) | нарисуем линию из начала
| линия(x,y) | в текущее положение
| курсор_показать
| нц
|  a:=getkey          | ввод символа
|  выч_смещ(a,dx,dy) | вычислим смещение
|  поз(x0,y0)         | сотрем «старую» линию
|  линия(x,y)
|  x:=x+dx;y:=y+dy | изменим текущую позицию
|  курсор_поз(x,y) | переместим курсор
|  поз(x0,y0)       | нарисуем линию из «закреплен-
|  линия(x,y)       | ной» точки в текущую позицию
| кц_при a="Enter" или a="Esc"
|
| курсор_спрятать
| поз(x0,y0)  | удалим нарисованную линию
| линия(x,y)
| | если мы решили нарисовать линию,
| | то установим режим прямой записи
| |    и нарисуем линию
| если a="Enter"
|  то лог_оп:='i'
|  поз(x0,y0);линия(x,y)
| все
| лог_оп:=s; курсор_показать
кон
```


В алгоритм Графический Редактор надо добавить вызов алгоритма рез_линия при нажатии на клавишу F1.

```
| выбор
|  при a=" ":
|  ...
|  при a="F1":
|   лин:=0; курсор_тип("обычн")
|   рез_линия
|  ...
| все
```

Запустите гипертекст и посмотрите, как работает «резиновая нить».

11.9. Рисование прямоугольников

В предыдущем пункте описано рисование линий при помощи «резиновой нити». Попробуем с помощью похожего принципа нарисовать прямоугольник.

Аналогично с «резиновой нитью» закрепим текущее положение по нажатии F2, затем будем перемещать курсор и ожидать нажатия Enter или Esc. В отличие от «резиновой нити», прямоугольник можно рисовать двояко: полным (закрашенным) или пустым (контурным). Для этого в алгоритме рисования прямоугольника введем логический аргумент закрашивать (да или нет).

```
алг рез_прям (арг лог закрашивать)
нач лит a, сим s
| цел x0,y0 | позиция «закрепленного» конца
| цел dx,dy | смещение
| x0:=x; y0:=y
| s:=лог_оп; лог_оп:='^'
| курсор_спрятать
| поз(x0,y0)   | нарисуем линию из начала
| контур(0,0) | нулевого размера
| курсор_показать
| нц
| | a:=getkey; выч_смещ(a,dx,dy)
| | поз(x0,y0)          | сотрем «старый»
| | контур(x-x0,y-y0) |   прямоугольник
| | x:=x+dx;y:=y+dy    | изменим текущую позицию
| | курсор_поз(x,y)    | переместим курсор
| | поз(x0,y0)          | нарисуем «новый»
| | контур(x-x0,y-y0) |   прямоугольник
| кц_при a="Enter" или a="Esc"
|
| курсор_спрятать
| поз(x0,y0)   | удалим нарисованный
| линия(x,y)   | прямоугольник
|
| | если мы решили нарисовать прямоугольник,
| | то установим режим прямой записи
|   и нарисуем прямоугольник или контур
| если a="Enter"
| | то лог_оп:='i'
| | поз(x0,y0)
| | если закрашивать
| | | то    блок  (x-x0,y-y0)
| | | иначе контур(x-x0,y-y0)
| | все
| все
| лог_оп:=s; курсор_показать
кон
```

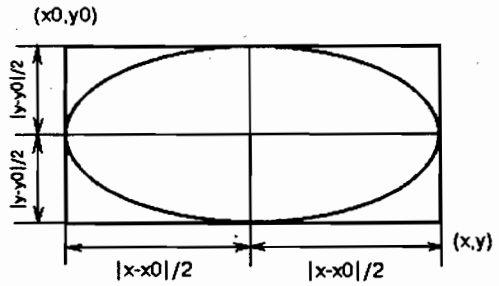



Рис. 5.

В главный алгоритм Графический Редактор надо внести лишь небольшое дополнение:

```
| выбор
| | при a="F1":
| | ...
| | при a="F2":
| |  лин:=0; курсор_тип("обыкн")
| |  рез_прям(нет) | контур
| | при a="Ctrl+F2":
| |  лин:=0; курсор_тип("обыкн")
| |  рез_прям(да)  | закрашенный
| все
```

11.10. Эллипсы и окружности

Попробуем нарисовать простейшую картинку: домик, горизонт, солнце... Стоп. Наш редактор еще не умеет рисовать окружности. И если прямоугольник можно было нарисовать с помощью четырех отрезков, то окружность таким методом изобразить невозможно. Добавим в наш редактор алгоритмы рисования окружностей и эллипсов. Это будет несложно, так как ГраТекс сам «умеет» рисовать и окружности, и эллипсы — нам остается только написать вызовы соответствующих алгоритмов в графический редактор.

Начнем с реализации алгоритма рисования эллипса. Аналогично прямоугольникам ГраТекс умеет рисовать как контурный эллипс, так и закрашенный (овал). Эллипс будем строить с помощью описанного вокруг него прямоугольника. Предназначение прямоугольника — задать узловые точки фигуры (рис. 5).

Алгоритм «резинового овала» очень похож на реализованный ранее алгоритм «резинового прямоугольника». Единственное различие состоит в том, что если для рисования прямоугольника Гра-

Тексу было необходимо задание вершины (текущей позицией) и длин сторон, то для рисования эллипса или овала необходимо задание центра и полуосей.

Обратимся к рис. 5. Пусть (x_0, y_0) — закрепленная точка, (x, y) — текущая позиция курсора, тогда центр эллипса будет $((x + x_0)/2, (y + y_0)/2)$, а его полуоси $|x - x_0|/2$ и $|y - y_0|/2$. Таким образом, рисование эллипса, вписанного в прямоугольник с вершинами (x0,y0) и (x,y), выглядит следующим образом:

```
поз(цел((x+x0)/2),цел((y+y0)/2))
эллипс(цел(abs(x-x0)/2),цел(abs(y-y0)/2))
```

или

```
поз(цел((x+x0)/2),цел((y+y0)/2))
овал(цел(abs(x-x0)/2),цел(abs(y-y0)/2))
```

Как же изменится программа при добавлении алгоритма «резинового овала»?

1. В обработку введенного символа добавится фрагмент:

```
выбор
| при a="Ctrl+F2":
| ...
| при a="F3":
|  лин:=0; курсор_тип("обычн")
|  рез_овал(нет) | контур
| при a="Ctrl+F3":
|  лин:=0; курсор_тип("обычн")
|  рез_овал(да)  | закрашенный
| ...
все
```

2. В программу добавится алгоритм

```
алг рез_овал (арг лог закрашивать)
нач
| ...
кон
```

Рисовать окружности таким же методом невозможно, так как не во всякий прямоугольник можно вписать окружность.

Окружность будем рисовать так: закрепленная точка (x_0, y_0) станет центром окружности, а сама окружность будет проходить через текущую позицию курсора; то есть радиус R окружности равен расстоянию от закрепленной точки до текущей позиции:

$$R = \sqrt{(x - x_0)^2 + (y - y_0)^2}.$$

Таким образом, для рисования окружности надо написать команды:

```
поз(x0,y0)
окружность(цел(sqrt((x-x0)**2+(y-y0)**2)))
```

а для рисования круга (закрашенной окружности) — команды:

```
поз(x0,y0)
круг(цел(sqrt((x-x0)**2+(y-y0)**2)))
```

Аналогично рисованию овала изменения в редакторе будут следующими:

1. В обработку введенного символа добавится фрагмент:

```
выбор
| ...
| при a="F4":
|  лин:=0; курсор_тип("обычн")
|  рез_круг(нет) | контур
| при a="Ctrl+F4":
|  лин:=0; курсор_тип("обычн")
|  рез_круг(да)  | закрашенный
| ...
все
```

2. В программу добавится алгоритм

```
алг рез_круг (арг лог закрашивать)
нач
| ...
кон
```

Упражнение:

39. Напишите алгоритмы рез_овал и рез_круг по образцу алгоритма рез_прям.

Запустите гипертекст и посмотрите, как работает получившийся графический редактор.

§ 12. ИНФОРМАЦИОННАЯ МОДЕЛЬ В ИГРЕ «МУДРЫЙ КРОТ»

В данном параграфе речь пойдет о компьютерной игре-головоломке «Мудрый Крот». В мире подземных лабиринтов живет Крот. Он постоянно трудится, перетаскивает съестные припасы, помещенные в громоздкие ящики. Ящики столь велики, что занимают весь коридор, и Крот может только толкать ящики перед собой. Ящики столь тяжелы, что Крот не может передвигать сразу больше одного ящика.

Кроту требуется переместить все ящики на склад. Надо быть очень внимательным, чтобы не задвинуть ящик в тупик. Ведь вытащить ящик из тупика Крот уже никогда не сможет, и игру придется начинать заново.

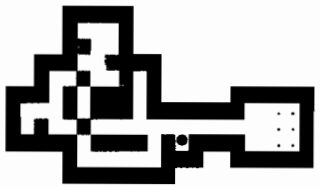


Рис. 6. Игровое поле Мудрого Крота

Запустите гипертекст и помогите Кроту перетащить все ящики в первом лабиринте.

12.1. Представление игрового поля в компьютере

Программа из гипертекста моделирует поведение Крота в подземном лабиринте. Попробуем разобраться, как она устроена.

Первый вопрос, который надо рассмотреть, — это как закодировать лабиринт на компьютере. Прежде всего, необходимо знать размеры лабиринта. Рассмотрим первый лабиринт (рис. 6). Его размеры: ширина лабиринта — 22 клетки, высота — 11 клеток.

Закодируем лабиринт прямоугольной таблицей цел таб поле [1:22,1:11]. Каждое поле лабиринта может быть отнесено к одному из следующих пяти типов:

_пусто		– пустая клетка (проход)
_стена	■	– клетка, занятая стеной
_ящик	■	– клетка с ящиком
_склад	.	– пустая позиция на складе
_ящик_на_складе	■	– ящик, размещенный на складе

Этих данных достаточно, чтобы изобразить на экране компьютера любое промежуточное состояние лабиринта. Остается проверить, насколько просто в такой модели делать ход.

Упражнения:

40. Чего еще не хватает в информационной модели игры?

41. При каком условии лабиринт считается пройденным?

12.2. Можно ли сделать шаг вправо?

Крот должен в каждый момент уметь определять, можно ли переместиться на шаг — ведь недаром же он Мудрый! Определим, при каких условиях он может двигаться. Это возможно тогда, когда в направлении его движения — свободное поле (ведь наш Крот не умеет прорывать новые ходы). Если Крот двигает ящик, то позади ящика должно быть свободное поле.

Приведем фрагмент алгоритма Мудрый Крот, отвечающий за его перемещения.

```
алг пойти(арг цел dx,dy)
нач цел впереди,через,i
  впереди:=поле[x+dx,y+dy]
  если там не пусто(впереди)
  то
    если это ящик(впереди)
    то через:=поле[x+dx+dx,y+dy+dy]
      если там пусто(через)
      то сдвинуть(x+dx,y+dy,x+dx+dx,y+dy+dy)
      иначе
        выход | сдвинуть ящик нельзя
      все
    иначе
      выход | сделать шаг нельзя
    все
  все
  экран(x,y,поле[x,y])
  x:=x+dx;y:=y+dy
  экран(x,y,1)
кон
```

Сначала Крот проверяет, пусто ли поле, на которое он хочет переместиться (вызывая вспомогательный алгоритм там пусто). Если там оказывается стена, то шаг сделать нельзя. Если ящик, то шаг можно сделать, только если позади ящика свободное место.

12.3*. Упакованное хранение лабиринтов

Хранить лабиринт для игры «Мудрый Крот» можно в явном виде: сначала записываются размеры лабиринта, затем следует описание каждого поля в очередном ряду. При этом каждому типу соответствует некоторое число, например, пустому месту — 0, стене — 1 и т.д. Кроме того, необходимо указать начальное положение Крота в лабиринте.

Однако, имея в виду, что в компьютерной реализации игры очень много лабиринтов, применение этого метода может привести к нехватке места. Чтобы избежать этого, информацию о лабиринтах упаковывают.

Надо помнить, что упакованное хранение данных (в частности, лабиринтов) хотя и экономит объем, но оборачивается неизбежной потерей времени, требуемого на распаковку данных. Поэтому следует выбирать, что в каждом конкретном случае важнее сэкономить — объем или время.

Глава 4. КОМПЬЮТЕРЫ В ОБУЧЕНИИ

§ 13. СИСТЕМА ПЛАНИМИР

Один из самых наглядных способов представления геометрической информации — это чертеж. Чертежи можно рисовать на глиняных пластинах, на песке морского пляжа, на классной доске или на обычной бумаге. В XX в. построение чертежей, как и любая другая работа с информацией, все чаще и чаще проводится на компьютере.

Каждый из перечисленных способов рисования чертежей имеет свои достоинства и недостатки. Чертежи на классной доске или на песке пляжа недолговечны, глиняные пластинки долговечны, но тяжелы и неудобны, а на листе бумаги исправить чертеж труднее, чем на классной доске. К недостаткам компьютера можно отнести его дороговизну.

Однако достоинства выполнения чертежа на ЭВМ с лихвой окупают недостатки: такие чертежи легко тиражировать, пересылать, хранить. Но главное, ни один другой способ работы не дает такой легкости при изменении чертежа. На компьютере чертеж можно модифицировать уже после его построения, да еще так, чтобы изменение одного объекта приводило к изменению других, им порожденных. Так, например, построив медианы треугольника, можно перемещать вершины и наблюдать за поведением медиан. Такого рода эксперименты с чертежами позволяют угадывать многие геометрические теоремы (например, теорему о том, что медианы пересекаются в одной точке).

Система ***ПланиМир*** создавалась как инструмент для построения легко модифицируемых, «живых» чертежей. Система может применяться в школьном курсе планиметрии. Особенно удобно применять ее для решения задач на построение. ПланиМир дает возможность заниматься геометрией, не отвлекаясь на разные второстепенные детали (прямая линия оказалась кривой или чертеж не поместился на листе). Возможность перемещать построенные точки и другие объекты и наблюдать за изменением чертежа повышает наглядность построений и облегчает отыскание ошибок.

13.1. Начинаем работу

Для запуска системы надо набрать:

```
C> PLANIMIR
```

На экране появится головная страница гипертекста. Выберите поле Демонстрация и нажмите Enter. Вы увидите изображение нескольких фигур, построение которых доступно в системе (см. рис. 7). Войдите в поле По шагам и нажимайте клавишу пробел. Теперь эти фигуры появляются постепенно, одна за другой. Процесс построения оказывается разделенным на части, каждая из которых проводится после одного нажатия. Этот метод демонстрации построений будет часто использоваться в дальнейшем.

Чтобы научиться работать с системой, прежде всего изучим вход в систему и выход из нее. Итак, нажмите Esc, и вы увидите уже знакомый вам текст. Войдите в поле Свободная работа в Планимире. Перед вами пустое рабочее поле (плоскость чертежа) и главное меню (рис. 8). Строить чертежи мы научимся позже, а пока разберемся, как вернуться назад.

В общем случае это можно сделать двумя способами: сохранив сделанные построения или потеряв их. (Сейчас выбор способа безразличен, так как пока ничего не построено.) Как и в любой

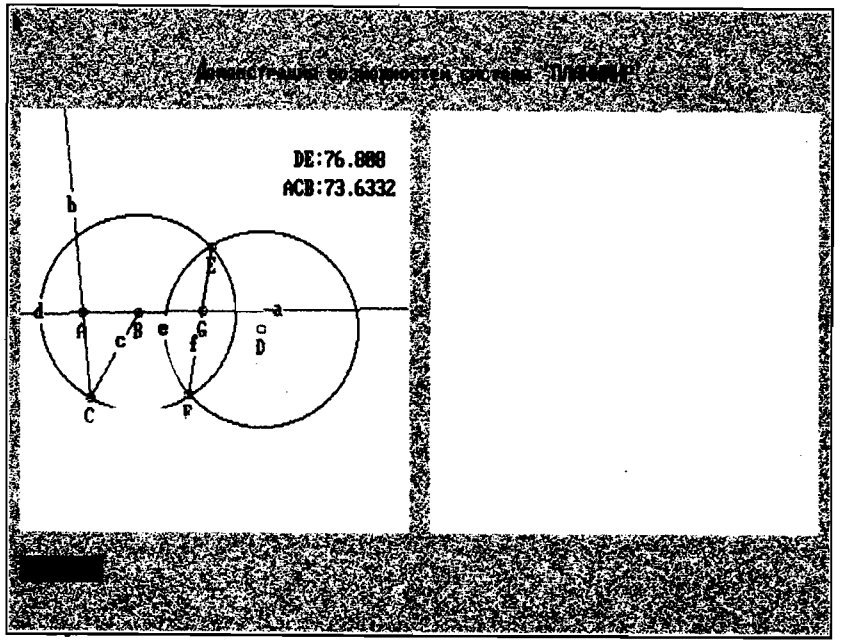


Рис. 7. Демонстрация работы в ПланиМире

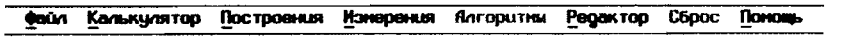


Рис. 8. Главное меню

другой современной системе, подобные действия в ПланиМире делаются в меню Файл. Щелкните левой кнопкой мыши на слове «Файл» в левом верхнем углу экрана. Если мыши нет, нажмите правую клавишу Alt и, не отпуская ее, клавишу Ф (такое нажатие ниже будем обозначать $Alt_{пр}$ Ф). Открывается меню. В нем вы можете выбрать требуемый способ выхода и нажать Enter.

Войдите еще раз. Выйти можно быстро, не обременяя себя входом в подменю. Нажмите последовательно Esc ↑ для выхода с сохранением построений и Esc End для выхода без сохранения.

Комбинации клавиш, которые нужно нажимать для выбора тех или иных команд, указаны в каждом меню.

В том же меню есть команды Загрузить и Записать. С помощью первой из них можно загружать другие чертежи. Например, войдите в среду ПланиМира и подайте команду Загрузить (т.е. войдите в меню Файл и выберите Загрузить или нажмите аккорд Alt F3). В ответ на запрос имени файла — введите demoload. Перед вами чертеж, который вы видели в демонстрации, его можно менять, а потом сохранить на диск или выйти без сохранения.

С помощью команды Записать можно периодически записывать чертеж на диск, не выходя из системы. Эта команда может оказаться полезной при неустойчивой работе компьютера или при сбоях питания: если вы пользовались этой командой, то возможно не придется восстанавливать всю картинку сначала, вы сможете продолжить работу с того момента, когда вы ее записали на диск.

13.2. Построение точек

Базовым объектом системы является *точка*. Все остальные фигуры являются производными от точек. Так, например, прямая проводится через две точки, а отрезок соединяет их. Важно научиться строить точки.

Войдем в среду ПланиМира. Чтобы поставить точку, нужно поставить мышью стрелочку на экране в нужное положение и нажать левую кнопку мыши. Точка появится возле острия стрелочки.

Строить точки можно и без мыши: нажмите $Alt_{пр}$ П, выберите в открывшемся меню команду Точка и нажмите Enter. На экране появится точка. Вместо этого, впрочем, можно просто нажать клавишу Т. На экране в случайном месте возникнет помеченная точка. Помеченные объекты легко двигать по плоскости чертежа.

Рассмотрим, как это делается. Предположим, что на экране помечена (выделена цветом) ровно одна точка. Установите стрелочку в любое свободное место на экране; нажмите правую кнопку мыши и, не отпуская кнопки, перемещайте мышь. Точка на экране повторяет движения мыши. Отпустите кнопку — мышь опять свободна, а точка переместилась. (Если помечено сразу несколько точек или других объектов, то при нажатой правой кнопке они двигаются одновременно.)

Помеченную точку можно двигать и без помощи мыши, управляющими клавишами [←], [↑], [↓], [→]. Если нажимать стрелки одновременно с клавишей [Shift], то перемещения будут в 4 раза быстрее, с клавишей [Ctrl] — в 16 раз быстрее.

13.3. Пометка объектов

Как помечать точку или снимать эту пометку? Очень просто. Укажите курсором точку и нажмите правую кнопку мыши. У точки изменится цвет, показывая, что команда выполнена. Если точка была помеченной, то она станет непомеченной, и наоборот. Аналогично можно работать и с другими объектами: нажатие правой кнопки мыши приводит к изменению состояния объекта, на который указывает курсор. Если объект был помечен, то пометка снимается. Если объект не был помечен, то он помечается. Попробуйте создать на экране еще пару точек и подвигать первую, вторую, а затем и все три.

Работать с пометками можно и без мыши, по командам. Иногда это удобнее, чем иметь дело с мышью. Чтобы снять пометку со всех объектов, нажмите [Alt$_{пр}$] [C] (C — первая буква слова *Сброс*).

Все объекты в ПланиМире имеют имена. При создании объект получает имя от системы. Однако в любой момент имя может быть изменено человеком. Чтобы пометить объект без мыши, нужно использовать его имя. Нажмите [Alt$_{пр}$] [P] (Редактор). Появляется подменю Редактора. Выберите в нем команду Пометить. В ответ на запрос имени объекта введите имя, например имя точки, и нажмите [Enter]. Точка станет помеченной. Если вы передумали помечать объект, когда на экране уже появился запрос на имя, то вместо [Enter] нажмите [Esc]. Для ускорения работы вместо входа в меню и выбора команды Пометить можно просто нажать клавишу [F5]. Попробуйте подвигать первую, вторую, все три точки, не используя мышь.

Выйдите из ПланиМира без сохранения результатов и снова войдите в него, чтобы выполнить следующее задание.

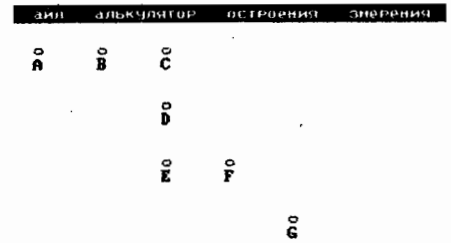


Рис. 9

Постройте точки так, как показано на рисунке 9. Переместите минимально возможное количество точек так, чтобы никакие 3 из них не лежали на одной прямой.

В системе ПланиМир есть средства проверить, лежат ли точки на одной прямой, но пока мы с ними не знакомы (да они в этой задаче и не нужны). Добейтесь того, чтобы «на глаз» было видно, что никакие три точки не лежат на одной прямой.

13.4. Работа с меню

С меню можно работать как с помощью мыши, так и без нее. Для того чтобы раскрыть подменю, необходимо поставить курсор на его заголовок и ***щелкнуть*** любой кнопкой мыши (слово *щелкнуть*, как обычно, обозначает кратковременное нажатие кнопки). Нажав правую клавишу [Alt] одновременно с подсвеченной («горячей») буквой выбираемого пункта, вы добьетесь того же результата. Исключением является позиция Помощь главного меню. Выбрать ее можно, нажав [Ctrl] [?].

Если при работе с выбранным подменю нажать клавиши [→] или [←], то активизируется позиция (раскроется соответствующее подменю), расположенная справа или слева от выбранного.

Позицию меню можно выбрать мышью (указать ее курсором и щелкнуть любой кнопкой) или с клавиатуры (подвести специальный курсор к нужному элементу с помощью стрелок и нажать [Enter]). Чтобы покинуть меню, не выполняя никаких действий, надо нажать [Esc].

Некоторые элементы меню могут быть изображены не яркими буквами, а тусклыми. Это означает, что в данный момент их нельзя выбрать, так как выполнение соответствующей команды невозможно.

13.5. Построение прямой

Войдите в среду ПланиМира и взгляните на главное меню, расположенное в верхней части экрана. Найдите пункт Построения и войдите в него. Перед вами меню со следующими командами: Точка, Точка на объекте, Окружность с радиусом, Окружность, Пересечь, Отрезок, Прямая, Луч. Рассмотрим их по очереди.

С построением точки вы уже познакомились ранее. Напомним лишь, что точка является наиболее простым, базовым объектом системы ПланиМир.

Построение прямой состоит из нескольких этапов. Первый из них — выбрать и пометить (или построить) точки, через которые будет проведена прямая.

> В ПланиМире есть только одна базовая команда построения прямой — через две отмеченные точки. Но пользователь ПланиМира может создать и свои собственные команды построения прямой: параллельная прямая, перпендикулярная прямая и т.д. Использование этих команд будет отличаться только тем, что их нужно выбирать из меню Алгоритмы, а не из меню Построения.

Далее следует выбрать пункт Прямая в меню Построения. На экране появится прямая. Войдите в подменю Построения. Заметьте, что справа от пункта Прямая стоит буква я. Как уже отмечалось, это означает, что если отмечены ровно две точки, то прямую через них можно провести простым нажатием клавиши [Я].

Построенная прямая помечена. Это общее правило в ПланиМире: *последний построенный объект помечен*.

13.6. Родители и дети

Введем понятия ***объектов-детей*** и ***объектов-родителей***. При построении прямой точки являются для прямой родителями, а прямая для точек — ребенком. (Заметим, что в ПланиМире родителей у объекта может быть не только 2, но и 0, 1, 3 и т.д. Разумеется, и детей у родителей может быть больше одного.) При построении родители теряют пометку, а ребенок получает. Отношение ***родители – ребенок*** полезно для описания команд перемещения объектов. Можно не очень строго сказать так:

> **все отношения между родителями и детьми не нарушаются при перемещениях.**

Сбросьте пометку со всех объектов (для этого нажмите [Alt пр] [C]). Пометьте одну из точек-родителей прямой. Перемещайте точку. Вы видите, что прямая поворачивается таким образом, что проходит через помеченную точку. Вообще для всех объектов верно, что все порожденные объекты модифицируются в соответствии с изменением порождающих.

Теперь снимите пометку с точки и пометьте прямую, двигайте ее. Она перемещается вместе с породившими ее точками. Это также общее свойство. При перемещении любого объекта перемещаются все его прародители.

13.7. Построение отрезка и луча

Теперь научимся строить отрезок. На всякий случай сбросьте пометку со всех объектов (нажмите [Alt пр] [C]). Постройте две точки. Войдите в меню Построения и выберите команду Отрезок (или нажмите клавишу [E]). На экране появился отрезок. Он помечен, подвигайте его. Сбросьте пометку и пометьте одну из точек, подвигайте и ее. Что можно сказать об отношениях между родителями и ребенком в этом примере?

Луч строится аналогично прямой и отрезку, но имеется одно важное отличие. При построении прямой или отрезка порядок выбора двух точек-родителей был не важен. В случае луча это уже не так: первая отмеченная точка будет вершиной луча, а вторая будет лежать на луче.

13.8. Построение окружностей

Система ПланиМир позволяет строить окружности двумя способами. Первый из них — окружность по заданному центру и точке, принадлежащей этой окружности. Второй — окружность по заданному центру и радиусу (радиус задается как расстояние между двумя точками).

Рассмотрим первый способ. Постройте две точки. Выберите в меню Построения команду Окружность. Появилась окружность, первая точка стала центром, вторая лежит на окружности. Попробуйте подвигать точку, лежащую на окружности, — окружность раздувается и сжимается. Подвигайте центр — теперь окружность перемещается и одновременно растягивается, подвигайте обе точки — окружность просто перемещается.

Рассмотрим второй случай. Построим окружность по центру и радиусу. Для этого понадобятся три точки. Постройте их и выберите команду Окружность с радиусом в меню Построения (или нажмите [Shift] [O]). Построенная окружность будет иметь центр в первой отмеченной точке и радиус, равный расстоянию между второй и третьей.

13.9. Построение точек, лежащих на объектах

В системе ПланиМир точки могут быть трех видов:

1) свободные;
2) принадлежащие заданному объекту (прямой, отрезку, лучу, окружности);
3) точки пересечения объектов.

Построение свободных точек мы рассмотрели выше.

Точки, принадлежащие объектам, строятся так же, как и свободные (нажатием левой кнопки мыши), но курсор должен указывать при этом не на свободное место экрана, а на некоторое положение на объекте. Попробуйте создать по точке на построенных ранее прямой, отрезке, луче, окружности. Построенные таким образом точки навсегда связаны с объектом и перемещаются вместе с ним.

Наконец, третий способ получения точек — пересечение двух объектов. Если пересекаются прямые, отрезки, лучи в любой комбинации, то может получиться либо одна точка, либо ни одной (если объекты не пересекаются). Возьмите построенный ранее отрезок и переместите его так, чтобы он пересекался с ранее построенной прямой. Пометьте прямую, пометьте отрезок и нажмите П. Появится точка пересечения. Того же эффекта можно достичь, войдя в меню Построения и выбрав команду Пересечь. Отметьте еще одну точку и соедините ее прямой с точкой пересечения.

Теперь подвиньте отрезок так, чтобы он перестал пересекаться с прямой. Посмотрите, что произошло. Во-первых, исчезла точка пересечения отрезка с прямой. Во-вторых, исчезла и прямая, проходившая через исчезнувшую точку. В системе ПланиМир в том случае, когда исчезает какой-то объект, исчезают и все его потомки. Но исчезают они не бесследно, память о них остается. Проверьте это. Переместите отрезок так, чтобы он опять пересек прямую. Из небытия появилась точка вместе с дочерней прямой.

13.10. Измерения и вычисления

Для выявления многих геометрических фактов бывают нужны измерения и вычисления. Система ПланиМир позволяет их проводить. Вы имеете возможность измерять углы в градусах (от 0 до 180) и расстояние в условных единицах.

Попробуем измерить расстояние между двумя точками. Постройте две точки, нажмите Р или войдите в меню Измерения и выберите пункт Расстояние. На экране появится результат измерения — число. Это число — полноправный объект системы ПланиМир. У него есть родители — две точки, расстояние между которыми мы измеряли. Этот объект помечен, что позволяет переместить его в удобное место на экране. Позже мы увидим, что у этого объекта могут быть и дети — другие числа.

Угол измеряется похожим образом. Поставьте третью точку и пометьте две, поставленные ранее. Войдите в подменю Измерения и выберите пункт Угол или просто нажмите У. Появилось второе число. Пусть сначала была помечена точка A, потом точка B, а точка C оказалась помеченной в процессе ее создания. Тогда новое число равно углу ABC. Если же сбросить пометки, а потом пометить точки A, C, B в указанном порядке и измерить угол, то будет измерен угол ACB.

Наряду с измерениями приходится производить и вычисления. Например, бывает нужно сложить два расстояния или взять тангенс угла.

Вычислительные операции в ПланиМире делятся на три группы:

1) операция Число не нуждается в аргументах и позволяет ввести произвольное число;
2) операции arctg, arcctg, arcsin, arccos, lg, ctg, sin, cos, tg, ln, exp, neg, abs, sqrt и копия нуждаются в одном аргументе;
3) операции сумма, разность, произведение, отношение, степень нуждаются в двух аргументах.

Рассмотрим подробно все эти три группы.

1) Выполните операцию Число. Система попросит ввести число в десятичном виде. Если ввести какое-нибудь число и нажать Enter, то число появится на экране. Если нажать Esc — запрос пропадет, а на экране ничего не появится. С помощью этой команды можно вводить произвольные числа, необходимые в вычислениях.

2) Во второй группе рассмотрим, например, операцию tg, остальные операции этой группы аналогичны. Пометьте число, полученное при измерении угла. Выберите в меню Калькулятор позицию tg. Появилось число, равное тангенсу помеченного числа. Обратите внимание на то, что с использованного значения пометка снимается, а вновь полученное значение становится помеченным.

Попробуйте перемещать одну из точек угла. Изменяется значение угла и его тангенса. Если вам удастся получить прямой угол, то число, соответствующее тангенсу, исчезнет, так как тангенс прямого угла не определен.

Заметим в заключение, что операция копия просто-напросто копирует помеченное число.

3) В третьей группе рассмотрим операцию разность, остальные операции этой группы аналогичны. Пометим два числа, выполним операцию разность. На экране появится разность между числом, помеченным первым, и числом, помеченным вторым.

Постройте три точки, произведите измерения и вычисления, необходимые для выяснения, лежат ли эти точки на одной прямой. Переместите точки так, чтобы они лежали на одной прямой.

13.11. Сервисные возможности системы

Для повышения удобства работы с системой предусмотрены некоторые дополнительные возможности. Все они доступны через меню Редактор.

Команда Откатить отменяет действие, выполненное последним. Если это было построение прямой, — прямая исчезнет с экрана. Если это было измерение расстояния, — с экрана исчезнет полученное число и т.д. При повторной «откатке» будет отменено предпоследнее действие и т.д.

Команда Накатить. Что делать, если, применяя команду Откатить, вы увлеклись и откатились слишком далеко? В таком случае нужно применить команду Накатить. Эта команда восстанавливает действие, удаленное последней командой Откатить.

Команда Переименовать. Имена, которые система автоматически дает объектам, могут оказаться неудобными. В этом случае можно переименовать объект. Для этого его надо пометить и выполнить команду Переименовать. Запрашивается новое имя, которое вступит в силу после нажатия клавиши Enter.

Команды Увеличить и Уменьшить. ПланиМир помогает избавиться от неудобств, связанных с размерами чертежа. Если чертеж вышел за пределы экрана, воспользуйтесь командой Уменьшить и чертеж поместится на экране. Наоборот, если чертеж оказался слишком мал, его можно увеличить с помощью команды Увеличить.

Команды Спрятать и Показать. В процессе выполнения чертежа часто бывает необходимо сделать некоторые вспомогательные построения. Иногда вспомогательные объекты, возникшие в таких построениях, загромождают рисунок и ухудшают восприятие. Такие объекты можно убрать с помощью команды Спрятать. Эта команда прячет все помеченные объекты. Команда Показать возвращает на экран все ранее спрятанные объекты.

Команды Спрятать имя и Показать имя. При геометрических построениях имя объекта иногда только загромождает чертеж. Поэтому предусмотрена возможность прятать или показывать имена помеченных объектов.

13.12. Элементарные построения

Теперь, освоив приемы работы с ПланиМиром, попробуем решить с его помощью несколько школьных задач.

Условия приведенных задач заимствованы из школьного учебника «Геометрия» А. В. Погорелова. Запустите гипертекст и выберите Геометрический практикум. Там выберите тему и, наконец, саму задачу. Для некоторых задач приведено решение, его можно посмотреть по шагам. В каждой теме предусмотрены как решенные задачи, так и задачи для самостоятельного решения.

Построение треугольника с заданными сторонами

Задача. Построить треугольник с заданными сторонами AB, CD, EF.

Решение. Проводим произвольную прямую и отмечаем на ней произвольную точку X. Описываем окружность с центром X и радиусом AB. Пусть Y — точка ее пересечения с прямой. Теперь описываем окружность с центром X и радиусом CD и окружность с центром Y и радиусом EF. Пусть Z — точка пересечения этих окружностей. Проведем отрезки XZ и YZ. Треугольник XYZ имеет стороны AB, CD, EF.

Упражнения:

42. Дан треугольник ABC. Постройте равный ему треугольник ABD.

43. Постройте окружность радиуса AB, проходящую через точки C и D.

Построение угла, равного данному

Задача. Отложить от полупрямой a в нижнюю полуплоскость угол, равный углу A.

Решение. Проведем произвольную окружность с центром в вершине A данного угла. Пусть B и C — точки пересечения окружности со сторонами угла. Радиусом AB проведем окружность с центром в точке O — начальной точке данной полупрямой. Точку пересечения этой окружности с данной полупрямой обозна-

чим B_1. Опишем окружность с центром B_1 и радиусом BC. Точка C_1 пересечения построенных окружностей в указанной полуплоскости лежит на стороне искомого угла.

Доказательство. Для доказательства достаточно заметить, что треугольники ABC и OB_1C_1 равны как треугольники с соответственно равными сторонами. Углы A и O являются соответствующими углами этих треугольников.

Упражнения:

44. Постройте треугольник XYZ такой, что $XY = AB$, $YZ = CD$, $\angle Y = \angle E$.

45. Постройте равнобедренный треугольник XYZ такой, что $XY = YZ = AB$, $\angle X = \angle C$.

Построение биссектрисы угла

Задача. Построить биссектрису угла A.

Решение. Из вершины A данного угла как из центра описываем окружность произвольного радиуса. Пусть B и C — точки ее пересечения со сторонами угла. Из точек B и C тем же радиусом описываем окружности. Пусть D — точка их пересечения, отличная от A. Проводим полупрямую AD.

Доказательство. Луч AD является биссектрисой, так как делит угол BAC пополам. Это следует из равенства треугольников ABD и ACD, у которых углы DAB и DAC являются соответствующими.

Упражнения:

46. Постройте окружность, вписанную в треугольник ABC.

47. Разделите угол A на четыре части.

Деление отрезка пополам

Задача. Разделить отрезок AB пополам.

Решение. Пусть AB — данный отрезок. Из точек A и B радиусом AB описываем окружности. Пусть C и C_1 — точки пересечения этих окружностей. Они лежат в разных полуплоскостях относительно прямой AB. Отрезок CC_1 пересекает прямую в некоторой точке O. Эта точка есть середина отрезка AB.

Доказательство. Действительно, треугольники CAC_1 и CBC_1 равны по трем сторонам. Отсюда следует равенство углов ACO и BCO. Треугольники ACO и BCO равны по двум сторонам и углу между ними. Стороны AO и BO этих треугольников являются

соответствующими, а поэтому они равны. Таким образом, O — середина отрезка AB.

Упражнения:

48. Дан треугольник ABC. Постройте его медианы.

49. Постройте треугольник XYZ такой, что $XY = AB$, $YZ = CD$, $ZU = EF$, где ZU — медиана, проведенная из Z к стороне XY.

13.13. Более сложные построения

Овладев методикой выполнения простейших построений, можно теперь приступить к более сложным построениям, использующим элементарные.

Построение перпендикулярной прямой

Задача. Через данную точку O провести прямую, перпендикулярную данной прямой a.

Решение. Возможны два случая:

1) точка O лежит на прямой a,
2) точка O не лежит на прямой a.

Рассмотрим первый случай. Из точки O проводим произвольным радиусом окружность. Она пересекает прямую a в двух точках: A и B. Из точек A и B проводим окружности радиусом AB. Пусть C — точка их пересечения. Искомая прямая проходит через точки O и C.

Доказательство. Перпендикулярность прямых OC и AB следует из равенства углов при вершине O треугольников ACO и BCO. Эти треугольники равны по трем сторонам.

Рассмотрим второй случай. Из точки O проводим окружность, пересекающую прямую a. Пусть A и B — точки ее пересечения с прямой a. Из точек A и B тем же радиусом проводим окружности. Пусть O_1 — точка их пересечения, лежащая в полуплоскости, отличной от той, в которой лежит точка O. Искомая прямая проходит через точки O и O_1.

Доказательство. Обозначим через C точку пересечения прямых AB и OO_1. Треугольники AOB и AO_1B равны по трем сторонам. Поэтому угол OAC равен углу O_1AC. А тогда треугольники OAC и O_1AC равны по двум сторонам и углу между ними. Значит, углы ACO и ACO_1 равны. А так как они смежные, то они прямые. Таким образом, OC — перпендикуляр, опущенный из точки O на прямую a.

Упражнения:

50. Дан треугольник ABC. Постройте его высоты.

51. Постройте окружность, описанную около треугольника ABC.

Метод геометрических мест

Сущность метода геометрических мест, используемого при решении задач на построение, состоит в следующем. Пусть, решая задачу на построение, надо найти точку X, удовлетворяющую двум условиям. Геометрическое место точек, удовлетворяющих первому условию, есть некоторая фигура F_1, а геометрическое место точек, удовлетворяющих второму условию, есть некоторая фигура F_2. Искомая точка X принадлежит F_1 и F_2, т.е. является их точкой пересечения. Если эти геометрические места простые (например, состоят из прямых и окружностей), то можно их построить и найти интересующую нас точку X.

Задача. Даны три точки A, B, C. Постройте точку X, которая одинаково удалена от точек A и B и находится на расстоянии EF от точки C.

Решение. Искомая точка X удовлетворяет двум условиям:

1) она одинаково удалена от точек A и B;

2) она находится на данном расстоянии от точки C.

Геометрическое место точек, удовлетворяющих первому условию, есть прямая, перпендикулярная отрезку AB и проходящая через его середину. Геометрическое место точек, удовлетворяющих второму условию, есть окружность данного радиуса с центром в точке C. Искомая точка X лежит на пересечении этих геометрических мест.

Упражнения:

52. На данной прямой a найти точку, которая находится на расстоянии AB от прямой b.

53. На данной прямой a найти точку, равноудаленную от двух данных точек A и B.

Параллелограммы и ромбы

Задача. Постройте параллелограмм по двум сторонам и диагонали.

Решение. Построим треугольник ABC по трем сторонам, причем так, что AB имеет длину диагонали, а BC и AC — сторон. Построим треугольник ABD такой, что $AD = BC$, $AC = BD$. Четырехугольник $ACBD$ — параллелограмм.

Доказательство. Треугольник ABC равен треугольнику BDA по трем сторонам. Углы DAB и CBA и углы DBA и CAB равны как соответствующие в равных треугольниках. Прямые CA и DB и прямые CB и DA параллельны по признаку параллельности прямых.

Упражнения:

54. Постройте параллелограмм $WXYZ$ такой, что:

а) $WX = AB$, $WY = CD$, $XZ = EF$;

б) $WX = AB$, $XY = CD$, $\angle X = \angle E$;

в) $WY = AB$, $XZ = CD$, угол между ними равен $\angle E$.

55. Постройте ромб $WXYZ$ такой, что:

а) $\angle W = \angle A$, $WY = BC$;

б) $\angle W = \angle A$, $XZ = BC$;

в) $WX = AB$, $WY = CD$;

г) $WY = AB$, $XZ = CD$.

§ 14. КАК УСТРОЕН ПЛАНИМИР

Попробуем теперь разобраться во внутреннем устройстве ПланиМира. Чертеж, который изображается на экране, должен каким-то образом запоминаться в ПланиМире, чтобы впоследствии с ним можно было выполнять все требуемые действия.

14.1. Модель чертежа в ПланиМире

Вообще говоря, есть много разных способов запомнить чертеж. Например, можно попытаться запомнить, какие пикселы на экране компьютера окрашены в те или иные цвета. Или можно хранить информацию обо всех объектах, которые были созданы при построении чертежа.

В ПланиМире принят другой способ хранения чертежа. Грубо говоря, чертеж не хранится вовсе, а хранятся лишь команды, которые последовательно давал человек, строя чертеж. Такая последовательность команд — ***протокол построения*** — позволяет в любой момент построить чертеж заново. Такой метод хранения чертежа позволяет оперативно модифицировать чертеж при изменении одного или нескольких объектов.

Этот метод хранения чертежа напоминает запись шахматной позиции в виде последовательности ходов, приведших к ней.

Осталось только понять, кому же дает команды человек, строя чертеж, и в каком виде записывается протокол.

Ответ таков: команды даются специальному исполнителю PLANE, а протокол записывается в виде алгоритма на школьном алгоритмическом языке.

Рассмотрим алгоритм, соответствующий чертежу, увиденному в самом начале изучения ПланиМира в разделе Демонстрация (рис. 7). Каждая команда этого алгоритма предваряется комментарием:

```
алг
нач
| | Ставим точку A
| точка("A",да,да,-103.219,-6.10308)
| | Ставим точку B
| точка("B",да,да,-60.9812,-6.7732)
| | Проводим прямую a через точки A и B
| прямая("a",да,да,"A","B")
| | Ставим точку C
| точка("C",да,да,-97.084,61.021)
| | Проводим луч b из точки C в точку A
| луч("b",да,да,"C","A")
| | Строим отрезок c с концами в точках C и B
| отрезок("c",да,да,"C","B")
| | Строим окружность d с центром в B,
| | проходящую через C
| окружность("d",да,да,"B","C")
| | Ставим точку D
| точка("D",да,да,36.5449,8.14318)
| | Проводим окружность e с центром в D и радиусом CB
| окружность с радиусом("e",да,да,"D","C","B")
| | Пересекаем  окружности d и e,
| | точки пересечения - E и F
| пересечь окружность("E",да,да,"F",да,да,"d","e")
| | Строим отрезок f с концами E и F
| отрезок("f",да,да,"E","F")
| | Пересекаем отрезок f с прямой a,
| | точка пересечения - G
| пересечь("G",да,да,"f","a")
| | Измеряем расстояние от точки D до точки E
| расстояние("DE",да,да,"D","E",86.,-134.4)
| | Измеряем угол ACB
| угол("ACB",да,да,"A","C","D",86.,-112.457)
кон
```


При работе в ПланиМире в памяти компьютера хранится протокол — алгоритм на языке КуМир, описывающий построение. При выполнении этого алгоритма и возникает чертеж.

При решении задач на построение и других работах в ПланиМире этот протокол не нужен. Но при желании его можно посмотреть, выбрав команду Протокол в меню Редактор. Вернуться к чертежу после просмотра протокола можно нажав Alt Tab.

14.2. Команда «точка»

Чтобы получить общее представление об этом исполнителе, рассмотрим подробно команду точка.

```
точка (арг точ имя_точки, плог п,и, коорд x,y)
```

Эта команда ставит точку с координатами x, y и именем имя_точки. Ее аргументы:

имя_точки — имя создаваемой точки;

п,и — показывать ли точку/имя точки (да — показывать, нет — не показывать);

x,y — координаты создаваемой точки (начало координат находится в центре экрана).

В исполнителе PLANE вместо типа лог используется тип плог, а вместо типа вещ — тип коорд. Это сделано для упрощения реализации этого исполнителя.

Рассмотрим пример. Войдите в раздел Свободная работа в ПланиМире и создайте точку. Посмотрите протокол. Вы увидите:

```
алг
нач
| точка("A",да,да,x,y)
кон
```

где вместо x и y стоят некоторые числа, определяющие положение точки. Подвигайте точку — x и y изменились. Спрячьте точку, теперь программа имеет вид:

```
алг
нач
| точка("A",нет,да,x,y)
кон
```

Покажите точку и спрячьте ее имя:

```
алг
нач
| точка("A",да,нет,x,y)
кон
```

Постройте еще несколько объектов: отрезок, окружность, прямую. Посмотрите, как изменился протокол. Экспериментируя, вы легко сможете разобраться в любом протоколе системы ПланиМир.

Если возникнут неясности или сомнения, их поможет разрешить следующий пункт.

14.3. Список команд исполнителя PLANE

```
точка на прямой(арг точ имя_точки,арг плог п,и,
                арг прям имя_прямой,арг коорд t)
```

Ставит точку с именем имя_точки на прямую с именем имя_прямой в положение, определяемое параметром t.

```
точка на луче(арг точ имя_точки,арг плог п,и,
              арг лч имя_луча,арг коорд t)
```

Ставит точку с именем имя_точки на луч с именем имя_луча в положение, определяемое параметром t.

```
точка на отрезке(арг точ имя_точки,арг плог п,и,
                 арг отр имя_отрезка,арг коорд t)
```

Ставит точку с именем имя_точки на отрезок с именем имя_отрезка в положение, определяемое параметром t.

```
точка на окружности(арг точ имя_тчк,арг плог п,и,
                    арг окр имя_окр,арг коорд t)
```

Ставит точку с именем имя_тчк на окружность с именем имя_окр в положение, определяемое параметром t.

```
прямая(арг прям имя_прямой,арг плог п,и,
       арг точ имя_точки1,имя_точки2)
```

Строит прямую с именем имя_прямой, проходящую через точки с именами имя_точки1 и имя_точки2.

```
отрезок(арг отр имя_отрезка,арг плог п,и,
        арг точ имя_точки1,имя_точки2)
```

Строит отрезок с именем имя_отрезка, с концами в точках с именами имя_точки1 и имя_точки2.

```
луч(арг лч имя_луча,арг плог п,и,
    арг точ имя_точки1,имя_точки2)
```

Строит луч с именем имя_луча, с началом в точке с именем имя_точки1 и проходящий через точку с именем имя_точки2.

```
окружность(арг окр имя_окр,арг плог п,и,
           арг точ имя_точки1,имя_точки2)
```

Строит окружность с именем имя_окр, с центром в точке с именем имя_точки1 и проходящую через точку с именем имя_точки2.

```
окружность с радиусом(арг окр имя_окр,
                      арг плог п,и,
                      арг точ имя1,имя2,имя3)
```

Строит окружность с именем имя_ок, с центром в точке с именем имя_1 и радиусом, равным расстоянию от точки с именем имя_2 до точки с именем имя_3.

```
пересечь(арг точ имя_точки,арг плог п,и,
         арг тип1 имя1,арг тип2 имя2)
```

Строит точку с именем имя_точки, являющуюся точкой пересечения прямой, луча или отрезка с именем имя1 и прямой/луча/отрезка с именем имя2.

В этой команде *тип1* и *тип2* — это один из типов: отр (отрезок), прям (прямая), лч (луч).

```
пересечь окружность(арг точ имя1,арг плог п1,и1,
                    арг точ имя2,арг плог п2,и2,
                    арг окр имя_окр,арг тип имя)
```

Строит точки с именами имя1 и имя2, являющиеся точками пересечения окружности с именем имя_окр и окружности/прямой/луча/отрезка с именем имя.

В этой команде *тип* — это один из типов: отр (отрезок), прям (прямая), лч (луч), окр (окружность).

```
число(арг числ имя_числа,арг плог п,и,
      арг вещ f,арг коорд x,y)
```

Создает объект «число» со значением f и помещает его в позицию (x,y).

```
расстояние(арг числ имя_числа,арг плог п,и,
           арг точ имя_точки,арг тип имя,
                          арг коорд x,y)
```

Создает объект «число» со значением, равным расстоянию от точки с именем имя_точки до точки/прямой/отрезка/луча/окружности с именем имя, и помещает его в позицию (x,y).

```
угол(арг числ имя_числа,арг плог п,и,
     арг точ A,B,C,арг коорд x,y)
```

Создает объект «число» со значением, равным углу ABC, и помещает его в позицию (x,y).

```
операция(арг числ имя_числа,арг плог п,и,
         арг числ A,B,арг коорд x,y)
```

Создает объект «число» со значением, равным операции (A,B), и помещает его в позицию (x,y); *операция* — одна из операций: сумма, разность, произведение, отношение, степень.

```
функция(арг числ имя_числа,арг плог п,и,
        арг числ A,арг коорд x,y)
```

Создает объект «число» со значением, равным функции (A), и помещает его в позицию (x,y); *функция* — одна из функций: копия, neg, sqrt, abs, exp, ln, lg, sin, cos, tg, ctg, arccos, arcsin, arctg, arcctg (где копия(A) = A, neg(A) = -A, остальные функции общеизвестны).

14.4. Непосредственное программирование исполнителя PLANE

Протокол любого построения в ПланиМире является линейной программой — он не содержит команд ветвления, циклов и т.п. Однако КуМир позволяет написать для исполнителя PLANE алгоритм, использующий любые управляющие конструкции. Вот пример такого алгоритма:

```
алг
нач лит к, лог т
| т := да
| нц пока т
| | к := getkey()
| | выбор
| | | при к = 'A':
| | |  точка('A',да,да,0.,0.)
| | |  т := нет
| | | при к = 'B':
| | |  точка('B',да,да,0.,0.)
| | |  т := нет
| | все
| кц
кон
```


14.5. Вспомогательные алгоритмы

При составлении программ часто бывает удобным использование вспомогательных алгоритмов. Например, один раз построив середину отрезка, удобно оформить это как вспомогательный алгоритм и впоследствии использовать его. Это можно сделать вручную и вставлять его вызов при непосредственном программировании чертежа. Система позволяет также специально оформленные алгоритмы вызывать непосредственно в процессе построения. Для этого используется меню Алгоритмы.

Пункт Алгоритмы главного меню обычно не активен. Однако стоит создать собственные алгоритмы, и тогда этот пункт станет активен и вы сможете использовать собственные алгоритмы наряду с базовыми командами ПланиМира. Некоторые особо часто используемые и полезные алгоритмы уже реализованы разработчиками ПланиМира и записаны в файл TOOLS.E. В этот файл входят:

1) построение середины отрезка, заданного точками–концами;
2) построение биссектрисы угла, заданного тремя точками;
3) построение прямой, перпендикулярной данной и проходящей через заданную точку;
4) построение прямой, параллельной данной и проходящей через заданную точку.

Чтобы использовать этот файл, выберите в гипертексте тему Подпрограммы в задачах на построение и далее тему Демонстрация библиотеки вспомогательных алгоритмов.

«Откройте» меню Алгоритмы, и вы увидите список имеющихся вспомогательных алгоритмов, ни один из них не активен, так как алгоритм становится активным, когда на экране помечены необходимые алгоритму аргументы. Попробуйте построить на эк-

ране несколько объектов, пометить их и запустить по очереди все доступные вспомогательные алгоритмы.

Как самому составить вспомогательный алгоритм-построение

Перейдите в протокол, нажав Alt Tab. Вставьте в начало файла шаблон алгоритма, нажав Esc A. Вернитесь к построениям, нажав снова Alt Tab, и выполните вспомогательное построение.

Снова перейдите в протокол. Вы увидите, что во вставленный ранее шаблон алгоритма уже вписаны команды построения. Однако полученный алгоритм надо слегка изменить. Рассмотрим сначала пример. Пусть вспомогательное построение было построением середины заданного отрезка и протокол выглядит так:

Протокол построения середины отрезка:

```
алг
нач
| точка("A",да,да,10.,20.)
| точка("B",да,да,50.,33.)
| отрезок("a",нет,нет,"A","B")
| окружность("b",нет,нет,"B","A")
| окружность("c",нет,нет,"A","B")
| пересечь окружность("C",нет,нет,"D",нет,нет,"b","c")
| отрезок("d",нет,нет,"D","C")
| пересечь("E",да,да,"d","a")
кон
```

В этом случае алгоритм должен быть изменен так:

Алгоритм построения середины отрезка:

```
алг середина(арг точ E,арг плог п1,и1,точ A,B)
нач
| отрезок("a",нет,нет,A,B)
| окружность("b",нет,нет,B,A)
| окружность("c",нет,нет,A,B)
| пересечь окружность("C",нет,нет,"D",нет,нет,"b","c")
| отрезок("d",нет,нет,"D","C")
| пересечь(E,п1,и1,"d","a")
кон
```

В общем случае надо внести следующие изменения в протокол построения:

1) ввести имя алгоритма в строке алг;
2) удалить построение объектов, которые являлись аргументами, и описать их как аргументы алгоритма;
3) конкретные имена и параметры (показывать ли объект и его имя) заменить на переменные, передаваемые, как параметры.

Заголовок алгоритма должен принять вид:

алг имя_алг (арг *тип* имя_рез$_1$,арг плог п$_1$,и$_1$,...
арг *тип* имя_арг$_1$,...)

где

тип — тип результата/аргумента: точ (точка), отр (отрезок), прям (прямая), лч (луч), окр (окружность), числ (число);

имя_рез$_i$ — имя i-го результата;

п$_i$ — показывать (да) или не показывать (нет) на экране этот результат;

и$_i$ — показывать (да) или не показывать (нет) на экране имя этого результата;

имя_арг$_i$ — имя i-го аргумента.

Количество аргументов может быть любым. Алгоритм будет выполняться только в том случае, когда помечено достаточное количество аргументов нужных типов. Например, нельзя найти середину отрезка, если помечена всего одна точка (даже если еще помечено несколько прямых).

14.6. Теорема о пропорциональных отрезках

Хорошей иллюстрацией к вышесказанному может послужить задача о построении пропорциональных отрезков. В ее решении используется вспомогательный алгоритм построения параллельной прямой, который, в свою очередь, использует вспомогательный алгоритм построения перпендикулярной прямой. Для работы над этой задачей выберите пункт Подпрограммы в задачах на построение.

Задача. Даны отрезки a, b, c. Построить отрезок $x = (bc)/a$.

Решение. Строим любой неразвернутый угол с вершиной O. Откладываем на одной стороне угла отрезки $OA = a$ и $OB = b$, а на другой стороне отрезок $OC = c$. Соединяем точки A и C прямой и проводим параллельную ей прямую BD через точку B. Отрезок $OD = x$.

Доказательство. Действительно, по теореме о пропорциональных отрезках $OA/OB = OC/OD$. Отсюда $OD = (OB \times OC)/OA = (bc)/a$. Таким образом, отрезок OD есть искомый отрезок x.

Замечание. Построенный отрезок x называется *четвертым пропорциональным*. Это название связано с тем, что он является четвертым членом пропорции $a : b = c : x$.

Протокол построения:

```
алг
нач
  точка("O",да,да,-88.586,31.5184)
  точка("ф",нет,да,57.1845,-42.8081)
  точка("ы",нет,да,77.1161,55.0204)
  луч("a",да,нет,"O","ф")
  луч("b",да,нет,"O","ы")
  точка на луче("A",да,да,"a",0.801433)
  точка на луче("B",да,да,"a",1.1405)
  точка на луче("C",да,да,"b",0.777352)
  прямая("c",да,нет,"A","C")
  параллель("d",да,нет,"c","B")
  пересечь("D",да,да,"b","d")
кон
алг параллель(прям c,плог п1,и1,прям a,точ C)
нач
  перпендикуляр("b",нет,да,a,C)
  перпендикуляр(c,п1,и1,"b",C)
кон
алг перпендикуляр(арг прям e,плог п1,и1,прям a,точ C)
нач
  точка на прямой("D",нет,да,a,0.168589)
  окружность("b",нет,да,C,"D")
  пересечь окружность("E",нет,да,"F",нет,да,"b",a)
  окружность("c",нет,да,"E","F")
  окружность("d",нет,да,"F","E")
  пересечь окружность("G",нет,да,"H",нет,да,"c","d")
  прямая(e,п1,и1,C,"H")
кон
```

Заметим, что два вспомогательных алгоритма параллель и перпендикуляр при решении задачи считаются данными и решать задачу надо с их использованием.

Упражнение:

56. Даны отрезки a, b, c, d, e. Постройте отрезок $x = (abc)/(de)$.

Оглавление

9 класс

10 класс

Учебное издание

Кушниренко Анатолий Георгиевич
Леонов Александр Георгиевич
Эпиктетов Михаил Геннадьевич
и другие

ИНФОРМАЦИОННАЯ КУЛЬТУРА

Кодирование информации
Информационные модели
Класс 9—10

Учебник для общеобразовательных учебных заведений

Ответственный редактор *М. Г. Циновская*
Художник обложки *А. В. Кузнецов*
Оригинал-макет подготовил *М. Г. Эпиктетов*
Корректор *Т. К. Остроумова*

ЛР № 061622 от 23 сентября 1992 г.

Подписано в печать с готовых диапозитивов 17.06.96. Формат 60×90 1/16.
Бумага кн.-журн. Гарнитура «Школьная». Печать офсетная. Усл. печ. л. 13,0.
Уч.-изд. л. 14,95. Тираж 50 000 экз. Заказ № 5640.

Издательский дом «Дрофа».
127018, Москва, Сущевский вал, 49.

Отпечатано на Смоленском полиграфическом комбинате Комитета Российской Федерации по печати. 214020, Смоленск, ул. Смольянинова, 1.