

MINISTERUL EDUCAȚIEI, CULTURII ȘI CERCETĂRII

ИНФОРМАТИКА

Учебник для 11-го класса

АНАТОЛ ГРЕМАЛСКИ • ЮРИЕ МОКАНУ
ЛУДМИЛА ГРЕМАЛСКИ



Știința, 2021

Acest manual este proprietatea Ministerului Educației, Culturii și Cercetării.

Manualul școlar a fost realizat în conformitate cu prevederile Curriculumului la disciplină, aprobat prin Ordinul Ministerului Educației, Culturii și Cercetării nr. 906 din 17 iulie 2019. Manualul a fost aprobat prin Ordinul Ministerului Educației, Culturii și Cercetării nr. 1219 din 6 noiembrie 2020, urmare a evaluării calității metodico-științifice.

Editat din sursele financiare ale *Fondului Special pentru Manuale*.

Comisia de evaluare:

Svetlana Brînză, profesor, grad didactic superior, Liceul Teoretic „N.M. Spătaru”, metodist DGETS, Chișinău (coordonator);

Gheorghe Chistruga, profesor, grad didactic superior, Liceul Teoretic „M. Eminescu”, Drochia;

Natalia Schițco, profesor, grad didactic superior, Liceul Teoretic „Socrate”, Chișinău;

Violina Bargan, profesor, grad didactic superior, Liceul Teoretic „Gh. Asachi”, Chișinău;

Ecaterina Adam, profesor, grad didactic unu, Liceul de Creativitate și Inventică „Prometeu-Prim”, Chișinău

Denumirea instituției de învățământ _____				
Manualul a fost folosit: _____				
Anul de folosire	Numele, prenumele elevului	Anul de studii	Aspectul manualului	
			la primire	la returnare

Dirigintele verifică dacă numele, prenumele elevului sunt scrise corect.

Elevii nu vor face niciun fel de însemnări în manual.

Aspectul manualului (la primire și la returnare) se va aprecia cu unul dintre următorii termeni: *nou, bun, satisfăcător, nesatisfăcător*.

Responsabil de ediție: Larisa Dohotaru

Redactor: Larisa Nosacenco

Corector: Alefina Olari

Redactor tehnic: Nina Duduciuc

Machetare computerizată: Tudor Jalbă

Copertă: Romeo Șveț

ÎNȚREPRINDEREA
EDITORIAL-POLIGRAFICĂ

ȘTIINȚA

str. Academiei, nr. 3; MD-2028, Chișinău, Republica Moldova

tel.: (+373 22) 73-96-16; fax: (+373 22) 73-96-27

e-mail: prini_stiinta@yahoo.com

www.editurastiinta.md

Toate drepturile asupra acestei ediții aparțin Întreprinderii Editorial-Poligrafice *Știința*.

Descrierea CIP a Camerei Naționale a Cărții

Информатика: Учебник для 11-го класса / Анато́л Гремалски, Ю́рие Мокану, Лудмила Гремалски; traducere din română: Veronica Mustață [et al.] ; comisia de evaluare: Svetlana Brînză (coordonator) [et al.] ; Ministerul Educației, Culturii și Cercetării. – [Ed. a 3-a]. – Ch.: Î.E.P. *Știința*, 2021 (Combinatul Poligrafic). – 224 p. : fig., tab.

Proprietate a Min. Educației, Culturii și Cercet.

ISBN 978-9975-85-262-3

Imprimare la **COMBINATUL POLIGRAFIC**

str. Petru Movilă, 35 MD-2004, Chișinău, Republica Moldova Comanda nr.

© Anatol Gremalschi, Iurie Mocanu, Ludmila Gremalschi. 2008, 2014, 2021

© Traducere din română: Veronica Mustață, Arcadie Malearovici, Irina Ciobanu. 2008, 2014, 2021

© Întreprinderea Editorial-Poligrafică *Știința*. 2008, 2014, 2021

ОГЛАВЛЕНИЕ

Содержание	Гуманитарный	Реальный	Страница
Введение			5
1. СОСТАВНЫЕ ТИПЫ ДАННЫХ			
1.1. Простые и составные типы данных	•	•	6
1.2. Тип данных <i>массив</i>	•	•	10
1.3. Типы данных <i>строка символов</i>	•	•	26
1.4. Типы данных <i>запись</i>	•	•	40
1.5. Оператор with	•	•	49
1.6. Типы данных <i>множество</i>	•	•	52
1.7. Общие сведения о файлах	•	•	62
1.8. Файлы с последовательным доступом	•	•	71
1.9. Текстовые файлы	•	•	78
2. ИНФОРМАЦИЯ			
2.1. Количество информации	•	•	92
2.2. Кодирование и декодирование информации	•	•	95
2.3. Часто используемые коды	•	•	97
2.4. Информация непрерывных сообщений		•	103
2.5. Квантование изображений		•	106
2.6. Представление и передача информации		•	109
3. АРИФМЕТИЧЕСКИЕ ОСНОВЫ ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ			
3.1. Системы счисления		•	114
3.2. Перевод чисел из одной системы счисления в другую		•	117
3.3. Перевод чисел из двоичной системы счисления в восьмеричную, шестнадцатеричную и обратно		•	119
3.4. Арифметические операции в двоичной системе счисления		•	122
3.5. Представление натуральных чисел в компьютере		•	124
3.6. Представление целых чисел		•	125
3.7. Представление вещественных чисел		•	128
4. БУЛЕВА АЛГЕБРА			
4.1. Логические переменные и выражения		•	133
4.2. Логические функции		•	137
4.3. Часто используемые логические функции		•	139

Содержание	Гуманитарный	Реальный	Страница
5. ЛОГИЧЕСКИЕ СХЕМЫ			
5.1. Логические элементы		•	142
5.2. Классификация логических схем		•	147
5.3. Сумматор		•	147
5.4. Часто используемые комбинационные схемы		•	151
5.5. <i>RS</i> -триггер		•	154
5.6. Часто используемые последовательностные схемы		•	157
5.7. Генераторы импульсов		•	160
6. УСТРОЙСТВО И РАБОТА КОМПЬЮТЕРА			
6.1. Функциональная схема компьютера	•	•	163
6.2. Форматы команд		•	165
6.3. Типы команд		•	168
6.4. Машинный язык и язык ассемблера		•	170
6.5. Аппаратные и программные ресурсы компьютера	•	•	172
6.6. Внешняя память на магнитных лентах и дисках	•	•	174
6.7. Внешняя память на оптических дисках	•	•	178
6.8. Видеомонитор и клавиатура	•	•	182
6.9. Принтеры	•	•	184
6.10. Классификация компьютеров	•	•	187
6.11. Микропроцессор		•	189
7. КОМПЬЮТЕРНЫЕ СЕТИ			
7.1. Введение в компьютерные сети	•	•	192
7.2. Технологии взаимодействия в компьютерной сети	•	•	195
7.3. Топология и архитектура компьютерных сетей		•	197
7.4. Глобальная сеть ИНТЕРНЕТ	•	•	201
7.5. Сервисы ИНТЕРНЕТа	•	•	206
8. МОДУЛИ ПО ВЫБОРУ			
8.1. Техника обработки аудио-видео информации	•	•	211
8.2. Визуальное программирование	•	•	216
8.3. Языки разметки гипертекста	•	•	221

Дорогие друзья!

Впечатляющие достижения в области информатики, создание суперкомпьютеров и персональных компьютеров, появление киберпространства, вошедший в нашу повседневную жизнь Интернет, расширивший возможности нашей реальности, – все это требует от нас глубокого знания, досконального изучения принципов работы устройства современных компьютеров, методов их программирования. Независимо от специфики будущей профессиональной деятельности каждого выпускника лицея, эти знания будут служить вам достойным ориентиром в цифровом мире, который постоянно меняется.

Непрерывное развитие информатики как фундаментальной науки и стремительное развитие информационных и коммуникационных технологий как прикладных наук приводят к появлению все нового и нового цифрового оборудования, которое практически полностью обновляется каждые два-три года. «Вторжение» гаджетов, растущее разнообразие программных продуктов, появление интернета вещей – все это требует пересмотра методов изучения информатики, переориентации их с формирования простой ловкости нажатия на кнопки, нажатия и перетаскивания объектов на сенсорных экранах на обучение навыкам использования нового цифрового оборудования и создания цифровых продуктов, таких как прикладные программы, веб-документы, цифровые онлайн-сервисы, цифровые произведения искусства.

В современном мире самые впечатляющие инновации возникают на стыке различных областей знаний. В качестве примеров можно привести оцифровку радио и телевидения, музыки и изобразительного искусства, расширение онлайн-торговли, возрастающую роль социальных сетей и усиление позиций обучения с использованием компьютера. Появление и развитие искусственного интеллекта, который, по прогнозам многих выдающихся личностей современности, существенно изменит эволюцию человеческой цивилизации, предполагает объединение усилий специалистов в различных областях, независимо от их природы.

Таким образом, независимо от профиля лицея, гуманитарного или реального, его выпускникам для достижения целей личностного развития, включая успешную карьеру и полноценную жизнь, потребуются цифровые компетенции, формирование и развитие которых является целью данного учебника. В частности, по окончании 11-го класса ученики смогут:

- разрабатывать на одном из языков программирования высокого уровня алгоритмы обработки структурированных данных;
- применять основные понятия теории информации, компьютерной арифметики и булевой алгебры для создания информационных моделей;
- интерпретировать результаты выполнения разработанных компьютерных программ;
- идентифицировать общую структуру применяемых цифровых систем, принципы работы систем передачи, накопления и обработки информации.

Чтобы реализовать свое призвание, впервые в истории компьютерного образования в Республике Молдова ученики могут выбрать для изучения один из трех модулей по выбору. Процесс обучения по этим модулям существенно отличается от традиционного, основная роль отводится не теоретическим занятиям, а практическим, основанным на исследовании упражнений с использованием средств компьютерного обучения, разработке проектов.

Изучение одного из модулей по выбору поможет вам применять методы и инструменты, специфичные для цифровой обработки, поможет развить мышление и творческий потенциал, интегрировать знания и навыки из области информатики со знаниями и навыками в других областях, повысит осознание важности соблюдения правил безопасности, эргономики и этики при создании и распространении цифровых продуктов.

С целью самооценки рекомендуем учащимся скачать с веб-страницы <http://www.ctice.gov.md> Центра Информационных и Коммуникационных Технологий в Образовании соответствующие тесты, выполнить предлагаемые в них задания и сравнить свои результаты с правильными ответами, которые можно узнать у преподавателя информатики.

Желаем успехов!

Авторы

Глава 1

СОСТАВНЫЕ ТИПЫ ДАННЫХ

1.1. Простые и составные типы данных

Мы уже знаем, что информация, которую может обрабатывать компьютер, должна быть представлена в виде данных. Данные состоят из цифр, букв, знаков, чисел, строк символов и т. д.

В машинном коде данные представляются в виде последовательностей двоичных цифр. Например, на уровне процессора натуральное число 1039 в двоичной системе счисления представляется как:

```
10000001111
```

Чтобы избавить пользователя от всех деталей, связанных с внутренним представлением данных, языки программирования используют различные типы данных. Напомним, что тип данных означает множество значений и множество операций, которые могут быть выполнены с этими значениями.

В предыдущих классах вы изучили следующие типы данных:

- `integer`/**`int`**, предназначенный для компьютерной обработки целых чисел;
- `real`/**`float`**, предназначенный для обработки вещественных чисел;
- `boolean`/**`bool`** используется при обработке значений истинности;
- `char`/**`char`**, предназначенный для представления и обработки символов;
- *перечисляемый* (**`enum`**), который включает упорядоченный набор значений, определенных идентификаторами;
- *интервальный* (только на языке ПАСКАЛЬ), который включает подмножество значений типа `integer`, `boolean`, `char` или *перечисляемого*.

С точки зрения языков программирования, целые числа, вещественные числа, значения истинности, символы, упорядоченные значения, заданные идентификаторами, называются **простыми данными**, а вышеперечисленные типы – **простыми типами данных**.

В общем, простых данных недостаточно для представления и эффективной обработки с помощью компьютера информации из окружающего мира.

Например, в случае обработки текстов операции удаления, копирования и перемещения выполняются не только на уровне символов (простой тип данных `char`), но также на уровне слов, предложений, строк, абзацев и даже целых страниц. Очевидно, что слово или предложение можно представить как последовательность символов, то есть через структуру, состоящую из последовательности простых данных типа `char`. Для описания таких структур языки программирования высокого уровня используют типы данных, называемые

строками символов. Следовательно, буквы латинского алфавита A, a, B, b, C, c, ..., Z, z представлены на компьютере простыми данными типа char, а слова, образованные из них, например Liceu, Patrie, Informatica и т. д. — с помощью составных данных типа строка символов.

Другой пример, иллюстрирующий необходимость использования составных данных, — это обработка информации о ежедневном потреблении электроэнергии в домашнем хозяйстве. Объем потребления, в кВт·ч, может быть представлен на компьютере вещественным числом (простой тип данных real или float). Однако если мы хотим проанализировать изменения суточного потребления электроэнергии в течение месяца, нам придется представить соответствующие данные в виде одномерной таблицы, состоящей из одной строки:

День месяца	1	2	3	...	31
Ежедневное потребление, кВт·ч	13,4	18,6	9,4	...	15,0

В первой ячейке этой таблицы будет содержаться потребление, в кВт·ч, в первый день месяца; во второй ячейке — потребление во второй день месяца и так далее до ячейки 31, в зависимости от количества дней в рассматриваемом месяце.

При анализе суточного потребления электроэнергии в течение года нам придется использовать таблицу, состоящую из 12 строк и 31 столбца, то есть двумерную таблицу:

День месяца	1	2	3	...	31
Январь	13,4	18,6	9,4	...	15,0
Февраль	12,9	14,3	21,7
...
Декабрь	11,7	10,4	3,4	...	25,3

Первая строка такой таблицы будет содержать потребление для каждого из дней января; вторая строка — потребление для каждого дня февраля и так далее до строки, соответствующей декабрю.

Можно заметить, что приведенные выше таблицы представляют собой составные данные, состоящие из простых данных типа real / float. Обычно на компьютере соответствующие таблицы представляются с помощью специальных типов составных данных, называемых массивами.

Подчеркнем, что массивы содержат только данные в ячейках, без имен строк и столбцов. На компьютере эти имена представлены другими величинами, называемыми индексами. Очевидно, что в случае одномерных массивов требуется только один индекс, а в случае двумерных массивов — два. Например, в случае одномерного массива, который представляет на компьютере ежедневное потребление электроэнергии в течение месяца, в качестве индекса используется целая величина, которая может принимать только значения 1, 2, 3, ..., 31.

В случае двумерного массива, который представляет ежедневное потребление электроэнергии в течение года, необходимы два индекса: один для строк и один для столбцов. Перечисляемый тип, который принимает значения Ianuarie, Februarie, Martie, ..., Decembrie, используется в качестве

индекса для строк. Для индекса столбца используется целочисленная величина, которая принимает значения 1, 2, 3, ... 31.

Данные, сформированные путем агрегирования (объединения в единое целое) простых данных, называются *составными данными*. Типы данных, используемые для определения таких данных, называются *составными типами данных*.

В дополнение к *строкам символов* и *массивам*, вышеописанным в общих чертах, языки программирования предлагают специалистам возможность использовать и другие составные типы данных, наиболее часто используемые, — *записи, множества и файлы*.

Вопросы и упражнения

- ❶ Объясните значения термина *тип данных*. Приведите примеры.
- ❷ В чем отличие между простыми и составными типами данных?
- ❸ Приведите примеры простых типов данных и составных типов данных.
- ❹ ОБРАТИТЕ ВНИМАНИЕ! Определите, какой тип, простой или составной, имеют данные в нижеследующих примерах:

a)

138

b)

3.14

c)

Munteanu Elena

d)

Clasa a 11-a

e)

12	6	231	5
----	---	-----	---

f)

32.51	149,28	318,56	20013.9	0.4536	721.3
-------	--------	--------	---------	--------	-------

g)

4	635	-8	+27
72	41	319	432
16	-20	45	1830

В случае составных данных укажите тип простых данных, входящих в их состав.

Примеры ответов:

— Простое данное типа *целое число*.

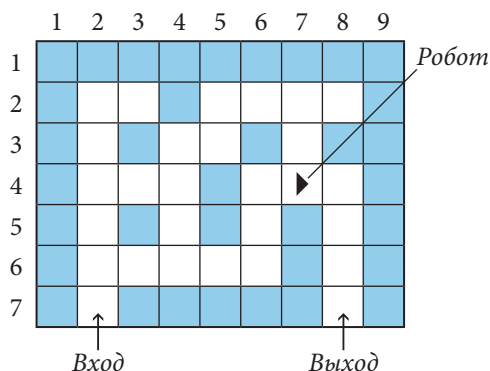
— Составное данное типа *одномерный массив*, состоящий из *целых чисел*.

- ❺ ТВОРИТЕ! Составьте одномерную таблицу, содержащую ежемесячные платежи за электроэнергию, потребленную вашей семьей в течение последнего календарного года. Используйте для этой цели информацию в счетах на оплату, выставленных компанией-поставщиком электроэнергии. Какие типы данных вы будете применять для представления информации в этой таблице на компьютере?
- ❻ ИССЛЕДУЙТЕ! На сайте Национального банка Молдовы размещена информация о средних официальных курсах валютного обмена (обменном курсе). Например, в январе 2019 года средний официальный обменный курс единой валюты Европейского союза составлял: 1 евро = 19,6501 молдавских леев. Составьте: — одномерную таблицу, содержащую официальные среднемесячные курсы этой валюты за последний календарный год;

– двумерную таблицу, содержащую официальные среднемесячные обменные курсы этой валюты за последние три года.

Какие типы данных вы будете использовать для представления информации в этих таблицах на компьютере?

- 7 ТВОРИТЕ! После посещения знаменитых Криковских подвалов, во время которого ученый-информатик имел возможность путешествовать по подземным галереям этого важного туристического объекта, ученый сконструировал робота, который находит кратчайший путь из одного выставочного зала в другой. В памяти компьютера, управляющего действиями робота, план галерей представлен прямоугольником, разделенным на квадраты (см. рисунок ниже).



Закрашенные квадраты представляют препятствия (стены галерей и выставочных залов, стенды с экспонатами), не закрашенные — свободные пространства. Робот может выполнять только команды ВВЕРХ, ВНИЗ, НАПРАВО, НАЛЕВО, согласно которым он перемещается в один из соседних квадратов. Если в этом квадрате есть препятствие, например стена или стенд, робот останавливается и не реагирует ни на какие команды.

Разработайте структуру данных, необходимую для представления плана Криковских подвалов на компьютере. Убедитесь, что продуманная структура содержит всю информацию о плане: препятствия, свободные пространства, квадраты, в которых находятся начальное положение робота, вход и выход.

- 8 УЧИТЕСЬ УЧИТЬСЯ! Вы уже знаете, что составные данные получаются путем агрегирования (объединения в единое целое) простых данных. Языки ПАСКАЛЬ и С++ позволяют агрегировать не только простые данные, но и составные. Например, в ячейках таблицы можно хранить не только простые данные (целые числа, действительные числа, символы и т. д.), но и составные данные, такие как строки.

Дан список учеников лицейского класса:

Мунтяну Ион
Флоря Елена
...
Присэкару Александра

Разработайте структуру данных, необходимую для представления такого списка на компьютере. Убедитесь, что продуманная структура позволяет сортировать список учащихся по алфавиту.

1.2. Тип данных массив

PASCAL

Для определения типа данных *массив* в языке ПАСКАЛЬ используется ключевое слово **array**. Массивы состояются из фиксированного числа компонентов одного и того же типа, который называется **базовым**. Ссылка на компоненты осуществляется с помощью одного или нескольких **индексов**.

Тип данных *одномерный массив* определяется конструкцией вида:

type <Имя типа> = **array** [T_1] **of** T_2 ;

где T_1 – тип индекса, который должен быть порядковым, а T_2 – тип компоненты (базовый тип), который может быть любым.

Примеры:

- 1) **type** Vector = **array** [1..5] **of** real;
var x : Vector;
- 2) **type** Zi = (L, Ma, Mi, J, V, S, D);
Venit = **array** [Zi] **of** real;
var v : Venit;
z : Zi;
- 3) **type** Ora = 0..23;
Grade = -40..40;
Temperatura = **array** [Ora] **of** Grade;
var t : Temperatura;
h : Ora;

Структура данных, используемых в этих примерах, представлена на рис. 1.1.

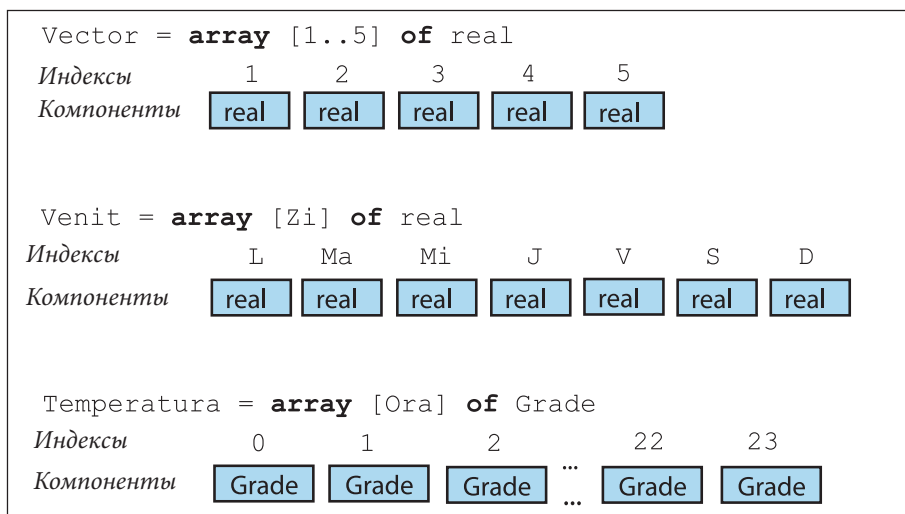


Рис. 1.1. Структура данных типа Vector, Venit и Temperatura

Доступ к компонентам переменной типа *одномерный массив* осуществляется явно через имя переменной, за которой следует соответствующий индекс, заключенный в квадратные скобки.

Примеры:

- 1) `x[1], x[4];`
- 2) `v[L], v[Ma], v[J];`
- 3) `t[0], t[15], t[23];`
- 4) `v[z], t[h].`

К компонентам данных типа *массив* можно применять все операции, допустимые для соответствующего базового типа. Следующая программа выводит на экран сумму компонентов переменной *x* типа *Vector*. Значения переменных *x[1]*, *x[2]*, ..., *x[5]* вводятся с клавиатуры.

```
Программа P78;  
{ Сумма компонентов переменной x типа Vector }  
{ Фиксированное количество компонентов  
type Vector = array [1..5] of real;  
var x : Vector;  
    i : integer;  
    s : real;  
begin  
  writeln('Введите 5 чисел:');  
  for i:=1 to 5 do readln(x[i]);  
  writeln('БЫЛИ введены:');  
  for i:=1 to 5 do writeln(x[i]);  
  s:=0;  
  for i:=1 to 5 do s:=s+x[i];  
  writeln('Сумма=', s);  
  readln;  
end.
```

Для того чтобы расширить область применения программы, количество компонентов данных типа **array** рекомендуется указывать через константы.

Например, программу P78 можно изменить таким образом, чтобы она вычисляла сумму *n* вещественных чисел, $n \leq 100$:

```
Программа P79;  
{ Сумма компонентов переменной x типа Vector }  
{ Переменное количество компонентов }  
const nmax = 100;  
type Vector = array [1..nmax] of real;  
var x : Vector;  
    n : 1..nmax;
```

```

    i : integer;
    s : real;
begin
    write('n='); readln(n);
    writeln('Введите ', n, ' чисел:');
    for i:=1 to n do readln(x[i]);
    writeln('Были введены:');
    for i:=1 to n do writeln(x[i]);
    s:=0;
    for i:=1 to n do
        s:=s+x[i];
    writeln('Сумма=', s);
    readln;
end.

```

В информатике одномерные массивы часто используются для сортировки данных в определенном порядке, например в порядке возрастания. Для такой сортировки используется следующий метод:

- 1) каждый компонент массива, начиная с первого, последовательно сравнивается с каждым из следующих за ним компонентов;
- 2) если в результате сравнения компонентов выясняется, что текущий компонент больше того, с которым он сравнивается, компоненты меняются местами;
- 3) процесс обхода массива продолжается до сравнения последнего и предпоследнего компонентов.

На быденном языке информатиков такая сортировка называется пузырьковой сортировкой, потому что компоненты массива меняют свое положение, как пузырьки в жидкости: самые легкие поднимаются на поверхность, а самые тяжелые уходят на дно.

Например, в следующей программе пузырьковый метод используется для сортировки компонентов массива А в порядке возрастания. Компоненты этого массива считываются с клавиатуры, а фактическая сортировка выполняется в таблице В.

```

Программа P80;
{ Сортировка пузырьковым методом }
const nmax=100;
type Tablou = array[1..nmax] of integer;
var A, B : Tablou;
    n, i, j : integer;
    x : integer;
begin
    write('Введите количество компонентов n= ');
    readln(n);
    writeln('Введите компоненты массива A:');
    for i:=1 to n do read(A[i]);
    readln;

```

```

B:=A;
for i:=1 to n-1 do
  for j:=i+1 to n do
    if (B[i]>B[j]) then
      begin
        x:=B[i];
        B[i]:=B[j];
        B[j]:=x;
      end;
writeln('Начальный массив  A:');
for i:=1 to n do write(A[i], ' ');
writeln;
writeln('Отсортированный массив  B:');
for i:=1 to n do write(B[i], ' ');
writeln;
readln;
end.

```

Тип данных *двумерный массив* определяется с помощью конструкции вида:

```
type <Имя типа> = array [ $T_1, T_2$ ] of  $T_3$ ;
```

где T_1 и T_2 указывают тип индексов, а T_3 — тип компонентов.

В качестве примера на *рис. 1.2* представлена структура данных Matrice:

```
type Matrice = array [1..3, 1..4] of real;
```

	Matrice = array [1..3, 1..4] of real			
	1	2	3	4
1	real	real	real	real
2	real	real	real	real
3	real	real	real	real

Рис. 1.2. Структура данных типа Matrice

Доступ к компонентам переменной типа *двумерный массив* осуществляется явно через имя переменной, за которой следуют соответствующие индексы, разделенные запятой и заключенные в квадратные скобки.

Например, при описании

```
var m : Matrice;
```

обозначение $M[1, 1]$ означает ссылку на компонент, расположенный в строке 1 и в столбце 1 (см. *рис. 1.2*); обозначение $M[1, 2]$ означает ссылку на компонент, расположенный в строке 1 и в столбце 2; обозначение $M[i, j]$ означает ссылку на компонент, расположенный в строке i и столбце j .

Следующая программа выводит на экран сумму компонентов переменной *M* типа *Matrice*. Значения компонентов *M*[1,1], *M*[1,2], ..., *M*[3,4] вводятся с клавиатуры.

```
Программа P81;  
{ Сумма компонентов переменной M типа Matrice }  
type Matrice = array [1..3, 1..4] of real;  
var M : Matrice;  
    i, j : integer;  
    s : real;  
begin  
  writeln('Введите компоненты M[i,j]:');  
  for i:=1 to 3 do  
    for j:=1 to 4 do  
      begin  
        write('M[', i, ', ', j, ']=');  
        readln(M[i,j]);  
      end;  
  writeln('Были введены:');  
  for i:=1 to 3 do  
    begin  
      for j:=1 to 4 do write(M[i,j]);  
      writeln;  
    end;  
  S:=0;  
  for i:=1 to 3 do  
    for j:=1 to 4 do  
      S:=S+M[i,j];  
  writeln('Сумма=', S);  
  readln;  
end.
```

В общем виде тип *n*-мерный массив (*n* = 1, 2, 3 и т. д.) определяется с помощью синтаксических диаграмм, приведенных на *рис. 1.3*. Слово **packed** (упакованный) указывает компилятору, что область памяти для элементов типа **array** должна быть выделена с применением оптимизации. Отметим, что в большинстве компьютеров использование этого префикса не обязательно, так как оптимизация осуществляется автоматически.

Если даны две переменные типа *массив* одного и того же базового типа, то имена этих переменных могут встречаться в операциях присваивания. Такое присваивание означает копирование всех компонентов массива, расположенного в правой части, в массив, расположенный в левой части.

Например, при описании

```
var a, b : Matrice;
```

оператор

```
a:=b
```

является правильным.

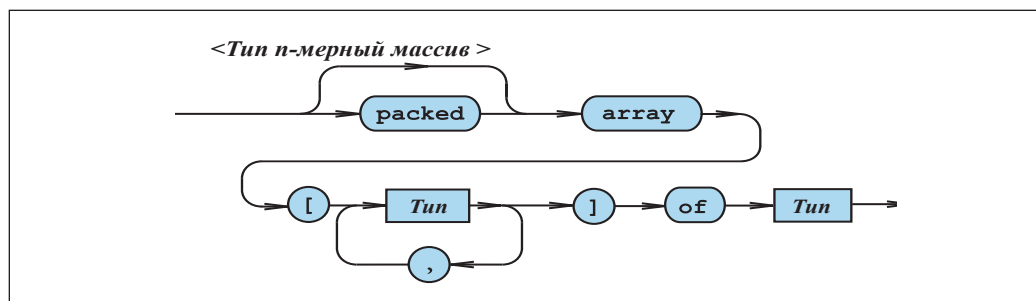


Рис. 1.3. Синтаксическая диаграмма <Тип n-мерный массив>

В примерах, рассмотренных выше, базовый тип (тип компонентов) всегда был простым. Так как базовый тип в большинстве случаев может быть любым, то не исключена возможность определения массивов, компоненты которых относятся к структурированному типу. Рассмотрим пример, в котором базовым типом является сам тип **array**:

```
Type Linie = array [1..4] of real;
      Tabel = array [1..3] of Linie;
var L : Linie;
    T : Tabel;
    x : real;
```

Переменная T состоит из 3-х компонентов: T[1], T[2] и T[3] типа Linie. Переменная L состоит из 4-х компонентов: L[1], L[2], L[3] и L[4] типа real.

Следовательно, операторы присваивания

```
L[1]:=x; x:=L[3]; T[2]:=L; L:=T[1]
```

являются правильными.

Доступ к элементам переменной T может осуществляться через T[i][j] или T[i, j]. Здесь индекс i обозначает номер компонента типа Linie переменной T, а j — номер компонента типа real компонента T[i] типа Linie.

Отметим, что объявления вида

```
type array [T1, T2] of T3
```

и

```
type array [T1] of array [T2] of T3
```

определяют различные типы данных. Первое объявление определяет двумерные массивы с компонентами типа T₃. Второе — определяет одномерные массивы с компонентами типа **array** [T₂] **of** T₃.

В программах на языке ПАСКАЛЬ массивы используются для группировки под одним именем нескольких переменных, обладающих одинаковыми характеристиками.

C++

В языке программирования C++ тип данных *одномерный массив* определяется конструкцией вида:

typedef <Тип компонентов> <Имя типа массив> [<Количество компонентов>];

где <Тип компонентов> – это тип компонентов массива, который может быть практически любым типом данных, а <Количество компонентов> указывает количество компонентов массива.

Индексы компонентов массива могут принимать только последовательные целочисленные значения, начиная с нуля: 0, 1, 2,..., <Количество компонентов> - 1.

Примеры:

- 1) **typedef int** Vector[5];
Vector V;
- 2) **typedef char** Simbol[10];
Simbol S;
- 3) **typedef float** Acceleratie[45];
Acceleratie A;

Структура данных из рассмотренных примеров представлена на *рисунке 1.1**.

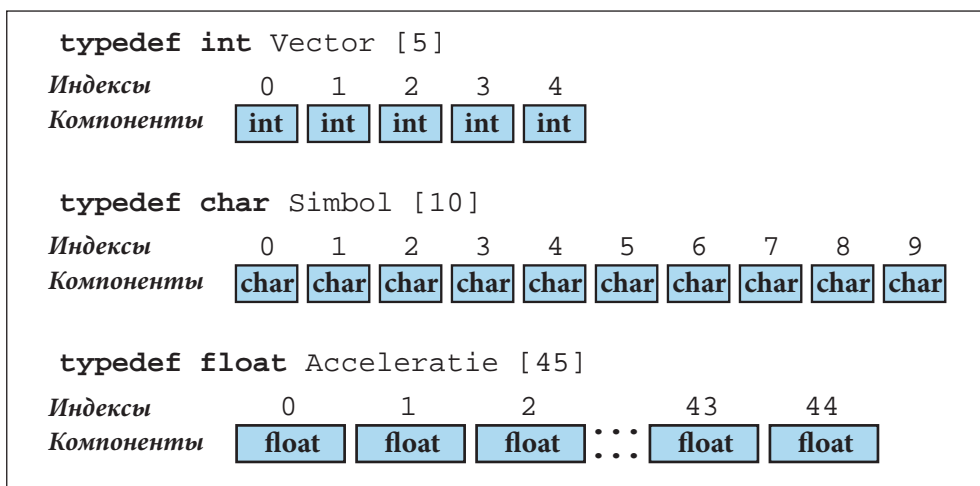


Рис. 1.1* Структура данных типа Vector, Simbol и Acceleratie

Каждый компонент переменной одномерного массива можно указать явно, определив имя переменной, за которым следует соответствующий индекс, заключенный в квадратные скобки. Индексы должны быть целочисленными выражениями.

Примеры:

- 1) V[1], V[4];
- 2) S[0], S[5], S[9];
- 3) A[0], A[44], A[23];

Над компонентами данных одномерного массива могут выполняться все операции, допустимые над их базовым типом. Следующая программа отображает на экране сумму компонентов переменной одномерного массива V . Значения компонентов $V[0]$, $V[1]$, ..., $V[4]$ считываются с клавиатуры.

```
// Программа P78
// Сумма компонентов переменной V типа Vector
// Фиксированное количество компонентов
#include <iostream>
using namespace std;
int main()
{
    typedef int Vector[5];
    Vector V;
    int i, S;
    cout << "Введите 5 целых чисел:" << endl;
    for (i=0; i<5; i++) cin>>V[i];
    cout << "Были введены:" << endl;
    for (i=0; i<5; i++) cout << V[i] << ' ';
    cout << endl;
    S=0;
    for (i=0; i<5; i++) S=S+V[i];
    cout << "Сумма= " << S;
    return 0;
}
```

Чтобы расширить область действия программы, рекомендуется указывать количество компонентов данных типа *массив* через константы. Это дает большую гибкость программам на C++, поскольку они могут применяться для обработки массивов, количество компонентов которых на момент написания программы неизвестно.

Например, программа P78 может быть изменена таким образом, чтобы она вычисляла сумму n целых чисел, $n \leq 100$. Конкретное количество значений n , которые будут сохранены в одномерном массиве V , считывается с клавиатуры.

```
//Программа P79
// Сумма компонентов переменной V типа Vector
// Переменное количество компонентов
#include <iostream>
using namespace std;
int main()
{
    const int nmax=100;
    typedef int Vector[nmax];
    Vector V;
    int n, i, S;
```

```

cout << "Введите n= "; cin >> n;
cout << "Введите "<n<<" целых чисел:" << endl;
for (i=0; i<n; i++) cin >> V[i];
cout << "Были введены:" << endl;
for (i=0; i<n; i++) cout << V[i] << ' ';
cout << endl;
S=0;
for (i=0; i<n; i++) S=S+V[i];
cout << "Сумма= " << S;
return 0;
}

```

В информатике одномерные массивы часто используются для сортировки данных в определенном порядке, например в порядке возрастания. Для такой сортировки используется следующий метод:

- 1) каждый компонент массива, начиная с первого, последовательно сравнивается с каждым из следующих за ним компонентов;
- 2) если в результате сравнения компонентов выясняется, что текущий компонент больше того, с которым он сравнивается, компоненты меняются местами;
- 3) процесс обхода массива продолжается до сравнения последнего и предпоследнего компонентов.

На быденном языке информатиков такая сортировка называется *пузырьковой сортировкой*, потому что компоненты массива меняют свое положение, как пузырьки в жидкости: самые легкие поднимаются на поверхность, а самые тяжелые уходят на дно.

Например, в следующей программе пузырьковый метод используется для сортировки компонентов массива A в порядке возрастания. Компоненты этого массива считываются с клавиатуры, а фактическая сортировка выполняется в таблице B.

```

// Программа P80
// Сортировка пузырьковым методом
#include <iostream>
using namespace std;
int main()
{
    const int nmax= 100;
    typedef int Tablou[nmax];
    Tablou A, B;
    int n, i, j;
    int x;
    cout<<"Введите количество компонентов n= ";
    cin>>n;
    cout<<"Введите компоненты массивы A: \n";
    for (i=0; i<n; i++) cin>>A[i];

```

```

for (i=0; i<n; i++) B[i]=A[i];
for (i=0; i<n-1; i++)
    for (j=i+1; j<n; j++)
        if (B[i]>B[j])
            { x=B[i]; B[i]=B[j]; B[j]=x;}
cout<<"Исходный массив: \n";
for (i=0; i<n; i++) cout<<A[i]<<" ";
cout << endl;
cout<<"Отсортированный массив: \n";
for (i=0; i<n; i++) cout<<B[i]<<" ";
cout << endl;
return 0;
}

```

Тип данных *двумерный массив* определяется грамматической конструкцией вида:

typedef <Тип компонентов> <Имя типа массив> [<Количество строк>] [<Количество столбцов>];

В качестве примера на *рис. 1.2** представлена структура данных типа Matrice:

```
typedef double Matrice[3][4];
```

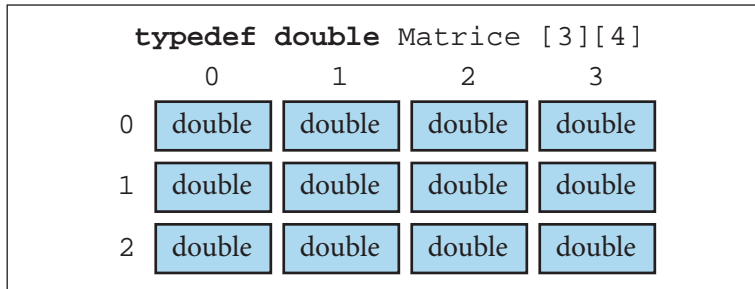


Рис. 1.2 *. Структура данных типа Matrice

Компоненты переменных типа *двумерный массив* явно указываются именем переменной, за которым следуют соответствующие индексы в квадратных скобках.

Например, при наличии объявления переменной M

```
Matrice M;
```

обозначение M[1, 1] указывает компонент из строки с номером 1 и столбца с номером 1 (см. *рис. 1.2**); обозначение M[1, 2] указывает компонент из строки 1 и столбца 2; обозначение M[i, j] указывает компонент из строки i и столбца j.

Следующая программа отображает на экране сумму компонентов переменной M типа Matrice. Значения компонентов M[0, 0], M[0, 1], ..., M[2, 3] считываются с клавиатуры.

```

// Программа P81
// Сумма компонентов переменной M типа Matrice
#include<iostream>
using namespace std;
int main()
{
    typedef double Matrice[3][4];
    Matrice M;
    int i, j;
    double S;
    cout << "Введите компоненты M[i,j]: " << endl;
    for (i=0; i<3; i++)
        for (j=0; j<4; j++)
        {
            cout << "M[" <<i<< ', ' << j << "]= ";
            cin >> M[i][j];
        }
    cout << "Были введены:" << endl;
    for (i=0; i<3; i++)
    {
        for (j=0; j<4; j++) cout << M[i][j] << ' ';
        cout << endl;
    }
    S=0;
    for (i=0; i<3; i++)
        for (j=0; j<4; j++)
            S=S+M[i][j];
    cout << "Сумма= " << S << endl;
    return 0;
}

```

В момент объявления компоненты массива можно инициализировать, например:

```

int T[4]={2, 5, 9, -4};

int M[3][2]={{9, 7}, {-12, 8}, {0, -4}};

```

Подчеркнем, что в языке C++ индексы компонентов массива могут принимать только последовательные целочисленные значения, начиная с нуля: 0, 1, 2, ..., <Количество компонентов> – 1. Иногда это может создавать определенные неудобства.

Таким образом, в случае примера суточного потребления электроэнергии в течение месяца (см. предыдущий параграф) соответствующая таблица может быть представлена на компьютере с помощью следующего типа данных *одномерный массив*:

```

typedef float Consum[31];
Consum C;

```

Индекс массива C может принимать только значения 0, 1, 2, ... 30, а компоненты массива специфицируются как C[0], C[1], ..., C[30]. Очевидно,

что индекс последнего дня месяца, т. е. 31-го не будет соответствовать какому-либо компоненту массива, а попытка использовать спецификацию `C[31]` в определенных ситуациях может вызвать ошибку выполнения. Несомненно, мы могли бы пронумеровать дни месяца не начиная с «1», а с «0», но это может сделать программы на C++ менее интуитивно понятными.

Чтобы избежать таких ситуаций, рекомендуется увеличить количество компонентов массива на одну единицу по сравнению с количеством, требуемым для представления составных данных на компьютере.

Например, в случае ежедневного потребления электроэнергии в течение месяца мы можем использовать следующий тип данных:

```
typedef float Consum[32];  
Consum C;
```

где первый день месяца будет соответствовать компоненту `C[1]`, второй день — компоненту `C[2]`, третий день — компоненту `C[3]` и так далее, а компонент `C[0]` останется неиспользованным.

Аналогичным образом можно использовать следующие типы данных для представления в компьютере ежедневного потребления электроэнергии в течение года:

```
typedef float Consum[13][32];  
Consum C;
```

где дни *января* будут соответствовать компонентам `C[1,1], ..., C[1,31]`; дни *февраля* — компонентам `C[2,1], ..., C[2,31]`, дни *декабря* — компонентам `C[12,1], ..., C[12,31]`. Очевидно, что некоторые компоненты таблицы `C`, особенно в строке и столбце 0, останутся неиспользованными. Однако поскольку читабельность и правильность программ гораздо важнее, чем неиспользование определенных компонентов, большинство информатиков считают, что этот прием не ухудшает качество программ на C++.

Вопросы и упражнения

- ❶ ОБРАТИТЕ ВНИМАНИЕ! Определите тип индексов и тип компонент в следующих объявлениях:

ПАСКАЛЬ

```
type  
P = array [1..5] of integer;  
Culoare = (Galben, Verde,  
Albastru, Violet);  
R = array [Culoare] of real;  
S = array [Culoare, 1..3]  
of boolean;  
T = array [boolean]  
of Culoare;
```

C++

```
typedef int P[5];  
enum Culoare {Galben, Verde,  
Albastru, Violet};  
typedef Culoare R[3];  
typedef bool S[4][3];  
typedef double T[2][5];
```

Нарисуйте структуры данных типа `P`, `R`, `S` и `T` (см. вышеприведенные соответствующие рисунки).

- ❷ (ПАСКАЛЬ) Укажите на синтаксической диаграмме *рис. 1.3* пути, которые соответствуют объявлениям из упражнения 1.

- ③ (ПАСКАЛЬ) Напишите металингвистические формулы, которые соответствуют синтаксической диаграмме *<Тип массив>* на рис. 1.3.
- ④ ПРОАНАЛИЗИРУЙТЕ! Определите, какие из нижеприведенных объявлений корректны:

ПАСКАЛЬ	С++
a: array [1..30] of integer;	int a[30];
a: array [1..10] of byte;	char a[1..10];
a: array [1.10..10.00] of real;	float a[1.10..10.00];
a: array [1..50] of char;	char a[50];

- ⑤ УПРАЖНЯЙТЕСЬ! Даны объявления:

ПАСКАЛЬ	С++
type Vector = array [1..5] of real; var x, y : Vector;	typedef double Vector[5]; Vector x,y;

Напишите арифметическое выражение, значением которого является:

- сумма первых трех компонентов переменной x;
- сумма всех компонентов переменной y;
- произведение всех компонентов переменной x;
- абсолютное значение третьего компонента переменной y;
- сумма первых компонентов переменных x и y.

- ⑥ ПРОАНАЛИЗИРУЙТЕ! Даны описания и операторы:

Объявления	Оператор	Результат
ПАСКАЛЬ		
1) var a: array [1..5] of byte;	for i:=1 to 5 do a[i]:=i-1;	a) 1 1 1 1 1 b) 0 0 0 0 0 c) 0 1 2 3 4 d) 1 2 3 4 0
2) var v: array [0..4] of integer;	for i:=1 to 5 do v[i]:=2*(i-1);	a) 2 4 6 8 1 b) 0 2 4 6 8 c) 0 1 2 3 4 d) 1 2 3 4 5

С++		
1) unsigned char a[4];	for (i=0; i<5; i++) a[i]=i;	a) 1 1 1 1 1 b) 0 0 0 0 0 c) 0 1 2 3 4 d) 1 2 3 4 0
2) int v[4];	for (i=0; i<5; i++) v[i]=2*i;	a) 2 4 6 8 10 b) 0 2 4 6 8 c) 0 1 2 3 4 d) 1 2 3 4 5

Выберите из списков справа варианты, содержащие правильные результаты выполнения соответствующих операторов.

7 РЕШИТЕ! Даны объявления:

ПАСКАЛЬ

C++

```
type Zi = (L, Ma, Mi, J, Vi, S, D);
    Venit = array [Zi] of real;
var v : Venit;
```

```
typedef double Venit[7];
Venit v;
```

Компоненты переменной *v* представляют собой ежедневный доход предприятия. Напишите программу, которая:

- вычисляет еженедельный доход предприятия;
- подсчитывает средний ежедневный доход;
- определяет день, когда был получен наибольший доход;
- определяет день, когда был получен наименьший доход.

8 РЕШИТЕ! Даны объявления:

ПАСКАЛЬ

C++

```
type Ora = 0..23;
    Grade = -40..40;
    Temperatura = array [Ora]
of Grade;
var t : Temperatura;
```

```
typedef int Ora
typedef int Temperatura [24];
Temperatura t;
```

Компоненты переменной *t* представляют собой значения температуры, измеряемой каждый час в течение 24 часов. Напишите программу, которая:

- вычисляет среднюю температуру;
- определяет минимальное и максимальное значения температуры;
- определяет час (часы), в который была зарегистрирована максимальная температура;
- определяет час (часы), в который была зарегистрирована минимальная температура.
- вычисляет количество дней, в течение которых была зафиксирована температура ниже нуля градусов;
- вычисляет количество дней, в которые были зарегистрированы температуры выше средней за неделю.

9 Даны объявления:

ПАСКАЛЬ

C++

```
type Oras = (Chisinau, Orhei,
Balti, Tighina, Tiraspol);
    Zi=(L, Ma, Mi, J, Vi, S, D);
    Consum = array [Oras,Zi] of real;
var C : Consum;
    r : Oras;
    z : Zi;
```

```
enum Oras {Chisinau, Orhei,
    Balti, Tighina, Tiraspol};
enum Zi {L, Ma, Mi, J, Vi, S, D};
typedef double Consum[5][7];
Consum C;
Oras r;
Zi z;
```

Компонент $C[r, z] / C[r][z]$ в C++ переменной *C* представляет собой потребление электроэнергии города *r* в день *z*. Напишите программу, которая:

- вычисляет количество электроэнергии, потребляемой каждым городом за неделю;
- вычисляет количество электроэнергии, потребляемой данными городами ежедневно;
- определяет город с максимальным еженедельным потреблением электроэнергии;

- г) определяет город с минимальным еженедельным потреблением электроэнергии;
 д) определяет день, в который города потребили наибольшее количество электроэнергии;
 е) определяет день с наименьшим потреблением электроэнергии.

10 Даны объявления:

ПАСКАЛЬ

```
type Vector = array [1..5] of real;
Matrice = array [1..3, 1..4] of
boolean;
Linie = array [1..4] of real;
Tabel = array [1..3] of Linie;
var V : Vector;
    M : Matrice;
    L : Linie;
    T : Tabel;
    x : real;
    i : integer;
```

C++

```
typedef float Vector[5];
typedef bool Matrice[3][4];
typedef float Linie[4];
typedef Linie Tabel[3];
Vector V;
Matrice M;
Linie L;
Tabel T;
float x;
int i;
```

Какие из следующих операторов присваивания корректны?

ПАСКАЛЬ	C++
a) T[3]:=T[1]	a) T[3]=T[1]
b) M:=T	b) M=T
c) L:=V	c) L=V
d) L[3]:=x	d) L[3]=x
e) x:=i	e) x=i
f) i:=x	f) i=x
g) L[3]:=i	g) L[3]=i
h) i:=M[1,2]	h) i=M[1][2]
i) x:=V[4]	i) x=V[4]
j) L[3]:=V[4]	j) L[3]=V[4]
k) T[1]:=4	k) T[1]=4
l) T[2]:=V	l) T[2]=V
m) L:=T[3]	m) L=T[3]
n) T[1,2]:=M[1,2]	n) T[1][2]=M[1][2]
o) T[2,1]:=M[1,2]	o) T[2][1]=M[1][2]
p) M[1]:=4	p) M[1]=4
q) M[1,3]:=L[2]	q) M[1][3]=L[2]
r) x:=T[1][2]	r) x=T[1,2]
s) x:=M[1]	s) x=M[1]
t) L:=M[1]	t) L=M[1]
u) V[5]:=M[3,4]	u) V[5]=M[3][4]
v) L:=M[3,4]	v) L=M[3][4]

- ⑪ РЕШИТЕ! С клавиатуры считываются компоненты одномерного массива, состоящего из n целых чисел, $n \leq 10$. Напишите программу, которая:
- а) отображает на экране компоненты массива с интервалом в 5 позиций;
 - б) отображает на экране компоненты в порядке, обратном их введению в компьютер;
 - в) сортирует компоненты массива в порядке убывания;
 - г) отображает на экране только четные компоненты;
 - д) отображает на экране среднее арифметическое четных компонентов;
 - е) отображает на экране только нечетные компоненты;
 - ж) отображает на экране только те компоненты, которые больше x и не делятся на y (значения x и y считываются с клавиатуры);
 - з) отображает на экране только те компоненты, которые больше x и меньше y (значения x и y считываются с клавиатуры);
 - и) отображает на экране позиции (индексы) отрицательных нечетных компонентов;
 - к) отображает на экране позиции (индексы) компонентов, которые содержат только две значащие цифры;
 - л) заменяет первый компонент данного массива на компонент этого массива с минимальным значением;
 - м) заменяет компонент с минимальным значением данного массива на его первый компонент;
 - н) создает новый массив, состоящий только из четных компонентов массива, введенного с клавиатуры;
 - о) создает новый массив, состоящий только из делимых на 3 компонентов массива, введенного с клавиатуры.
 - п) создает новый массив, состоящий только из тех компонентов массива, вводимых с клавиатуры, которые имеют не более четырех делителей.
- ⑫ РЕШИТЕ! Человек намеревается купить n различных продуктов, пронумерованных $i = 1, 2, 3, \dots, n$. Каждый из этих продуктов можно купить в любом из k доступных магазинов, отмеченных $j = 1, 2, 3, \dots, k$. Очевидно, что покупатель выбирает магазины с самыми низкими ценами. Разработайте программу, которая определяет для каждого продукта i магазин j , в котором он должен быть приобретен, а также общую стоимость покупок C , сделанных таким образом.
Входные данные: двумерный массив P , в котором компонент $P[i][j]$ представляет цену продукта i в магазине j .
Выходные данные: общая стоимость покупок C и одномерный массив M , в котором компонент $M[i]$ представляет магазин j , в котором будет приобретен продукт i .
Ограничения: $1 \leq n \leq 10$, $1 \leq k \leq 5$. Цены указываются вещественными числами с двумя цифрами после десятичной точки.
- ⑬ ТЕМАТИЧЕСКОЕ ИССЛЕДОВАНИЕ. Разработайте структуры данных типа *массив*, необходимые для представления в программах ПАСКАЛЬ/С++ массивов, содержащих информацию о суточном потреблении электроэнергии в течение месяца и соответственно за год, описанных в этом параграфе. Проанализируйте, как количество компонентов в объявлениях массива влияет на понимание пользователями программ.
- ⑭ УЧИТЕСЬ УЧИТЬСЯ! Двумерные массивы с одинаковым количеством строк и столбцов называются *квадратными массивами*. Набор компонентов, у которых индекс строки равен индексу столбца, образует главную диагональ квадратного массива. Очевидно, что этот набор можно обозначить воображаемой линией, соединяющей компонент в верхнем левом углу с компонентом в правом нижнем углу квадратного массива. Аналогичным образом может

быть определена вторичная диагональ квадратного массива, которая также может быть обозначена воображаемой линией, соединяющей компонент в нижнем левом углу с компонентом в правом верхнем углу.

Разработайте программу, которая считывает с клавиатуры квадратный массив с n строками, $2 \leq n \leq 10$, и отображает на экране сумму компонентов, которые находятся:

- а) на главной диагонали;
- б) на вторичной диагонали;
- в) над главной диагональю;
- г) ниже главной диагонали;
- д) над вторичной диагональю;
- е) ниже вторичной диагонали.

Считается, что компоненты квадратного массива являются целыми числами, которые вводятся с клавиатуры.

- ❿ ИЗУЧИТЕ! Язык C++ предоставляет альтернативный тип двумерных массивов, определенных в библиотеке `<array>`. Изучите эту библиотеку и перепишите программы из этого параграфа, используя возможности, которые она предлагает. Чем эти два способа определения массивов похожи и различны?

1.3. Типы данных строка символов

Особое значение имеют методы представления и обработки строк символов, поскольку многие объекты реального мира описываются не только числами, но и самыми разнообразными текстами. Обычно в современных средах разработки программ строки символов могут быть представлены несколькими структурами данных. Далее мы изучим следующие типы данных, используемые для представления строк символов:

- одномерные массивы с компонентами типа `char` (символы);
- *string*-и.

Напомним, что в случае языков программирования английское слово *string* имеет значение строки, ряда, последовательности символов.

PASCAL

Строки символов типа *одномерный массив*

В стандартном языке ПАСКАЛЬ тип данных *строка символов* является частным случаем типа **array** и определяется конструкцией вида:

`<Имя типа> ::= packed array [1..n] of char;`

Множеством значений данного типа являются все строки, содержащие ровно n символов.

Пример:

```
Program P82;
{ Строки символов типа одномерный массив }
type      Nume = packed array [1..8] of char;
          Prenume = packed array [1..5] of char;
var       N : Nume;
          P : Prenume;
```

```

begin
  N:='Munteanu';
  P:='Mihai';
  writeln(N);
  writeln(P);
  readln;
end.

```

Результат, выводимый на экран:

```

Munteanu
Mihai

```

Так как строки различной длины принадлежат разным типам данных, в программе P82 недопустимы присваивания вида:

```

N:= 'Olaru';
P:= 'Ion'.

```

В таких случаях программисту необходимо заполнить соответствующее пространство пробелами для того, чтобы в строке было ровно n символов, например:

```

N:= 'Olaru ';
P:= 'Ion '.

```

Значения любой переменной v типа **packed array** [1.. n] of char можно ввести с клавиатуры только путем поочередного считывания соответствующих компонентов:

```

read(v[1]); read(v[2]); ...; read(v[n]).

```

Однако вывести всю строку на экран можно с помощью одного вызова write(v) или writeln(v).

Особо выделим тот факт, что строки символов типа **packed array** [1.. n] of char содержат ровно n символов, т. е. являются строками постоянной длины. Их длину нельзя изменять в процессе выполнения соответствующей программы. Это усложняет создание программ, предназначенных для обработки строк символов произвольной длины.

Строки символов типа **string**

Чтобы устранить указанный недостаток, современные версии языка ПАСКАЛЬ разрешают использование строк символов произвольной длины.

Тип данных *строка символов*, множеством значений которого являются строки произвольной длины, определяется с помощью конструкции вида:

```

type <Имя типа> = string; или type <Имя типа> = string [nmax];

```

где $nmax$ — максимальная длина, которую могут иметь соответствующие строки. При отсутствии параметра $nmax$ максимальная длина устанавливается по умолчанию, как правило — 255 символов.

К строкам типа **string** можно применять операцию конкатенации (склеивания), обозначаемую знаком «+». Текущую длину любой переменной v типа **string** можно узнать с помощью стандартной функции length(v), которая

возвращает значение типа `integer`. Независимо от длины все строки символов типа `string` являются совместимыми.

Пример:

```
Program P83_a;
{ Строки символов типа  string }
type Nume = string [8];
      Prenume = string [5];
      NumePrenume = string;
var N : Nume;
    P : Prenume;
    NP : NumePrenume;
    L : integer;
begin
  N:= 'Munteanu';    L:=length(N);    writeln(N, L:4);
  P:= 'Mihai';       L:=length(P);    writeln(P, L:4);
  NP:=N+' ' +P;      L:=length(NP);   writeln(NP, L:4);
  N:= 'Olaru';       L:=length(N);    writeln(N, L:4);
  P:= 'Ion';         L:=length(P);    writeln(P, L:4);
  NP:=N+' ' +P;      L:=length(NP);   writeln(NP, L:4);
  readln;
end.
```

Результаты, выводимые на экран:

```
Munteanu    8
Mihai       5
Munteanu Mihai  14
Olaru       5
Ion         3
Olaru Ion    9
```

Отметим, что в процессе выполнения данной программы длина строк символов `N`, `P` и `NP` изменяется.

К строкам символов можно применять операции отношения `<`, `<=`, `=`, `>=`, `>`, `<>`. Строки сравниваются посимвольно слева направо в соответствии с порядковыми номерами символов типа данных `char`. Оба операнда должны относиться к типу `packed array [1..n] of char` с одинаковым числом компонентов либо к типу `string`. Естественно, что операнды типа `string` могут быть различной длины.

Например, результатом операции:

```
'AC' < 'BA'
```

будет `true`, а результатом операции

```
'AAAAC' < 'AAAAB'
```

является `false`.

Переменную типа *строка символов* можно использовать полностью или частично, обращаясь к отдельному символу строки.

Например, в строке `P='Mihai'` имеем `P[1]='M'`, `P[2]='i'`, `P[3]='h'` и т. д. После выполнения последовательности операторов

```
P[1]:='P';
P[2]:='e';
P[3]:='t';
P[4]:='r';
P[5]:='u'
```

переменная `P` примет значение `'Petru'`.

Следующая программа вводит с клавиатуры произвольные строки символов и выводит на экран количество пробелов в соответствующей строке. Работа программы завершается после введения строки `'Конец'`.

```
Program P83_b;
{ Количество пробелов в строке символов }
var S : string;
    i, j : integer;
begin
    writeln('Введите строку символов:');
    repeat
        readln(S);
        i:=0;
        for j:=1 to length(S) do
            if S[j]=' ' then i:=i+1;
        writeln('Количество пробелов=', i);
    until S='Конец';
end.
```

Отметим, что современные компиляторы языка ПАСКАЛЬ содержат несколько стандартных функций обработки строк, например объединение, копирование, упорядочение, вставка или удаление определенных символов в строке и т. д. Описание этих функций можно найти в системах поддержки интегрированных сред разработки программ.

C++

Строки символов типа *одномерный массив*

В программах C++ строки символов могут быть представлены с помощью *одномерных массивов* с компонентами типа **char**. Такие типы объявляются грамматическими конструкциями вида:

```
typedef char <Имя типа строка символов> [nmax];
```

где `nmax` указывает максимальное количество символов, которое могут иметь строки этого типа.

Набор значений типа данных <Имя типа строка символов> состоит из строк, содержащих до `nmax` символов. Поскольку во внутреннем представлении один из компонентов одномерного массива, представляющего строку, используется для обозначения конца строки, длина строки будет составлять не более `nmax-1` значащих символов.

Примеры:

- 1) `typedef char Nume[20];`
`Nume N;`
- 2) `typedef char Prenume[10];`
`Prenume P;`
- 3) `typedef char Companie[45];`
`Companie C;`

Как было отмечено ранее, во внутреннем представлении конец строки указывается специальным символом, а именно символом *escape* `'\0'`.

Например, если переменная `P` типа `Prenume` содержит строку символов `"Ion"`, соответствующий массив имеет вид:

0	1	2	3	4	5	6	7	8	9
I	o	n	\0						

Если та же переменная `P` содержит строку символов `"Cristina"`, соответствующий массив будет иметь вид:

0	1	2	3	4	5	6	7	8	9
C	r	i	s	t	i	n	a	\0	

Поэтому, несмотря на то что в объявлении типа данных `Prenume` указано 10 символов, строки этого типа могут содержать до 9 значащих символов.

В принципе, строки, представленные одномерными массивами, могут обрабатываться путем прямого доступа к их компонентам.

Например, если вы хотите ввести строку `"Ion"` в переменную `P`, для этой цели можно использовать следующие операторы:

```
P[0]= 'I';  
P[1]= 'o';  
P[2]= 'n';  
P[3]= '\0';
```

Подчеркнем, что указание конца строки с помощью *escape*-символа `'\0'` обязательно. В противном случае возникнут ошибки выполнения.

Значения строковых переменных — одномерных массивов можно отобразить на экране с помощью оператора передачи данных `<<` в потоке вывода `cout`.

Пример:

```
// Программа P82_a  
// Строки символов типа одномерный массив  
#include <iostream>  
using namespace std;  
int main()  
{
```

```

typedef char Nume[20];
Nume N;
typedef char Prenume[10];
Prenume P;
N[0]='L'; N[1]='u'; N[2]='p'; N[3]='u'; N[4] = '\0';
P[0]='I'; P[1]='o'; P[2]='n'; P[3]='\0';
cout << N << endl;
cout << P << endl;
return 0;
}

```

Результаты, отображаемые на экране:

```

Lupu
Ion

```

При **чтении строк символов** с клавиатуры необходимо иметь в виду, что стандартный ввод рассматривается как источник, обеспечивающий непрерывный поток символов `cin`. Чтобы иметь возможность извлечь нужные строки из этого потока, программа должна знать, какой символ их разделяет. Обычно в качестве *разделителя* используются так называемые *белые символы* (пробел, табуляция →, конец строки), но программист может использовать и другие символы.

Оператор `>>` используется для чтения строк, **разделенных белыми символами**. Оператор `<<` используется для отображения/вывода строк.

Пример:

```

// Программа P82_b
// Считывание строк символов
// Разделители: пробел либо конец строки
#include <iostream>
using namespace std;
int main()
{
    typedef char Nume[20];
    Nume N;
    typedef char Prenume[10];
    Prenume P;
    cin >> N; // считывается строка символов N
    cin >> P; // считывается строка символов P
    cout << N << endl; // вывод строки символов N
    cout << P << endl; // вывод строки символов P
    return 0;
}

```

Предположим, что пользователь после запуска программы на выполнение вводит с клавиатуры:

```

Маркова<ENTER>
Кристина<ENTER>

```

В этом случае введенный `cin` имеет следующий вид:

```
Маркова<ENTER>Кристина<ENTER>
```

↑

где символ ↑ представляет его курсор. Изначально курсор находится в начале потока, то есть перед буквой М.

В процессе выполнения оператора `cin>>N` программа, начиная с текущей позиции курсора, обходит входной поток слева направо, смещая курсор после каждого прочитанного символа и помещая этот символ в переменную `N`. Как только курсор достигает разделителя конца строки `<ENTER>`, оператор `cin>>N` записывает в переменную `N` *escape*-символ `'\0'` и заканчивает считывание первой строки символов. После выполнения команды `cin>>N` переменная `N` примет значение "Маркова", а курсор окажется перед символом К.

Аналогичным образом оператор `cin>>P` продолжает обход входного потока, перемещая курсор и помещая считанные символы один за другим в переменную `P`. Как только курсор достигает разделителя конца строки `<ENTER>`, оператор `cin>>P` вводит *escape*-символ `'\0'` в переменную `P` и завершает чтение второй строки. После выполнения оператора `cin>>P` переменная `P` получит значение "Кристина", а курсор окажется в конце потока.

В результате программа отобразит на экране:

```
Маркова
Кристина
```

Подчеркнем, что операторы для чтения данных из входного потока `cin>>` используют в качестве разделителя не только конец строки `<ENTER>`, но и пробел. Таким образом, если пользователь наберет:

```
Маркова Кристина Лупу Ион <ENTER>
```

↑

переменная `N` получит значение "Маркова", переменная `P` — значение "Кристина", а курсор будет перед символом Л. Остальные символы в потоке данных, а именно "Лупу Ион", останутся сохраненными во входном потоке.

Для чтения строк, разделенных только концом строки, т. е. нажатием клавиши `<ENTER>`, используется функция `cin.getline`. Вызов этой функции имеет вид:

```
cin.getline (S, n);
```

где

`S` — переменная, в которую будет помещена строка символов из входного потока;

`n` — максимальное количество символов, которые можно считать из входного потока.

Пример:

```
// Программа P82_c
// Считывание строк символов
// Разделитель: конец строки
#include <iostream>
```

```
using namespace std;
int main()
{
    typedef char NumePrenume[30];
    NumePrenume NP;
    cin.getline(NP, 30); // Разделитель: конец строки NP
    cout << NP << endl;
    return 0;
}
```

Если в процессе выполнения вышеприведенной программы пользователь введет:

Пэдурау Елена - Переведена<ENTER>

переменная NP получит значение "Пэдурау Елена - Переведена".

Другой способ прочитать строку символов, которая может содержать пробелы, — использовать функцию `get()`.

Поскольку обработка строк путем доступа к каждому из компонентов этих массивов затруднительна, язык C++ содержит набор предопределенных функций, упрощающих этот процесс. Вот некоторые из них:

`strcpy(S_1 , S_2)` — копирует строку символов S_2 в строку S_1 ;
`strcat(S_1 , S_2)` — приклеивает к строке S_1 строку символов S_2 ;
`strlen(S)` — возвращает длину строки символов S .

Среды разработки программ на C++ содержат и другие предопределенные функции, предназначенные для обработки строк. Описание и способы использования этих функций можно найти в системах поддержки этих сред.

Для использования этих функций в разрабатываемую программу должна быть включена следующая директива:

```
#include <cstring>;
```

Пример:

```
// Программа P82_d
/* Предопределенные функции для обработки */
/* строк символов типа одномерный массив */
#include <iostream>
#include <cstring>
using namespace std;
int main()
{
    typedef char Nume[20];
    Nume N;
    typedef char Prenume[10];
    Prenume P;
    strcpy(N, "Мунтяну");
    cout << N << " " << strlen(N) << endl;
```

```
strcpy(P, "Михай");
cout << P << " " << strlen(P) << endl;
strcat(N, " ");
cout << strcat(N, P) << " " << strlen(N) << endl;
return 0;
}
```

Отображенные на экране результаты:

```
Мунтяну      7
Михай       5
Мунтяну Михай 13
```

В случае представления строк одномерными массивами они могут быть инициализированы во время объявления. Однако следует иметь в виду, что эти массивы должны содержать *escape*-символ `'\0'` (конец строки).

Примеры:

- 1) `char S[]={ 'I', 'o', 'n', '\0' };`
- 2) `char S[]="Ion";`
- 3) `char T[]={ 'E', 'l', 'e', 'n', 'a', '\0' };`

Строка символов типа `string`

Мы уже знаем, что строки символов типа `char` *<Имя типа строка символов>*`[nmax]` могут содержать не более чем `nmax-1` значимых символов. Но очень часто на момент написания программ на C++ значение `nmax` неизвестно, это усложняет процесс программирования.

Чтобы упростить работу программистов, язык программирования C++ содержит предопределенный тип строковых данных `string`, который позволяет объявлять строки без указания их максимально возможной длины. Для использования этого типа в декларативную часть программы на C++ необходимо добавить директиву:

```
#include <string>;
```

Строки типа `string` могут появляться в операторах присваивания, и с ними может выполняться операция конкатенации (склеивания), обозначенная символом `«+»`.

Пример:

```
// Программа P83_a
// Строка символов типа string
#include <iostream>
#include <string>
using namespace std;
int main()
{
```

```

string N;           // Фамилия
string P;           // Имя
string NP;          // Фамилия и имя
N = "Olaru";        // присваивание
P = "Andrei";       // присваивание
NP = N + " " + P;   // склеивание и присваивание
cout << N << endl;
cout << P << endl;
cout << NP << endl;
return 0;
}

```

Эта программа отобразит на экране:

```

Olaru
Andrei
Olaru Andrei

```

Над строками символов можно выполнять следующие операции отношения: $<$, $<=$, $=$, $>=$, $>$, $!=$. Строки сравниваются посимвольно слева направо в соответствии с положением символов в типе `char`.

Например, результат операции

```
'AC' < 'BA'
```

равен `true`, а результат операции

```
'AAAAC' < 'AAAAB'
```

равен `false`.

Переменная типа *строка символов* может быть использована целиком или частично, ссылаясь на символ в строке.

Например, после выполнения последовательности операторов

```

P="Mihai";
P[0]='P'; P[1]='e'; P[2]='t'; P[3]='r'; P[4]='u';

```

переменная `P` получит значение `"Petru"`.

Обычно чтение введенной с клавиатуры строки в переменную `S` типа `string` выполняется с помощью оператора вида:

```
getline(cin, S);
```

Длина l строки символов `S` типа `string` определяется с помощью оператора вида:

```
l = S.length();
```

где `S` — это обрабатываемая строка символов.

Следующая программа считывает с клавиатуры произвольные строки символов и отображает на экране количество пробелов в каждой считанной строке. Программа заканчивается после введения строки `'Конец'`.

```

// Программа P83_b
// Количество пробелов в строке символов
#include <iostream>
#include <string>
using namespace std;
int main()
{
    string Sir; // Считываемая с клавиатуры строка символов
    int L;      // Длина строки символов
    int n;      // Количество пробелов в считанной строке
    int i;
Repeta:
    cout << "Введите строку символов:" << endl;
    getline(cin, Sir); // Считывание строки
    L = Sir.length();  // Длина строки
    cout << "Lungimea sirului = " << L << endl;
    n = 0;
    for (i=0; i < L; i++)
        if (Sir[i] == ' ') n++;
    cout << "Количество пробелов в строке = " << n << endl;
    cout << endl;
    if (Sir != "Конец") goto Repeta;
    return 0;
}

```

Стандартные библиотеки C++ дают программисту возможность использовать оба представления строковых данных как через массивы, так и в виде *string*.

Преобразование *string* в массив можно выполнить с помощью оператора вида:

`S.c_str();`

где *S* — это переменная типа *string*.

Преобразование массива символов в *string* можно сделать простым присваиванием вида *S = T*.

Пример:

```

// Программа P83_c
// Преобразования string <--> массив символов
#include <iostream>
#include <cstring>
using namespace std;
int main()
{
    char A[20]; // A массив символов
    string B;   // B - string
    // Преобразование строки типа string B в массив A
    B="Informatica";
}

```

```

strcpy(A, B.c_str());
cout << A << endl;
// Преобразование массива A в string-ul B
A[0]='A'; A[1]='d'; A[2]='m'; A[3]='i'; A[4]='s'; A[5]='\0';
B=A;
cout << B << endl;
return 0;
}

```

Вопросы и упражнения

- ❶ Как определяется тип данных *строка символов*? Каковы два способа представления строк в программах на языке ПАСКАЛЬ/С++?
- ❷ Какие операции можно выполнять со строками, представленными одномерными массивами? А с данными типа *string*?
- ❸ ПРОГРАММИРУЙТЕ! Разработайте программу, которая запросит ввести с клавиатуры в две строки:
 - имя и фамилию ученика/ученицы;
 - профессию, к которой он/она стремится.
 Используйте наводящие вопросы и организуйте диалог человека с компьютером в соответствии со следующей моделью:

```

Как тебя зовут (имя и фамилия)?
Пэдурару Елена
Кем ты хочешь стать в будущем?
врачом
Тебя зовут Пэдурару Елена
Ты хочешь стать врачом

```

В данной модели данные, которые ввел пользователь, выделены жирным шрифтом.

- ❹ ОБРАТИТЕ ВНИМАНИЕ! Прокомментируйте следующую программу:

ПАСКАЛЬ

```

Program P84;
{ Ошибка }
var S : packed array [1..5] of
char;
begin
  S:='12345';
  writeln(S);
  S:='Sfat';
  writeln(S);
end.

```

С++

```

//Программа P84
// Ошибка
#include <iostream>
using namespace std;
int main()
{
  char S[]="12345";
  cout<<S<<endl;
  S="Sfat";
  cout<<S;
  return 0;
}

```

- 5 ЭКСПЕРИМЕНТИРУЙТЕ! Добавьте перед оператором „**return** 0;” из программы P82_b последовательность операторов:

```
cin >> N;
cin >> P;
cout << N << endl;
cout << P << endl;
```

Запустите программу на выполнение и введите строку:

Маркова Кристина Лупу Ион Ion<ENTER>

Объясните отображенные на экране сообщения.

- 6 ПРОАНАЛИЗИРУЙТЕ! Даны строки символов типа `string`. Укажите результаты операций отношения:

ПАСКАЛЬ	C++
a) 'B' < 'A'	a) 'B' < 'A'
b) 'BB' > 'AA'	b) "BB" > "AA"
c) 'BAAAA' < 'AAAAA'	c) "BAAAA" < "AAAAA"
d) 'CCCCD' > 'CCCCA'	d) "CCCCD" > "CCCCA"
e) 'A' = 'AA'	e) "A" == "AA"
f) 'BB ' < 'B'	f) "BB " < "B B"
g) 'A' = 'a'	g) "A" == "a"
h) 'Aa' > 'aA'	h) "Aa" > "aA"
i) '123' = '321'	i) "123" == "321"
j) '12345' > '12345'	j) "12345" > "12345"

- 7 РЕШИТЕ! Разработайте программу, которая считывает с клавиатуры строку символов S и на экране отобразится:

- количество вхождений символа 'A' в строке S ;
- строка, полученная заменой символа 'A' на символ '*';
- строка, полученная удалением из строки S всех вхождений символа 'B';
- количество появлений слога МА в строке S ;
- строка, полученная заменой всех вхождений в строке S слога МА на слог ТА;
- строка, полученная удалением из строки S всех вхождений слога ТО;
- обратная запись строки S ;
- `true`, если строка S палиндром, и `false` в противном случае;
- строка, полученная преобразованием всех строчных букв в составе строки S в прописные;
- строка, полученная преобразованием первой буквы каждого слова в составе строки S в заглавную букву;
- строка, полученная путем сортировки в алфавитном порядке символов в строке S .

- 8 ПРОГРАММИРУЙТЕ! Даны строки символов, состоящие из заглавных букв латинского алфавита и пробелов. Напишите программу, которая шифрует данные строки по следующим правилам:

- каждая буква от 'А' до 'У' заменяется на следующую за ней в алфавите букву;
- каждая буква 'Z' заменяется на букву 'А';
- каждый пробел заменяется на ' - '.

- 9 ПРОГРАММИРУЙТЕ! Напишите программу, которая расшифрует строки символов, зашифрованные по правилам из предыдущего упражнения.
- 10 ПРОГРАММИРУЙТЕ! Разработайте программу, которая считывает с клавиатуры строку символов S , состоящую из n символов, $n \leq 30$, и отобразит на экране все строки, получаемые путем удаления из строки S последних k символов, $k = 0, 1, 2, \dots, n - 1$. Например, для $S = \text{Test}$ программа отобразит на экране:

```
Test
Tes
Te
T
```

- 11 ТВОРИТЕ! Разработайте программу, которая:
- считает с клавиатуры m , $m \leq 100$, строк символов, состоящих из строчных букв латинского алфавита;
 - отсортирует строки в алфавитном порядке;
 - отобразит на экране отсортированные строки.
- 12 ПРОГРАММИРУЙТЕ! Строка S состоит из нескольких предложений. Предложения заканчиваются точкой, восклицательным или вопросительным знаком. Разработайте программу, которая отобразит:
- количество предложений в строке, введенной с клавиатуры;
 - количество слов в каждом из предложений.
- 13 УЧИТЕСЬ УЧИТЬСЯ! Известно, что современные компиляторы содержат несколько predefined функций для обработки строк символов, например функций конкатенации, копирования, упорядочения, вставки или удаления определенных символов и т. д. Описание этих функций можно найти в системах поддержки интегрированных сред разработки программ. Изучите индивидуально или в команде predefined функции обработки строк и заполните таблицу:

Функция	Описание	Примеры
...		
...		

Перепишите программы из этого параграфа, применив соответствующие функции. Оцените, в какой степени использование этих функций облегчает процесс разработки программ на языке ПАСКАЛЬ/С++.

- 14 ИССЛЕДУЙТЕ! Кроме функции `cin.getline`, для считывания строк символов можно использовать функцию `cin.get`. Изучите самостоятельно или совместно с одноклассниками эту функцию. Узнайте, в чем состоит различие между функциями `cin.getline` и `cin.get`. Определите, в каких случаях рекомендуется использование функции `cin.getline` и в каких — использование функции `cin.get`.

Используя функцию `cin.get`, разработайте программу, считывающую с клавиатуры строку символов и выводящую на экран ее длину. Строка может содержать один или более пробелов, а ее конец указывается символом 'x'.

1.4. Тип данных запись

Запись — тип, известный в информатике под названием *record*, или *структура*, представляет собой группу данных, объединенных под одним именем. Записи состоят из компонентов, называемых **полями**. В отличие от элементов массива поля могут относиться к разным типам. Каждое поле имеет свое имя (идентификатор поля).

Например, результаты экзамена на степень бакалавра можно представить в виде:

Имя поля →	Фамилия	Имя	Средний балл
	Маркова	Кристина	10,0
Запись →	Мунтяну	Ион	8,7

	Пэдуару	Елена	9,5

В этой модели для отображения результатов экзамена на степень бакалавра каждому ученику соответствует запись, содержащая три поля: *Nume* (Фамилия), *Prenume* (Имя) и *NotaMedie* (Средний балл). На компьютере данные в полях *Nume* и *Prenume* могут быть представлены строками символов, а данные в поле *NotaMedie* — вещественными числами.

PASCAL

В языке ПАСКАЛЬ тип данных *запись* определяется структурой вида:

```
type <Имя типа> = record
    <Имя поля 1> : T1;
    <Имя поля 2> : T2;
    ...
    <Имя поля n> : Tn;
end;
```

где T_1, T_2, \dots, T_n указывают тип соответствующих полей. Тип любого поля может быть произвольным, значит, поле, в свою очередь, может относиться к типу *запись*. Таким образом можно определять вложенные типы.

Примеры:

```
1) type Elev = record
    Nume : string;
    Prenume : string;
    NotaMedie : real;
end;
var E1, E2 : Elev;

2) type Punct = record
    x : real; { координата x }
    y : real; { координата y }
end;
var P1, P2 : Punct;
```

```

3) type Triunghi = record
        A : Punct; { вершина A }
        B : Punct; { вершина B }
        C : Punct; { вершина C }
    end;
var T1, T2, T3 : Triunghi;

```

Структура данных из приведенных выше примеров представлена на *рис. 1.4*.

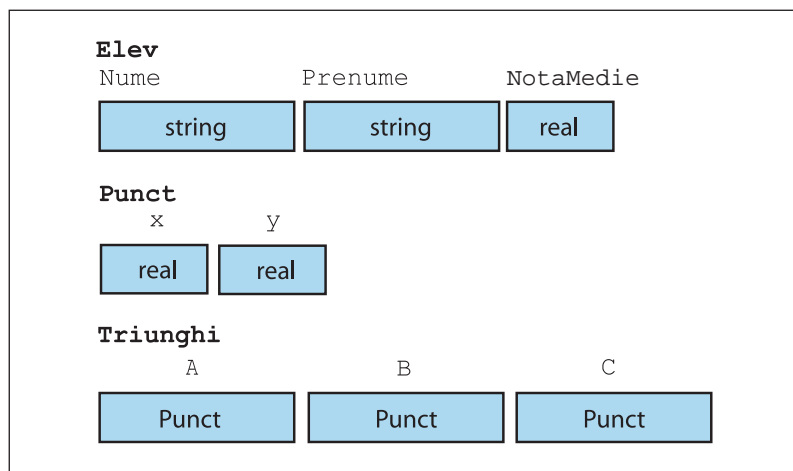


Рис. 1.4. Структура данных типа Elev, Punct и Triunghi

Если две переменные относятся к одному и тому же типу *запись*, то между ними разрешена операция присваивания. При таком присваивании все поля переменной, стоящей в правой части, копируются в переменную, стоящую в левой части. Например, для типов данных и переменных, определенных выше, следующие операторы присваивания корректны:

```

E1:=E2;
T2:=T3;
P2:=P1

```

К каждому элементу любой переменной типа **record** можно обращаться явно, по имени переменной и названию поля, которые разделяются точкой.

Примеры:

1) E1.Nume, E1.Prenume, E1.NotaMedie;

2) E2.Nume, E2.Prenume, E2.NotaMedie;

3) P1.x, P1.y, P2.x, P2.y;

4) T1.A, T1.B, T1.C, T2.A, T2.B, T2.C;

5) T1.A.x, T1.A.y, T2.B.x, T2.B.y.

Очевидно, элемент `E1.Nume` относится к типу **string**; элемент `P1.x` — к типу **real**; элемент `T1.A` — к типу **Punct**; элемент `T1.A.x` — к типу **real** и т. д.

К элементам данных типа *запись* можно применять все операции, допустимые типом соответствующего поля. Следующая программа сравнивает средние баллы двух учеников и выводит на экран имя и фамилию ученика с более высоким средним баллом. Считается, что средние баллы учеников различны.

```
Program P85;
{ Данные типа Elev }
type Elev = record
    Nume : string;
    Prenume : string;
    NotaMedie : real;
end;
var E1, E2, E3 : Elev;
begin
    writeln('Введите данные о первом ученике:');
    write('Фамилия:');    readln(E1.Nume);
    write('Имя:');    readln(E1.Prenume);
    write('Средний балл:'); readln(E1.NotaMedie);

    writeln('Введите данные о втором ученике:');
    write('Фамилия:');    readln(E2.Nume);
    write('Имя:');    readln(E2.Prenume);
    write('Средний балл:'); readln(E2.NotaMedie);

    if E1.NotaMedie > E2.NotaMedie then E3:=E1 else E3:=E2;

    writeln('Ученик с наиболее высоким средним баллом:');
    writeln(E3.Nume, ' ', E3.Prenume, ': ', E3.NotaMedie : 5:2);
    readln;
end.
```

Любой тип данных **record** может служить базовым типом для формирования других составных типов.

Пример:

```
type ListaElevilor = array [1..40] of Elev;
var LE : ListaElevilor;
```

Очевидно, обозначение `LE[i]` указывает на *i*-го ученика из списка; обозначение `LE[i].Nume` указывает на имя данного ученика и т. д. Следующая программа вводит с клавиатуры данные об *n* учениках и выводит на экран имя, фамилию и средний балл ученика с наибольшим средним баллом. Считается, что средние баллы учеников различны.

```

Program P86;
{ Массив с элементами типа Elev }
type Elev = record
    Nume : string;
    Prenume : string;
    NotaMedie : real;
end;
    ListaElev = array [1..40] of Elev;
var E : Elev;
    LE : ListaElev;
    n : 1..40;
    i : integer;
begin
    write('n='); readln(n);
    for i:=1 to n do
        begin
            writeln('Введите данные об ученике', i);
            write('Фамилия: '); readln(LE[i].Nume);
            write('Имя: '); readln(LE[i].Prenume);
            write('Средний балл: '); readln(LE[i].NotaMedie);
        end;
        E.NotaMedie:=0;
    for i:=1 to n do
        if LE[i].NotaMedie > E.NotaMedie then E:=LE[i];
    writeln('Ученик с наибольшим средним баллом:');
    writeln(E.Nume, ' ', E.Prenume, ': ', E.NotaMedie : 5:2);
    readln;
end.

```

В общем случае тип данных *запись* определяется с помощью синтаксических диаграмм на рис. 1.5. В дополнение к фиксированным записям (записям с фиксированным количеством полей) язык ПАСКАЛЬ позволяет использование вариантных записей. Такие записи изучаются в углубленных курсах информатики.

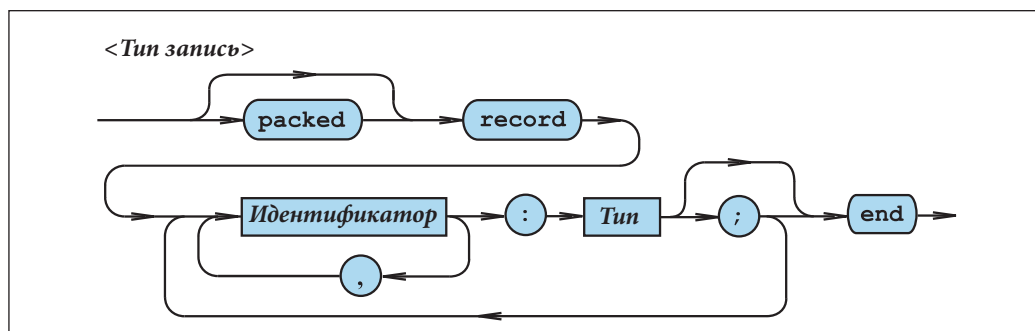


Рис. 1.5. Синтаксическая диаграмма <Тип запись>

C++

В языке C++ тип данных *запись* определяется грамматической конструкцией вида:

```
struct [<Имя типа>] {  
     $T_1$     <Имя поля 1>;  
     $T_2$     <Имя поля 2>;  
    ...  
     $T_n$     <Имя поля n>;  
};
```

где T_1, T_2, \dots, T_n указывают типы соответствующих полей. Тип любого поля может быть произвольным, значит, поле, в свою очередь, может относиться к типу *запись*. Таким образом можно определять вложенные типы.

Примеры:

- 1)

```
struct Elev {  
    string Nume[20];  
    string Prenume[25];  
    float NotaMedie;  
};  
Elev E1, E2;
```
- 2)

```
struct Punct {  
    double x;    // координата x  
    double y;    // координата y  
};  
Punct P1, P2;
```
- 3)

```
struct Triunghi {  
    Punct A;    // вершина A  
    Punct B;    // вершина B  
    Punct C;    // вершина C  
};  
Triunghi T1, T2, T3;
```

Структура данных из приведенных выше примеров представлена на *рис. 1.4**.

Если две переменные относятся к одному и тому же типу *запись*, то между ними разрешена операция присваивания. При таком присваивании все поля переменной, стоящей в правой части, копируются в переменную, стоящую в левой части. Например, для типов данных и переменных, определенных выше, следующие операторы присваивания корректны:

```
E1=E2;  
T2=T3;  
P2=P1
```

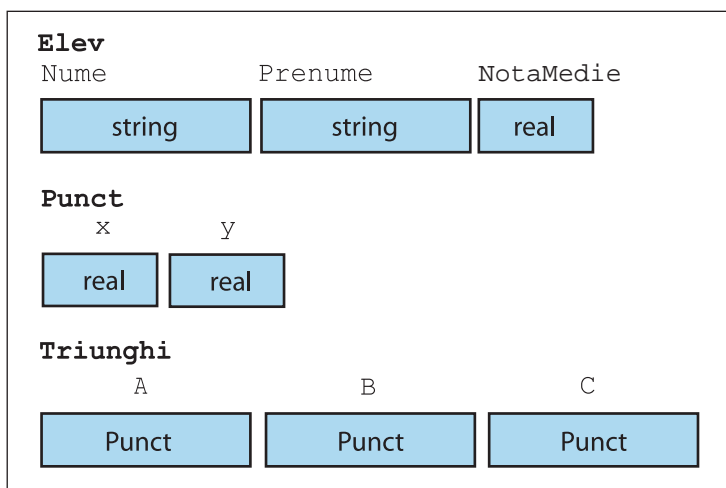


Рис. 1.4*. Структура данных типа Elev, Punct и Triunghi

К каждому компоненту любой переменной типа **struct** можно обращаться явно, по имени переменной и названию поля, которые разделяются точкой.

Примеры:

- 1) E1.Nume, E1.Prenume, E1.NotaMedie
- 2) E2.Nume, E2.Prenume, E2.NotaMedie
- 3) P1.x, P1.y, P2.x, P2.y
- 4) T1.A, T1.B, T1.C, T2.A, T2.B, T2.C
- 5) T1.A.x, T1.A.y, T2.B.x, T2.B.y
- 6) V[3].Nume, v[3].Prenume, V[3].NotaMedie

Очевидно, элемент E1.Nume относится к типу string; элемент P1.x — к типу **double**; элемент T1.A — к типу Punct; элемент T1.A.x — к типу **double** и т. д.

К элементам данных типа struct можно применять все операции, допустимые в типе соответствующего поля. Следующая программа сравнивает средние баллы двух учеников и выводит на экран имя и фамилию ученика с более высоким средним баллом. Считается, что средние баллы учеников различны.

```

//Программа P85
// Данные типа Elev
#include <iostream>
using namespace std;
int main()
{
  
```

```

struct Elev {
    char Nume[20];
    char Prenume[30];
    float NotaMedie;
};

Elev E1, E2, E3;
cout<<"Введите данные о первом ученике:"<<endl;
cout<<"Фамилия:"<<endl;      cin>>E1.Nume;
cout<<"Имя:"<<endl;          cin>>E1.Prenume;
cout<<"Средний балл:"<<endl;  cin>>E1.NotaMedie;
cout<<endl<<"Введите данные о втором ученике:"<<endl;
cout<<"Фамилия:"<<endl;      cin>>E2.Nume;
cout<<"Имя:"<<endl;          cin>>E2.Prenume;
cout<<"Средний балл:"<<endl;  cin>>E2.NotaMedie;
if (E1.NotaMedie > E2.NotaMedie) E3=E1; else E3=E2;
cout<<"Ученик с наиболее высоким средним баллом:"<<endl;
cout<<E3.Nume<<' ' <<E3.Prenume<<' ' <<E3.NotaMedie;
return 0;
}

```

Любой тип данных **struct** может служить базовым типом для формирования других структурированных типов.

Пример:

```

typedef Elev ListaElevilor[40];
ListaElevilor LE;

```

Очевидно, обозначение LE[i] указывает на *i*-го ученика из списка; обозначение LE[i].Nume указывает на имя данного ученика и т. д. Следующая программа вводит с клавиатуры данные об *n* учениках и выводит на экран имя, фамилию и средний балл ученика с наивысшим средним баллом. Считается, что средние баллы учеников различны.

```

//Программа P86
// Массив с элементами типа Elev
#include <iostream>
using namespace std;
int main()
{
    struct Elev {
        char Nume[20];
        char Prenume[30];
        float NotaMedie;
    };

    Elev E;
    int i;
    typedef Elev ListaElevilor[40];

```

```

ListaElevilor LE;
E.NotaMedie=0;
int n;
cout<<"n="; cin>>n;
for (i=0; i<n; i++)
{
    cout<<"Фамилия:"; cin>>LE[i].Nume;
    cout<<"Имя:";      cin>>LE[i].Prenume;
    cout<<"Средний балл:"; cin>>LE[i].NotaMedie;
};
for (i=0; i<n; i++)
    if (E.NotaMedie < LE[i].NotaMedie) E = LE[i];
cout << "Ученик с наивысшим средним баллом:" << endl;
cout << E.Nume << ' ' << E.Prenume << ' ' << E.NotaMedie
<< endl;
return 0;
}

```

В общем случае тип данных *запись* определяется с помощью синтаксических диаграмм на *рис. 1.5**. В дополнение к фиксированным записям (записям с фиксированным количеством полей) язык C++ позволяет использование вариантов записей, которые определяются типом данных **union**. Такие записи изучаются в более углубленных курсах информатики.

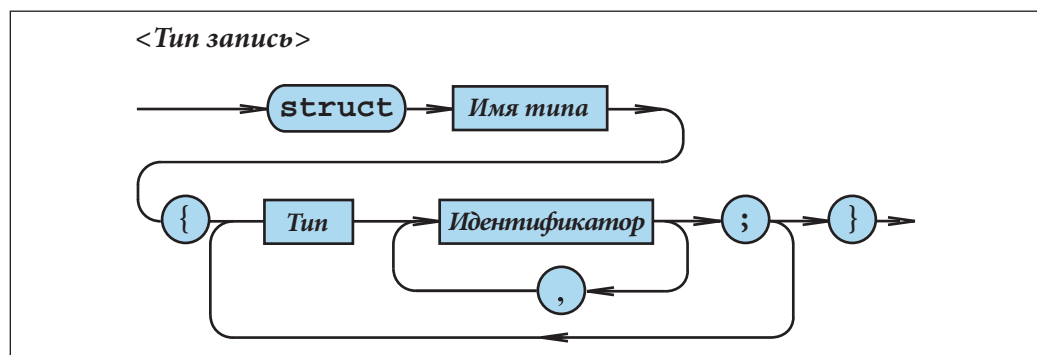


Рис. 1.5*. Синтаксическая диаграмма <Тип запись>

Вопросы и упражнения

- 1 Укажите множество значений типа данных *запись*?
- 2 Напишите металингвистические формулы, соответствующие синтаксической диаграмме на *рис. 1.5/1.5**.
- 3 Укажите на синтаксической диаграмме *рис. 1.5/1.5** пути, которые соответствуют определениям типов данных *запись* из программы Р85.
- 4 УПРАЖНЯЙТЕСЬ! Опишите типы данных *запись*, предназначенные для представления следующей информацией на компьютере:
 - а) декартовы координаты точки на плоскости;
 - б) декартовы координаты точки в пространстве;

- в) список книг в личной библиотеке: имя и фамилия автора, название книги, язык, издательство, год издания, страна, в которой она была издана;
- г) перечень автомобилей автосалона: марка, тип, цвет кузова, емкость бака, год выпуска, цена;
- д) список музыкальных произведений из личной фонотеки: имя и фамилия исполнителя, название, вид музыки, год записи;
- е) список авиапассажиров: имя, фамилия, номер рейса (строка длиной до восьми знаков), дата (день, месяц, год) и время взлета (часы, минуты), тип самолета, место в самолете (строка длиной до шести символов);
- ж) список сотрудников компании: имя, фамилия, занимаемая должность, дата приема на работу (день, месяц, год), возраст, домашний адрес (город, улица, дом, квартира).

6 ПРИМЕНИТЕ! Даны следующие объявления:

П А С К А Л Ь

```
type Elev = record
    Nume, Prenom : string;
    T, P:real;
end;
var E: Elev;
```

C ++

```
struct elev {
    char Nume[20];
    char Prenom[25];
    float T, P;
};
Elev E;
```

В описании типа Elev поле T представляет собой средний балл ученика в осеннем семестре, а поле P – это средний балл ученика в весеннем семестре. Напишите оператор присваивания, в котором рассчитывается среднегодовой балл M ученика E.

6 РЕШИТЕ! Даны следующие типы данных:

П А С К А Л Ь

```
type
Data = record
    Ziua : 1..31;
    Luna : 1..12;
    Anul : integer;
end;
Persoana = record
    NumePrenume : string;
    DataNasterii : Data;
end;
ListaPersoane = array [1..50] of
Persoana;
```

C ++

```
struct Data
{
    int Ziua;Luna;Anul;
};
struct Persoana {
    string NumePrenume;
    Data DataNasterii;
};
Persoana ListaPersoane[50];
```

Напишите программу, которая вводит с клавиатуры данные о n лицах ($n \leq 50$) и выводит на экран:

- а) фамилии и имена тех, кто родился в день z месяца;
- б) фамилии и имена тех, кто родился в месяц l года;
- в) фамилии и имена тех, кто родился в год a ;
- г) возраст каждого человека в годах, месяцах и днях $z.l.a$;
- д) фамилию и имя самого старшего человека;
- е) фамилию и имя самого младшего человека;

ж) возраст каждого человека в годах, месяцах и днях;

з) список лиц старше v лет;

и) список лиц в алфавитном порядке;

к) список лиц, упорядоченный согласно дате рождения;

л) список лиц одного возраста (рожденных в одном и том же году).

- 7 Дано n ($n \leq 30$) точек на евклидовой плоскости. Каждая точка i определяется координатами x_i, y_i . Расстояние между точками i, j вычисляется по формуле:

$$d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}.$$

Напишите программу, которая выводит на экран координаты двух точек, расстояние между которыми максимально.

- 8 Площадь треугольника вычисляется по формуле Герона:

$$S = \sqrt{p(p-a)(p-b)(p-c)},$$

где p — полупериметр, a, b и c — длины соответствующих сторон. Используя типы данных `Punct` и `Triunghi` из данного параграфа, напишите программу, которая считывает с клавиатуры информацию о n треугольниках ($n \leq 10$) и выводит на экран:

а) площадь каждого треугольника;

б) координаты вершин треугольника, площадь которого максимальна;

в) координаты вершин треугольника, площадь которого минимальна;

г) информацию о всех треугольниках в порядке возрастания их площадей.

1.5. Оператор `with`

ПРИМЕЧАНИЕ

В языке C++ такого оператора нет. Значит, данный параграф предназначен для тех учеников, которые изучают только язык ПАСКАЛЬ.

В языке ПАСКАЛЬ к каждому элементу любой переменной типа *запись* можно обращаться явно, по имени переменной и названию поля, которые разделяются точкой.

Например, при следующем описании

```
type Angajat = record
    NumePrenume : string;
    ZileLucrate : 1..31;
    PlataPeZi : real;
    PlataPeLuna : real;
end;
var A : Angajat;
```

к элементам переменной A можно обращаться через $A.NumePrenume$, $A.ZileLucrate$, $A.PlataPeZi$ и $A.PlataPeLuna$.

Так как имя A переменной типа *запись* постоянно повторяется, то такой способ обращения к элементам является очень громоздким. Этих повторений можно избежать при использовании оператора `with` (с).

Синтаксис данного оператора:

$\langle \text{Оператор with} \rangle ::= \text{with } \langle \text{Переменная} \rangle \{, \langle \text{Переменная} \rangle\} \text{ do } \langle \text{Оператор} \rangle$

Синтаксическая диаграмма представлена на рис. 1.6.

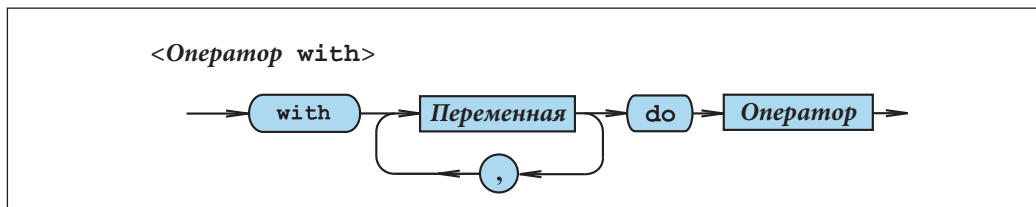


Рис. 1.6. Синтаксическая диаграмма оператора `with`

Внутри оператора **with** к элементам одной или нескольких переменных типа *запись* можно обращаться только по имени соответствующего поля.

Пример:

```
with A do PlataPeLuna:=PlataPeZi*ZileLucrate
```

Этот оператор эквивалентен следующему:

```
A.PlataPeLuna:=A.PlataPeZi*A.ZileLucrate
```

Использование оператора **with** требует особого внимания от программиста, который обязан объявлять однозначным образом элементы переменных типа *запись*. Внутри оператора **with**, при появлении некоторого идентификатора, вначале проводится проверка того, может ли он быть интерпретирован как имя поля соответствующей записи. Если да, то он будет интерпретирован как имя поля, даже если в данный момент доступна другая переменная под таким же именем.

Пример:

```
type Punct = record
    x : real;
    y : real;
end;
Segment = record
    A : Punct;
    B : Punct;
end;
var P : Punct;
    S : Segment;
    x : integer;
```

В нашем случае идентификатор `x` может указывать на переменную `x` типа `integer` или на поле `P.x` записи `P`.

В операторе

```
x:=1
```

идентификатор `x` указывает на переменную `x` типа `integer`.

В операторе

```
with P do x:=1
```

идентификатор x указывает на поле $P.x$ переменной P типа `Punct`.

Так как переменная S типа `Segment` не содержит поля с именем $S.x$, то в операторе

```
with S do x:=1
```

идентификатор x будет интерпретирован как переменная x типа `integer`.

Оператор вида

```
with  $v_1, v_2, \dots, v_n$  do <Оператор>,
```

где v_1, v_2, \dots, v_n — переменные типа *запись*, эквивалентен оператору:

```
with  $v_2$  do  
{ ... }  
with  $v_n$  do <Оператор>.
```

Очевидно, что поля записей v_1, v_2, \dots, v_n должны быть определены однозначно.

Например, для переменных P и S , описанных выше, можно записать:

```
with P, S do  
begin  
  x:=1.0;    { ссылка на P.x }  
  y:=1.0;  
  A.x:=0;    { ссылка на S.A.x }  
  A.y:=0;  
  B.x:=2.0; { ссылка на S.B.x }  
  B.y:=2.0;  
end;
```

Как правило, оператор **with** используется только в тех случаях, когда это приводит к значительному сокращению текста программы.

Вопросы и упражнения

- 1 Укажите на синтаксической диаграмме *рис. 1.6* пути, которые соответствуют операторам **with** из примеров, приведенных в данном параграфе.
- 2 В чем назначение оператора **with**?
- 3 ПРИМЕНИТЕ! Используя оператор **with**, исключите из программ P85 и P86, рассмотренных в предыдущем параграфе, повторения типа

```
E1.Nume, E1.Prenume, ...,  
LE[i].Nume, LE[i].Prenume.
```

4 РЕШИТЕ! Даны следующие типы данных:

```
type Angajat = record
    NumePrenume : string;
    ZileLucrate : 1..31;
    PlataPeZi : real;
    PlataPeLuna : real;
end;
ListaDePlata = array [1..50] of Angajat;
```

Ежемесячная зарплата каждого работника вычисляется путем умножения ежедневной платы на количество отработанных дней. Напишите программу, которая:

- а) вычисляет ежемесячную зарплату каждого работника;
- б) вычисляет среднюю зарплату всех работников, включенных в список;
- в) выводит на экран данные о всех работниках, ежемесячная зарплата которых максимальна;
- г) выводит на экран список работников в алфавитном порядке;
- д) выводит на экран список работников в порядке возрастания ежедневной платы;
- е) упорядочивает список работников в порядке возрастания ежемесячной зарплаты;
- ж) выводит на экран список работников в порядке возрастания количества отработанных дней.

5 РЕШИТЕ! Окружность может быть задана через координаты x , y центра и радиус r . Напишите программу, которая считывает с клавиатуры данные об n окружностях ($n \leq 50$) и выводит на экран:

- а) координаты центра и радиус окружности, которая описывает круг максимальной площади;
- б) количество окружностей, входящих в круг с максимальным радиусом, и координаты соответствующих центров;
- в) координаты центра и радиус окружности, которая описывает круг минимальной площади;
- г) количество окружностей, в которые входит круг с минимальным радиусом, и координаты соответствующих центров.

1.6. Тип данных множество

Напомним, что множество состоит из физических или виртуальных (мысленных) объектов, обладающих общим свойством. Объекты, составляющие множество, называются *элементами множества*. Элементы любого множества различны, а их порядок не важен.

PASCAL

В языке ПАСКАЛЬ тип данных *множество* определяется по отношению к базовому типу, который должен быть порядковым:

$\langle \text{Тип множества} \rangle ::= [\text{packed}] \text{ set of } \langle \text{Тип} \rangle$

Значениями типа данных *множество* являются множества, состоящие из значений базового типа. Если базовый тип имеет n значений, то тип *множество* будет иметь 2^n значений. Значение n ограничено: $n \leq 256$.

В языке ПАСКАЛЬ элементы множества могут перечисляться в квадратных скобках «[» и «]», которые являются аналогом фигурных скобок в математике.

Запись [] означает пустое множество.

Примеры:

```
type Indice = 1..10;  
  Zi = (L, Ma, Mi, J, V, S, D);  
  MultimeIndicii = set of Indice;  
  ZileDePrezenta = set of Zi;  
var MI : MultimeIndicii;  
  ZP : ZileDePrezenta;
```

Порядковый тип Indice имеет $n=10$ значений: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10. Тип MultimeIndicii имеет $2^{10} = 1024$ значений, а именно:

```
[], [1], [2], ..., [1, 2], [1, 3], ...,  
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10].
```

Таким образом, переменная MI может принимать любое из этих значений, например:

```
MI := [1, 3].
```

Порядковый тип Zi имеет $n=7$ значений: L, Ma, Mi, J, V, S, D. Тип ZileDePrezenta имеет $2^7 = 128$ значений, а именно:

```
[], [L], [Ma], [Mi], ..., [L, Ma], [L, Mi], ...,  
[L, Ma, Mi, J, V, S, D].
```

Переменная ZP может принимать любое из этих значений, например:

```
ZP := [L, Ma, Mi, V]
```

Значения типа *множество* могут определяться через конструктор множества. Синтаксическая диаграмма грамматической единицы <Конструктор множества> представлена на рис. 1.7.

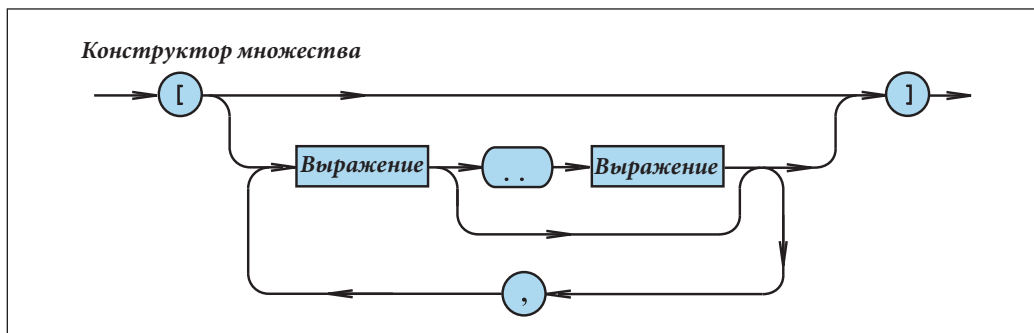


Рис. 1.7. Синтаксическая диаграмма <Конструктор множества>

Конструктор содержит значения элементов множества, разделенных запятыми и заключенных в квадратные скобки. Элемент может являться конкретным значением базового типа или интервалом вида:

<Выражение> .. <Выражение>.

Значения данных выражений представляют собой верхний и нижний пределы интервала.

Примеры:

- 1) `[];`
- 2) `[1, 2, 3, 8];`
- 3) `[1..4, 8..10];`
- 4) `[i-k..i+k];`
- 5) `[L, Ma, V..D].`

К значениям типа данных *множество* можно применять обычные операции:
+ объединение;
* пересечение;
- разность,

результат которых относится к типу *множество*, и операции отношения:

= равенство;
<> неравенство;
<=, >= включение;
in принадлежность,

результат которых относится к типу `boolean`.

Следующая программа выводит на экран результаты операций +, * и -, применяемых к переменным типа `MultimeIndicii`.

```
Program P87;  
{ Данные типа MultimeIndicii }  
type Indice = 1..10;  
    MultimeIndicii = set of Indice;  
var A, B, C : MultimeIndicii;  
    i : integer;  
begin  
    A:= [1..5, 8];      { A содержит 1, 2, 3, 4, 5, 8 }  
    B:= [1..3, 9, 10]; { B содержит 1, 2, 3, 9, 10 }  
    C:= [];             { C является пустым множеством }  
    C:=A+B;             { C содержит 1, 2, 3, 4, 5, 8, 9, 10 }  
  
    writeln('Объединение');  
    for i:=1 to 10 do  
        if i in C then write(i:3);  
    writeln;  
  
    C:=A*B;             { C содержит 1, 2, 3 }  
    writeln('Пересечение');  
    for i:=1 to 10 do  
        if i in C then write(i:3);  
    writeln;
```

```

C:=A-B;           { C содержит  4, 5, 8 }
writeln('Разность');
for i:=1 to 10 do
  if i in C then write(i:3);
writeln;
readln;
end.

```

В отличие от массивов и записей, к компонентам которых существует прямой доступ соответственно через индексы и названия полей, к элементам множества прямого доступа нет. Допускается только проверка на принадлежность элемента множеству (операция отношения **in**). Благодаря этому при использовании типов данных *множество* увеличивается скорость работы и улучшается читабельность программ на языке ПАСКАЛЬ.

Например, оператор

```
if (c='A') or (c='E') or (c='I') or (c='O') or (c='U') then ...
```

можно заменить более простым оператором:

```
if c in ['A','E','I','O','U'] then ...
```

Знаете ли вы?

Эратосфен был греческим математиком, географом, поэтом, астрономом и музыкантом, жившим между 276 и 194 годами до нашей эры. Он был первым человеком, который вычислил окружность Земли, вычислил наклон оси вращения Земли, сформулировал концепцию високосного года, нарисовал одну из первых карт мира. Эратосфен изобрел очень эффективный алгоритм для вычисления всех простых чисел, которые не превышают заданное число.

Источник: : <https://www.britannica.com>



Другим примером является использование типов данных *множество* для вычисления простых чисел, меньших заданного n , где n — натуральное число. Для этого применяется алгоритм *Решето Эратосфена*:

- 1) в решете находятся числа 2, 3, 4, ..., n ;
- 2) из решета удаляется наименьшее число i ;
- 3) указанное число включается в множество простых чисел;
- 4) из решета удаляются все числа m , кратные числу i ;
- 5) процесс завершается, когда решето становится пустым.

```

Program P88;
{ Решето Эратосфена }
const n = 50;

```

```

type MultimeDeNumere = set of 1..n;
var Sita, NumerePrime : MultimeDeNumere;
    i, m : integer;
begin
  {1} Sita:= [2..n];
    NumerePrime:=[];
    i:=2;
    repeat
  {2}   while not (i in Sita) do i:=succ(i);
  {3}   NumerePrime:=NumerePrime+[i];
        write(i:4);
        m:=i;
  {4}   while m<=n do
        begin Sita:=Sita-[m]; m:=m+i; end;
  {5} until Sita=[];
        writeln;
        readln;
end.

```

Соответствие между шагами алгоритма и операторами программы, их реализующими, указано в виде комментариев в левой части строк программы.

C++

В языке C++ множества могут быть представлены одномерными массивами. Обычно соответствующие компоненты объявлены порядковыми типами данных: **int**, **bool**, **char** и **enum**.

Примеры:

1) множества A , содержащие до 100 целых чисел, могут быть представлены массивом **int** $A[100]$;

2) множества B , содержащие до 26 символов латинского алфавита, могут быть представлены массивом **char** $B[26]$;

3) множества Z , сформированные из значений типа данных **enum** Z_i {L, Ma, Mi, J, V, S, D}, могут быть представлены массивом Z_i $Z[7]$.

Очевидно, что кроме самого массива, в котором хранятся элементы обрабатываемого множества, нам нужно будет объявить целочисленную переменную n , в которой будет храниться текущее количество элементов этого множества.

В случае вышеописанных множеств A , B и Z можно объявить переменные n_A , n_B и n_Z , например так: **int** n_A , n_B , n_Z .

Представляем далее программу, которая считывает с клавиатуры множества A и B , содержащие не более 20 целых чисел, и отображает на экране их пересечение, то есть множество $C = A \cap B$.

Напомним, что множество C должно содержать только те элементы, которые есть как во множестве A , так и во множестве B .

```

// Программа P87
// Представление множеств через массивы
// Пересечение множеств A и B

```

```

#include <iostream>
using namespace std;
int main ()
{
    int A[20], B[20], C[20],
        nA, nB, nC,
        i, j, k; // счетчики циклов
    bool Gasit; // флажок: элемент множества A
                // найден во множестве B
    cout << "Введите количество элементов множества A: ";
    cin >> nA;
    cout << "Введите элементы множества A:" << endl;
    for (i=0; i<nA; i++) cin >> A[i];
    cout << "Введите количество элементов множества B: ";
    cin >> nB;
    cout << "Введите элементы множества B:" << endl;
    for (j=0; j<nB; j++) cin >> B[j];
    nC=0; // изначально множество C пусто
    for (i=0; i<nA; i++)
    {
        Gasit = false;
        for (j=0; j<nB; j++) if (A[i]==B[j]) Gasit = true;
        if (Gasit==true) { C[nC] = A[i]; nC++; };
    }
    cout << "Количество элементов множества C = " << nC << endl;
    cout << "Множество C:" << endl;
    for (k=0; k<nC; k++) cout << C[k] << " ";
    cout << endl;
    return 0;
}

```

Другой метод представления множеств состоит в использовании так называемых **характеристических векторов**. Этот метод предполагает, что существует некое универсальное множество (*universum*), обозначенное через U , которое содержит все элементы, из которых могут формироваться необходимые для обработки множества.

Например, предполагается обработка множеств, элементами которых могут быть натуральные числа, меньшие 100, тогда $U = \{0, 1, 2, 3, \dots, 99\}$. Подобным образом, если предполагается обработка множеств, элементами которых могут быть заглавные буквы латинского алфавита, тогда $U = \{A, B, C, \dots, Z\}$.

Предположим, что универсальное множество содержит n элементов:

$$U = \{u_1, u_2, u_3, \dots, u_n\},$$

а обрабатываемое множество содержит m элементов:

$$A = \{a_1, a_2, a_3, \dots, a_m\}.$$

Это множество может быть представлено на компьютере с помощью характеристического вектора

$$V = (v_1, v_2, v_3, \dots, v_i, \dots, v_n),$$

где

$$v_i = \begin{cases} 1, & \text{если } u_i \in A; \\ 0, & \text{в противном случае.} \end{cases}$$

Другими словами, компонент i характеристического вектора V имеет значение 1, если соответствующий элемент универсального множества принадлежит множеству A , и значение 0 в противном случае.

Например, если $U = \{1, 2, 3, 4, 5\}$ и $A = \{2, 4\}$, то характеристический вектор:

$$V = (0, 1, 0, 1, 0).$$

Аналогично, если $U = \{a, b, c, d, e, f\}$ и $B = \{a, c, e, f\}$, то характеристический вектор:

$$V = (1, 0, 1, 0, 1, 1).$$

Из вышеприведенных примеров видно, что количество компонентов характеристического вектора V некоторого множества A совпадает с количеством элементов n универсального множества U . Очевидно, что количество ненулевых компонентов характеристического вектора V равно m — количеству элементов множества A .

Наглядным примером использования характеристических векторов для представления множеств, подлежащих обработке, является вычисление простых чисел, меньших заданного натурального числа n . Для этого используется алгоритм решета Эратосфена:

- 1) в решете находятся числа 2, 3, 4, ..., n ;
- 2) из решета извлекается наименьшее число i ;
- 3) извлеченное число включается в множество простых чисел;
- 4) все числа m , кратные числу i , удаляются из решета;
- 5) процесс заканчивается, когда решето становится пустым.

```
// Программа P88
// Решето Эратосфена
// Представление множеств через характеристические векторы
// Universum U = {0, 1, 2, ..., n}
#include <iostream>
using namespace std;
int main()
{
    const int nmax = 51;
    int n, S[nmax], // Характеристический вектор решето
        P[nmax],   // Характеристический вектор Простые числа
        i,         // Извлеченное из решета число
        j, k;      // Счетчики циклов
    cout << "Введите n = "; cin >> n;
    // Помещаем числа 2, 3, 4, ..., n in Sita
    S[0] = 0; S[1] = 0;
    for (j = 2; j <= n; j++) S[j] = 1;
    // Инициализируем множество простых чисел
    for (j = 0; j <= n; j++) P[j] = 0;
    // Извлекаем из решета наименьшее число i
```

```

for (i = 2; i <= n; i++)
{
    if (S[i] != 0)
    {
        P[i] = 1; // включаем i во множество простых чисел
        S[i] = 0; // Исключаем i из решета
        // Исключаем из решета все кратные i числа
        for (j = i; j <= n; j++) if ((j % i) == 0) S[j] = 0;
    }
}
cout << "Простые числа:" << endl;
for (j = 2; j <= n; j++) if (P[j] == 1) cout << j << ' ';
cout << endl;
return 0;
}

```

Вопросы и упражнения

- ❶ ОБРАТИТЕ ВНИМАНИЕ! (ПАСКАЛЬ) Перечислите допустимые значения переменных, описанных ниже:

```

var V : set of 'A'..'C';
      S : set of (A, B, C);
      I : set of '1'..'2';
      J : set of 1..2;

```

- ❷ ОБРАТИТЕ ВНИМАНИЕ! (ПАСКАЛЬ) Прокомментируйте следующую программу:

```

Program P89;
{ Ошибка }
type Multime = set of integer;
var M : Multime;
      i : integer;
begin
    M:=[1, 8, 13];
    for i:=1 to MaxInt do
        if i in M then writeln(i);
end.

```

- ❸ УПРАЖНЯЙТЕСЬ! (ПАСКАЛЬ) Даны следующие объявления:

```

type Culoare = (Galben, Verde, Albastru, Violet);
      Nuanta = set of Culoare;
var NT : Nuanta;

```

Перечислите значения, которые может принимать переменная NT.

- ❹ УПРАЖНЯЙТЕСЬ! (ПАСКАЛЬ) Напишите металингвистическую формулу, которая соответствует синтаксической диаграмме <Конструктор множества> рис. 1.7.

- 5 УПРАЖНЯЙТЕСЬ! (ПАСКАЛЬ) Рассматривается тип данных `MultimeIndicii`, описанный в этом параграфе. Перечислите элементы множеств, определяемых следующими конструкторами:

a) `[]`;

f) `[4..3]`;

b) `[1..10]`;

g) `[1..3, 7..6, 9]`;

c) `[1..3, 9..10]`;

h) `[4-2..7+1]`;

d) `[1+1, 4..7, 9]`;

i) `[7-5..4+4]`;

e) `[3, 7..9]`;

j) `[6, 9, 1..2]`.

- 6 УПРАЖНЯЙТЕСЬ! (C++) Дано универсальное множество $U = \{L, Ma, Mi, J, V, S, D\}$, представляющее дни недели. Запишите характеристические векторы следующих множеств:

$A = \{S, D\}$ — выходные дни недели;

$B = \{L, Ma, Mi, J, V\}$ — рабочие дни недели;

$C = \{L, Mi, J\}$ — дни, в которые проводятся консультации онлайн по информатике;

$D = \{Ma, Mi\}$ — дни занятий в творческих кружках лица;

$E = \{Ma, J, V\}$ — дни занятий в спортивных секциях лица;

$F = \{L, V\}$ — дни, в которые организуются книжные выставки в библиотеке лица.

- 7 УПРАЖНЯЙТЕСЬ! (C++) Зная множества U, A, B, C, D, E, F , описанные в предыдущем задании, напишите характеристические векторы множеств, которые содержат:

G — дни недели, в которые занятия проходят как в творческих кружках, так и в спортивных секциях;

H — дни недели, в которые занятия проходят либо в творческих кружках, либо в спортивных секциях;

I — дни недели, в которые не работают как творческие кружки, так и спортивные секции.

Напишите программу, которая вычисляет и отображает на экране эти множества.

- 8 ПРОГРАММИРУЙТЕ! Напишите программу, которая считывает с клавиатуры элементы множеств A и B и отображает на экране:

a) пересечение множеств A и B ;

b) объединение множеств A и B ;

в) разность множеств A и B .

Множества A и B состоят из целых чисел.

- 9 ПРОГРАММИРУЙТЕ! Напишите программу, которая считывает с клавиатуры элементы множеств A и B и отображает на экране:

a) пересечение множеств A и B ;

b) объединение множеств A и B ;

в) разность множеств A и B ;

Множества A и B состоят из заглавных букв латинского алфавита.

- 10 РЕШИТЕ! Напишите программу, отображающую на экране все подмножества множества $\{1, 2, 3, 4\}$.

- 11 РЕШИТЕ! Напишите программу, отображающую на экране все подмножества множества $\{A, B, C, D\}$.

- 12 Рассматриваются строки символов, состоящие из прописных и строчных букв латинского алфавита, в которых слова разделены пробелом, точкой, запятой, точкой с запятой, восклицательным или вопросительным знаком.

Разработайте программу, которая отображает на экране слова в строках, прочитанных с клавиатуры.

- 13 Рассматриваются строки символов, состоящие из прописных и строчных букв латинского алфавита. Разработайте программу, которая отображает на экране количество гласных в строках, прочитанных с клавиатуры.
- 14 Разработайте программу, которая считывает с клавиатуры две строки символов и отображает на экране:
- а) символы, которые встречаются хотя бы в одной из строк;
 - б) символы, которые появляются в обеих строках;
 - в) символы, которые появляются в первой строке и не появляются во второй строке.
- 15 Имя человека считается написанным правильно, если оно состоит из прописных и строчных букв латинского алфавита, начинается с заглавной буквы, а следующие буквы являются строчными. Напишите программу, которая проверяет, является ли строка, прочитанная с клавиатуры, правильным именем человека. Если имя правильное, на экране будет отображаться сообщение ПРАВИЛЬНО, а в противном случае — сообщение НЕПРАВИЛЬНО.
- 16 ТВОРИТЕ! (ПАСКАЛЬ) В текущих реализациях языка количество значений базового типа для типа данных *множество* ограничено, обычно $n \leq 256$. Следовательно, программа P88 не может вычислять простые числа больше n . Разработайте программу для вычисления простых чисел в диапазоне 8, ..., 10000. *Подсказка:* решето в алгоритме Эратосфена можно представить в виде массива, компоненты которого являются множествами.
- 17 ТЕМАТИЧЕСКОЕ ИССЛЕДОВАНИЕ. (C++) Напишите программу, которая вычисляет простые числа, используя алгоритм *решета Эратосфена*, представляющий множества чисел *Sita* и *NumerePrime* не характеристическими векторами, а массивами, содержащими их фактические элементы. Сравните сложность написанной программы с программой, использующей характеристические векторы.
- 18 ТВОРИТЕ! Даны телевизионные каналы *A* и *B*. В определенные дни недели, не обязательно в одни и те же, каждый из этих каналов не транслирует художественные фильмы. Например, канал *A* не транслирует художественные фильмы по *понедельникам* и *пятницам*, а канал *B* — по *вторникам* и *воскресеньям*. Напишите программу, которая для каждого из каналов *A* и *B* считывает с клавиатуры дни недели, когда художественные фильмы не транслируются, и отображает на экране дни, когда у зрителя есть возможность посмотреть художественный фильм хотя бы на одном из каналов. Дни недели будут считываться с клавиатуры и отображаться на экране обычным образом, то есть *понедельник*, *вторник*, *среда* и т. д., без использования их порядковых номеров или сокращений. *Подсказка:* используйте множества, элементы которых относятся к *перечисляемому* типу данных.
- 19 ИЗУЧИТЕ! (C++) Известно, что множества могут быть представлены одномерными массивами, компоненты которых содержат фактические элементы этих множеств. В то же время каждый массив должен ассоциироваться с целым числом — количеством элементов этого множества. Например, в случае множества *A*, состоящего не более чем из 20 целых чисел, переменная *nA* ассоциируется с массивом `int A[20]`. Чтобы сделать программы более интуитивно понятными, множества также могут быть представлены с помощью типа данных *запись*, который имеет вид `struct Multime {int E[nmax], nE;}`. В этом типе данных массив *E* предназначен для хранения элементов множества, а переменная *nE* — для хранения текущего количества элементов соответствующего множества. Перепишите программы C++ из этого параграфа, используя для представления множеств данные типа *запись*.

- ❷ УЧИТЕСЬ УЧИТЬСЯ! (C++) Стандартная библиотека языка C++ содержит специальный контейнер `Set`, предназначенный для представления и обработки данных типа *множество*. Рекомендуем увлеченным программированием ученикам изучить индивидуально или в команде средства, предлагаемые в этом контейнере. Попробуйте переписать программы из этого параграфа, используя функции, определенные в контейнере `Set`.

1.7. Общие сведения о файлах

До сих пор разработанные нами программы считывали обрабатываемые данные с клавиатуры и выводили результаты обработки на экран. Данные, введенные с клавиатуры, сохранялись в переменных из соответствующих программ. Однако как только выполнение программы завершается, введенные значения переменных теряются, что требует повторного ввода начальных данных при новом запуске на выполнение программы. В случае объемных входных данных их повторный ввод требует большого объема работы.

В то же время из опыта, приобретенного в процессе обработки текстов, изображений, аудио- и видеофрагментов, мы уже знаем, что данные, подлежащие обработке, и, конечно же, результаты обработки должны храниться в файлах. Напомним, что файл представляет собой набор данных, который имеет имя и хранится на внешнем носителе информации, например на магнитных дисках или оптических дисках, *флэш*-памяти и т. д.

Чтобы иметь возможность обрабатывать информацию из файлов на внешних носителях, современные языки программирования содержат структурированные типы данных, называемые *файлами*. В языках ПАСКАЛЬ и C++ под файлом понимается структура данных, состоящая из последовательности компонентов (записей). Количество компонентов в последовательности является произвольным, а конец последовательности обозначается специальным символом *EOF* (*End of File* — конец файла). Файл, не содержащий компонентов, называется *пустым файлом*.

В целом использование файлов дает следующие преимущества:

- обеспечивается долгосрочное хранение как входных, так и выходных данных, пользователь имеет возможность вернуться к ним в любое время после завершения процесса выполнения соответствующих программ;
- программы становятся универсальными, так как одну и ту же программу можно без изменений использовать для обработки данных из разных внешних файлов;
- сложная обработка может быть выполнена путем разделения задач между разными более простыми программами, поскольку выходные данные одной программы могут использоваться в качестве входных данных для другой программы.

ВНИМАНИЕ!

Способы обработки файлов в значительной степени зависят от специфики используемых операционных систем и сред программирования. Как следствие, результаты, предоставляемые программами обработки файлов, могут отличаться от компьютера к компьютеру.

Поэтому ученикам предлагается изучить средства обработки файлов методом компьютерного эксперимента. Для этого рекомендуется писать и запускать небольшие программы, которые обрабатывают данные с внешних носителей.

PASCAL

Под файлом понимают структуру данных, которая состоит из последовательности компонент. Все компоненты файла относятся к одному и тому же типу, который называется базовым. Число компонентов файла является произвольным, однако конец файла обозначается специальным символом: *EOF* (*End of File* – конец файла). Файл, который не содержит ни одного элемента, называется *пустым файлом*.

Файловый тип данных определяется следующим образом:

$\langle \text{Файловый тип} \rangle ::= [\text{packed}] \text{ file of } \langle \text{Тип} \rangle;$

где $\langle \text{Тип} \rangle$ является базовым типом. Базовый тип может быть любым, кроме самого файлового типа (не существует «файл файлов»).

Примеры:

- 1)

```
type FisierNumere = file of integer;
var FN : FisierNumere;
    n : integer;
```
- 2)

```
type FisierCaractere = file of char;
var FC : FisierCaractere;
    c : char;
```
- 3)

```
type Elev = record
    Nume : string;
    Prenume : string;
    NotaMedie : real;
end;
FisierElevi = file of Elev;
var FE : FisierElevi;
    E : Elev;
```

Структура данных файлового типа представлена на *рис. 1.8*. Отметим, что символ *EOF*, который означает конец файла, не является компонентом файла.

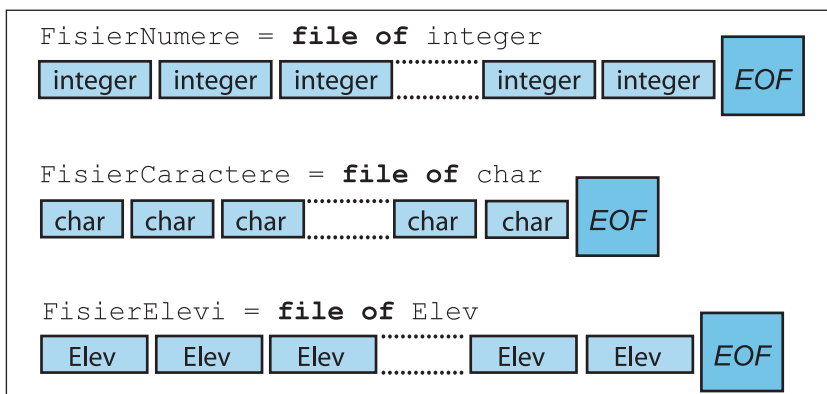


Рис. 1.8. Структура данных типа *FisierNumere*, *FisierCaractere* и *FisierElevi*

Переменные *FN*, *FC*, *FE* файлового типа называются **логическими файлами**, **файлами языка ПАСКАЛЬ**, или просто **файлами**. В отличие от остальных типов данных, значения которых хранятся во внутренней па-

мяти компьютера, данные файлов хранятся на периферийных устройствах — носителях информации (на дисках, магнитных лентах, оптических дисках, бумаге принтера или на устройстве считывания документов и др.). Информация на таких носителях хранится в виде **внешних файлов** в соответствии с требованиями операционной системы. Таким образом, перед использованием переменной файлового типа необходимо связать ее с некоторым внешним файлом. Методы связи зависят от версии языка и операционной системы, установленной на компьютере.

В стандартной версии языка связь осуществляется посредством включения переменных файлового типа в заголовок программы в качестве аргументов.

В Turbo PASCAL связь файловой переменной f с внешним файлом осуществляется вызовом процедуры:

```
assign(f, s);
```

где s — это выражение типа **string**, задающее путь доступа и имя внешнего файла.

Примеры:

1) `assign(FN, 'A:\REZULTAT\R.DAT')`

— файл FN связывается с внешним файлом R.DAT, находящимся в каталоге REZULTAT на диске A.

2) `assign(FC, 'C:\A.CHR')`

— файл FC связывается с файлом A.CHR, находящимся в корневом каталоге диска C.

3) `write('Введите имя файла:');
readln(str);
assign(FE, str);`

— файл FE связывается с внешним файлом, имя которого считывается с клавиатуры в переменную `str` типа **string**.

После выполнения оператора `assign(f, s)` все операции, осуществляемые над файлом f , фактически будут выполняться над внешним файлом s .

Самыми распространенными операциями, выполняемыми над файлами, являются считывание компонентов из файла и их запись в файл.

Считывание текущей компоненты из файла осуществляется с помощью оператора вызова процедуры:

```
read(f, v),
```

где v — переменная, которая относится к базовому типу файла f .

Запись следующей компоненты в файл осуществляется с помощью оператора вызова процедуры:

```
write(f, e),
```

где e — выражение, относящееся к базовому типу файла f .

Примеры:

- 1) `read(FN, n);`
- 2) `write(FC, c);`
- 3) `read(FE, E).`

По **типам операций**, применяемых к компонентам, файлы подразделяются на:

- входные (открыты только для чтения);
- выходные (открыты только для записи);
- рабочие (открыты и для чтения, и для записи).

По **методу доступа** к компонентам файлы подразделяются на:

- файлы последовательного доступа (доступ к компоненте i возможен только после считывания или записи компоненты $i - 1$);
- файлы прямого доступа (к любой компоненте есть прямой доступ через ее порядковый номер i в файле).

Отметим, что в стандартном языке допустимы только входные и выходные файлы последовательного доступа.

Тип файла (входной, выходной или рабочий) и метод доступа (прямой или последовательный) задаются при **открытии файла**. В стандартном языке существуют следующие процедуры для открытия файлов:

`reset(f)` — открывает существующий файл для чтения;

`rewrite(f)` — создает пустой файл для записи.

После завершения обработки компонент *файл* нужно закрыть. При **закрытии файла** операционная система записывает символ *EOF*, регистрирует только что созданный файл в соответствующем каталоге и т. д.

В стандартном языке по окончании работы программы все файлы закрываются автоматически. В Turbo PASCAL закрытие файла f осуществляется явно с помощью оператора процедуры `close(f)`.

В заключение приведем порядок вызова процедур, предназначенных для обработки данных файлового типа:

- | | |
|--|---|
| 1) <code>assign(f, s)</code> | — связывание файловой переменной f с внешним файлом s ; |
| 2) <code>reset(f)/rewrite(f)</code> | — открытие файла f для чтения / записи; |
| 3) <code>read(f, v)/write(f, e)</code> | — чтение/запись текущей компоненты файла f ; |
| 4) <code>close(f)</code> | — закрытие файла f . |

После закрытия файла переменная f может быть связана с другим внешним файлом.

Так как значения переменных файлового типа хранятся на внешних носителях информации, в языке ПАСКАЛЬ **оператор присваивания файлов запрещен**.

C++

В языке C++ файл состоит из записей (компонентов), размер которых может быть фиксированным или переменным. Количество записей в файле ограничено только емкостью используемого физического носителя.

В зависимости от **типа операций, разрешенных для компонентов**, файлы классифицируются по следующим категориям:

- входные файлы (разрешено только чтение);
- файлы вывода (разрешена только запись);
- обновляемые файлы (разрешены как запись, так и чтение).

По **доступу к компонентам** файлы подразделяются на:

- файлы с последовательным доступом (записи могут обрабатываться только в том порядке, в котором они хранятся в файле);
- файлы с прямым доступом (компоненты можно обрабатывать в любом порядке). В этом случае перед каждой операцией чтения/записи указывается информация, необходимая для выбора компонента для обработки.

По **интерпретации содержимого** файлы делятся на:

- текстовые файлы, содержащие только символы, организованные по строкам;
- двоичные файлы, в которых информация рассматривается как набор байтов.

С файлами можно выполнять следующие **операции**:

- открытие файла;
- чтение данных из файла/запись данных в файл;
- закрытие файла.

В языке C++ обработка данных из внешних файлов осуществляется с помощью потоков.

Потоки — это объекты, которые транспортируют и форматируют строки байтов. В качестве примера упомянем уже известные нам потоки:

`cin` — поток, предназначенный для чтения данных с внешнего носителя информации стандартного устройства ввода, обычно с клавиатуры;

`cout` — поток, предназначенный для записи данных на внешний носитель информации стандартного устройства вывода, обычно на экран.

Программист также может создавать собственные потоки. Такие потоки должны быть ассоциированы/связаны с файлами на внешних устройствах, например с файлами на магнитных дисках или *флэш*-памяти. После осуществления такой ассоциации операция включения данных в поток приводит к их записи в связанный файл, а операция извлечения данных из потока — к чтению их из этого файла.

Напоминаем, что включение данных в поток осуществляется с помощью оператора `<<`, а извлечение данных — с помощью оператора `>>`.

Чтобы иметь доступ к функциям для работы с потоками и файлами в программы на C++ необходимо включить директиву:

```
#include <fstream>
```

Создание входного потока и его связывание с некоторым внешним файлом осуществляются с помощью следующей грамматической конструкции:

```
ifstream <Имя входного потока> ( "<Имя внешнего файла>" );
```

Очевидно, что внешний файл, из которого будут считываться данные, должен существовать на внешнем носителе в момент запуска программы на выполнение. В противном случае в процессе выполнения программы, при попытке связать входной поток с внешним файлом возникнет ошибка

Создание выходного потока и его связывание с внешним файлом осуществляются с помощью следующей грамматической конструкции:

```
ofstream <Имя выходного потока> ( "<Имя внешнего файла>" );
```

Если на момент связывания выходного потока с внешним файлом последний еще не существует, он будет создан автоматически. Если внешний файл уже существует, все записи в нем будут удалены.

Примеры:

1) `ifstream FI("C:\\RESULTAT\\R.DAT");`

- создание входного потока FI и его ассоциация с внешним файлом R.DAT из папки RESULTAT на диске C:.

2) `ofstream FE("D:\\D.CHR");`

- создание выходного потока FE и его ассоциация с внешним файлом D.CHR из корневого каталога диска D:.

3) `cout << ("Введите имя файла:");
cin << (Str);
ofstream FE(Str);`

- создание выходного потока FE. Поток FE связан с внешним файлом, имя которого считывается с клавиатуры и сохраняется в переменной типа string Str.

Подчеркнем тот факт, что в последнем примере имя внешнего файла неизвестно во время написания программы, оно читается с клавиатуры только во время выполнения программы. Такой подход позволяет разрабатывать программы, которые могут обрабатывать данные не только из определенного внешнего файла, но и из любого другого файла, указанного пользователем.

Переменные (потоки) FI и FE в вышеприведенных примерах также называются **логическими файлами**, или просто **файлами**. В отличие от других типов данных, значения которых хранятся во внутренней памяти компьютера, файловые данные хранятся на носителях информации периферийного оборудования (диски и магнитные ленты, оптические диски, *флэш*-память и т. д.). Информация на исследуемых носителях организована в виде **внешних файлов** в соответствии с требованиями операционной системы.

Подчеркнем, что, если пользователь не указывает полное имя внешнего файла (каталог, в котором он находится, и его фактическое имя), поиск файла/создание файла будет осуществляться в каталоге среды разработки программ C++, установленном по умолчанию.

Закрытие файлов, связанных с **потоками**, производится вызовом вида:

`<Имя потока>.close();`

Примеры:

`FI.close(); FE.close();`

После того как файл был закрыт, в него больше нельзя записывать и из него нельзя считывать данные. Однако эти операции становятся возможными после повторного открытия файла. Другими словами, запись или чтение данных, в зависимости от ситуации, возможны только в период, когда файл открыт.

Очевидно, что файл, который был ранее открыт, а затем закрыт, может быть повторно открыт с любым из потоков в разрабатываемой программе, не

обязательно только с тем, с которым он был ранее связан. Другими словами, потоки (логические файлы) могут использоваться для открытия любых внешних файлов, а внешние файлы могут быть открыты любым из потоков в программе.

Чтение данных из внешнего файла выполняется с помощью оператора извлечения данных из входного потока, ассоциированного с этим файлом:

```
<Имя потока> >> <Переменная> { >> <Переменная> };
```

Примеры:

1) FI >> x;

2) FI >> x >> y;

3) FI >> x >> y >> z;

Запись данных во внешний файл осуществляется с помощью оператора включения данных в выходной поток, связанный с соответствующим файлом:

```
<Имя потока> << <Выражение> { << <Выражение> };
```

Примеры:

1) FE << x;

2) FE << "x=" << x;

3) FE << x " " << y << " " << z;

В заключение представим порядок, в котором необходимо вызывать функции, предназначенные для обработки файлов.

1) **Ассоциация/связывание** потока f с внешним файлом s и его открытие для чтения/записи: `ifstream f(s)`, `ofstream f(s)`.

2) **Чтение/запись** компонента файла, связанного с потоком f : $f >> v$, соответственно $f << e$.

3) **Закрытие файла**, связанного с потоком f : `f.close()`.

После закрытия файла поток f может быть ассоциирован с другим внешним файлом.

Вопросы и упражнения

- 1) Объясните термины *логический файл*, *внешний файл*.
- 2) Где хранятся данные логического файла? А данные внешнего файла?
- 3) Как классифицируются файлы в зависимости от операций, которые с ними можно осуществлять и в зависимости от способа доступа к их компонентам?
- 4) Какие операции можно осуществлять с файлами? А с компонентами файла?
- 5) Для чего нужна операция связывания логического файла с внешним файлом?
- 6) Для чего нужны операции открытия и закрытия файлов? Как они осуществляются?
- 7) В каком порядке необходимо осуществлять операции обработки данных из входного файла? А из выходного файла?

PASCAL

- 8 УПРАЖНЯЙТЕСЬ! Нарисуйте структуру, подобную той, которая приведена на рисунке 1.8, для следующих типов данных:

a) **type** Tabel = **array** [1..5, 1..10] **of** real;
FisierTabele = **file of** Tabel;

b) **type** Multime = **set of** 'A'..'C';
FisierMultimi = **file of** Multime;

c) **type** Punct = **record** x, y:real **end**;
Segment = **record** A, B:Punct **end**;
FisierSegmente = **file of** Segment;

- 9 Для чего нужны процедуры read и write? Какого типа должна быть переменная v в вызове процедуры read(f, v)? Какого типа должно быть выражение e в вызове процедуры write(f, e)?

- 10 ПРОАНАЛИЗИРУЙТЕ! Даны следующие объявления:

```
type Numere = file of integer;  
var F, FIN, FOUT : Numere;  
    i, j, k : integer;  
    x, y, z : integer;
```

Объясните, какие операции будет выполнять программа в процессе выполнения следующих последовательностей операторов:

a) assign(F, 'EXP.DAT');
rewrite(F);
i:=10; write(F, i);
j:=20; write(F, j);
k:=30; write(F, k);
close(F);

b) assign(F, 'EXP.DAT');
reset(F);
read(F, x);
read(F, y);
read(F, z);
close(F);

c) assign(FOUT, 'EXP.DAT');
rewrite(FOUT);
i:=40; j:=50; k:=60;
write(FOUT, i); write(FOUT, j); write(FOUT, k);
close(FOUT);
assign(FIN, 'EXP.DAT');
reset(FIN);
read(FIN, x); read(FIN, y); read(FIN, z);
close(FIN);

- 11 В условиях предыдущего задания, какие значения примут переменные x , y , z , если последовательности операторов a), b) и c) будут выполняться одна за другой?

- 12 Переменные A и B объявлены следующим образом:

```
var A, B : file of integer;
```

Корректна ли следующая запись?

```
A:=B
```

Обоснуйте свой ответ.

C++

- 13 Как объявляются потоки данных? Как ассоциируется поток с некоторым внешним файлом?
- 14 ПРОАНАЛИЗИРУЙТЕ! Даны объявления:

```
int i, j, k,  
int x, y, z;
```

Объясните, какие операции программа будет выполнять в процессе выполнения следующих последовательностей инструкций:

- a)

```
ofstream F("EXP.DAT");  
i:=10; F << i << " ";  
j:=20; F << j << " ";  
k:=30; F << k;  
F.close();
```
- b)

```
ifstream F("EXP.DAT");  
F >> x;  
F >> y;  
F >> z;  
F.close();
```
- c)

```
ofstream FOUT("EXP.DAT");  
i:=40; j:=50; k:=60;  
FOUT << i << " " << j << " " << k;  
FOUT.close();  
ifstream FIN("EXP.DAT");  
FIN >> x >> y >> z;  
FIN.close();
```

- 15 ПРОАНАЛИЗИРУЙТЕ! В условиях предыдущего задания, какие значения примут переменные x, y, z, если последовательности операторов a), b) и c) будут выполняться одна за другой?
- 16 ПРОАНАЛИЗИРУЙТЕ! Переменные A и B объявлены следующим образом:

```
fstream A("text.in");  
ofstream B("test.out");
```

Корректен ли следующий оператор?

```
B=A
```

Обоснуйте свой ответ.

- ⑦ ИЗУЧИТЕ! Кроме входных (ifstream) и выходных потоков (ofstream), язык C++ предоставляет пользователям поток типа fstream. Создание потока такого типа и его связывание с внешним файлом осуществляются с помощью грамматической конструкции:

```
fstream<Имя входного/выходного потока> ("<Имя внешнего файла>" ,  
<Опция>);
```

где <Опция> указывает способ открытия потока. Используя систему поддержки среды разработки программ и интернет-источники, выясните назначение опций, которые можно использовать для связывания таких потоков с внешними файлами. Заполните таблицу:

Опция	Назначение
ios::in	
ios::out	
ios::app	
ios::ate	
ios::trunc	
ios::binary	

- ⑧ ИЗУЧИТЕ! В языке C++ есть несколько функций, предназначенных для проверки текущего состояния потока. Используя систему поддержки среды разработки программ и интернет-источники, выясните, какие результаты возвращают эти функции, и заполните таблицу:

Функция	Назначение
bad()	
fail()	
eof()	
good()	
clear()	

1.8. Файлы с последовательным доступом

PASCAL

Рассмотрим следующие описания:

```
type FT = file of T;  
var f : FT; v : T;
```

посредством которых определяются файловый тип FT с базовым типом T, файловая переменная *f* и переменная *v* типа T.

Для открытия **выходного файла** последовательного доступа применяют вызов процедуры *rewrite(f)*. Затем в файл записываются соответствующие компоненты. Запись отдельных компонент производится с помощью процедуры:

```
write(f, e),
```

где e — выражение типа T. Оператор вида

```
write(f, e1, e2, ..., en)
```

эквивалентен последовательности операторов

```
write(f, e1) ; write(f, e2); ...; write(f, en).
```

После записи последней компоненты файл необходимо закрыть.

Пример:

```
Program P90;  
{ Создание файла с компонентами типа Elev }  
type Elev = record  
    Nume : string;  
    Prenume : string;  
    NotaMedie : real;  
end;  
    FisierElevi = file of Elev;  
var FE : FisierElevi;  
    E : Elev;  
    str : string;  
    i, n : integer;  
begin  
    write('Введите имя создаваемого файла: ');  
    readln(str);  
  
    assign(FE, str);      { связывает FE с именем str }  
    rewrite(FE);          { открывает файл FE для записи }  
  
    write('Введите количество учеников: '); readln(n);  
  
    for i:=1 to n do  
        begin  
            writeln('Введите данные об ученике ', i);  
  
            { Считывает поля переменной E с клавиатуры }  
            write('Фамилия: ');      readln(E.Nume);  
            write('Имя: ');          readln(E.Prenume);  
            write('Средняя оценка: '); readln(E.NotaMedie);  
  
            { Записывает значение переменной E в файл FE }  
            write(FE, E);  
        end;  
    close(FE);            { Закрывает файл FE }  
    readln;  
end.
```

Для открытия **входного файла** последовательного доступа используется процедура `reset(f)`. Чтение текущего элемента из файла выполняют с помощью вызова процедуры:

```
read(f, v).
```

Оператор вида

```
read(f, v1, v2, ..., vn)
```

эквивалентен последовательности операторов

```
read(f, v1); read(f, v2); ..., read(f, vn).
```

Конец файла можно обнаружить с помощью булевой функции `eof(f)`, которая возвращает значение `true` после чтения последнего элемента.

Пример:

```
Program P90;
{ Считывание файла с элементами типа Elev }
type Elev = record
    Nume : string;
    Prenume : string;
    NotaMedie : real;
end;
FisierElevi = file of Elev;
var FE : FisierElevi;
    E : Elev;
    str : string;
begin
    write('Введите имя файла:');
    readln(str);

    assign(FE, str);    { связывает FE с именем str }
    reset(FE);          { открывает файл FE для чтения }

    while not eof(FE) do
        begin
            { считывает E из файла FE }
            read(FE, E);
            { выводит E на экран }
            writeln(E.Nume, ' ', E.Prenume, ': ',
                    E.NotaMedie : 5:2);
        end;
        close(FE);      { закрывает файл FE }
        readln;
    end.
```

Отметим, что число элементов файла заранее неизвестно и не задается в описании соответствующего типа. Следовательно, в выходной файл последовательного доступа теоретически можно записать бесконечное множество

элементов. Однако практически количество элементов ограничено емкостью внешнего носителя информации. Считывание элементов любого входного файла последовательного доступа завершается при достижении символа *EOF*.

C++

В случае файла с последовательным доступом его компоненты могут быть доступны только один за другим, начиная с первого. Следующая программа показывает, как создать файл с последовательным доступом.

```
// Создание файла с последовательным доступом
#include<iostream>    //для использования  cout
#include<fstream>    //для использования  ofstream
using namespace std;
int main()
{
    ofstream f("test.out");    // открывает файл для записи
    f << "Мой первый файл!" << endl; //записывает в файл
    cout << "Файл создан" << endl;
    return 0;
}
```

После запуска этой программы на экране монитора появится сообщение *Файл создан*, а в папке проекта мы найдем файл `test.out`, который сможем прочитать с помощью любого текстового редактора, например приложения *Notepad* (Блокнот) в операционной системе *Windows*. Вновь созданный файл также можно открыть с помощью команды *Open* (Открыть) в меню *File* среды разработки программ C++. Очевидно, что рассматриваемый файл будет содержать текст *Мой первый файл!*

Как подчеркивалось в предыдущем параграфе, в языке C++ файлы обрабатываются как байтовые последовательности. Следовательно, группировка данных из файлов в лексические единицы (целые числа, вещественные числа, строки символов и т. д.) является обязанностью программиста. Для более глубокого понимания этого подхода рассмотрим следующую практическую задачу, часто встречаемую в повседневной жизни.

Задача: Для каждого ученика класса известны следующие данные: имя, фамилия, средний балл по определенному школьному предмету. Эти данные необходимо хранить в файле с последовательным доступом и отобразить их на экране.

Решение: В следующей программе соответствующие данные считываются с клавиатуры, сохраняются в таблице `e[50]` и одновременно сохраняются в файле `elevi.in`.

```
// Программа P90
// Создания файла с данными об учениках
#include <iostream>
#include <fstream>
#include <cstring>
using namespace std;
```

```

int main()
{
    ofstream f("elevi.in");
    struct elev
    {
        char nume [15];
        char prenume[20];
        float NotaMedie;
    };
    int i,n;
    elev e[50];
    cout<<"Введите количество учеников ";
    cin>>n;
    cin.get();
    for (i=1;i<=n;i++)
    {
        cout<<"Введите фамилию ученика  "<<i<<" ";
        cin.get(e[i].nume,15);
        cin.get();
        cout<<"Введите имя ученика  "<<i<<" ";
        cin.get(e[i].prenume,20);
        cin.get();
        cout<<"Введите средний балл  ";
        cin>>e[i].NotaMedie;
        cin.get();
        f<<e[i].nume<<' '<<e[i].prenume<<' '<<e[i].NotaMedie<<endl;
    }
    f.close();
    return 0;
}

```

В этой программе извлечение фамилии и имени очередного ученика из входного потока `cin` или, другими словами, чтение значений переменных `Nume` и `Prenume` с клавиатуры выполняется с помощью функции `get`. Если список аргументов в вызове этой функции не пуст, она извлекает (считывает) из обработанного потока строку символов и помещает ее в эту переменную. В случае вызова без аргументов функция `get` просто перемещает курсор через один символ, в нашем случае это специальный символ *EOL*. Помните, что символы *EOL* появляются в потоке `cin` при нажатии клавиши *<ENTER>*.

Текущее значение переменной `NotaMedie` извлекается из входного потока `cin` с помощью оператора `>>`. Этот оператор считывает, начиная с текущей позиции курсора, вещественное число из обрабатываемого потока и помещает его в переменную `NotaMedie`. В результате считывания курсор переместится после последней цифры этого числа, в нашем случае перед символом *EOL*. Для того чтобы перевести курсор через этот символ, вызывается функция `get`, естественно, без аргументов.

Считанные с клавиатуры данные об ученике включаются (записываются) в выходной поток `f`, связанный с внешним файлом `elevi.in`.

Поскольку в языке C++ любой поток представляет собой последовательность символов, не имеющую определенной структуры, простого включения строк `Nume`, `Prenume` и вещественного числа `NotaMedie` в выходной поток будет недостаточно. Проблема в том, что любая программа, которая читает этот файл, не будет знать, где заканчивается строка символов `Nume` и где начинается строка символов `Prenume`. Чтобы решить эту проблему, в данной программе, в выходном потоке после каждого значения `Nume`, `Prenume` ставится пробел, а после каждого значения `NotaMedie` — специальный символ EOL (конец строки).

Очевидно, что после завершения процесса выполнения программы P90 данные в таблице `e[5]` будут потеряны, а данные во вновь созданном файле `elevi.in` будут сохранены на внешнем носителе информации. Считывание данных из этого файла и отображение их на экране можно выполнить с помощью следующей программы.

```
// Программа P91
// Считывание файла, содержащего данные об учениках
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
    ifstream f("elevi.in"); //открывается файл "elevi.in"
    struct elev
    {
        char nume [15];
        char prenume[20];
        float NotaMedie;
    };

    elev e;
    f>>e.nume>>e.prenume>>e.NotaMedie;
    while (!f.eof())
    {
        cout<<e.nume<<' '<<e.prenume<<' '<<e.NotaMedie<<endl;
        f>>e.nume>>e.prenume>>e.NotaMedie;
    }
    f.close(); //Закрывает файл
    return 0;
}
```

Данные из входного потока `f` извлекаются (читаются) до тех пор, пока функция `eof` не сообщит о достижении символа *EOF*, то есть о конце внешнего файла `elevi.in`.

Подчеркнем, что разработка программ последовательной обработки файлов возможна без предварительного знания количества компонентов в этих файлах. Теоретически количество компонентов, которые можно записать в файл последовательного вывода, не ограничено. На практике, однако, это количество ограничено емкостью внешнего носителя информации. В случае входных файлов программа должна закончить чтение компонентов из файла последовательного доступа в момент достижения элемента *EOF*.

Вопросы и упражнения

- ❶ Из скольких компонентов может состоять файл? В каком порядке записываются и считываются компоненты файла с последовательным доступом?
- ❷ УПРАЖНЯЙТЕСЬ! Даны следующие типы данных:

ПАСКАЛЬ

```
type Data = record
    Ziua : 1..31;
    Luna : 1..12;
    Anul : integer;
end;
Persoana = record
    NumePrenume : string;
    DataNasterii : Data;
end;
FisierPersoane = file of
Persoana;
```

C++

```
struct Data
{
    int Ziua, Luna, Anul;
};
struct elev
{
    char NumePrenume [40];
    Data DataNasterii;
};
```

Напишите программу, которая считывает с клавиатуры данные об n лицах (фамилия, имя, дата рождения) и записывает их в файл. Создайте файлы: FILE1.PRS, FILE2.PRS, FILE3.PRS, в которых должны содержаться данные соответственно 2, 7 и 10 человек.

- ❸ РЕШИТЕ! Напишите программу, которая читает файлы, созданные программой из предыдущего упражнения, и выводит на экран:
 - а) данные о всех лицах, внесенных в файл;
 - б) данные о лицах, родившихся в год a ;
 - в) данные о лицах, у кого дата рождения $z.l.a$;
 - г) данные о самом старшем человеке;
 - д) данные о самом младшем человеке.
- ❹ РЕШИТЕ! Напишите программу, которая выводит на экран среднее арифметическое чисел, записанных в файле с вещественными числами.
- ❺ РЕШИТЕ! В файле записаны произвольные символы. Напишите программу, которая выводит на экран количество согласных, содержащихся в файле.
- ❻ ОБРАТИТЕ ВНИМАНИЕ! Прокомментируйте программу:

ПАСКАЛЬ

```
Program P92;
{ Ошибка }
type
    FisierNumere = file of
integer;
var FN : FisierNumere;
    i : integer;
    r : real;
    s : string;
begin
    Writeln('Имя файла: ');
    readln(s);
```

C++

```
// Программа P92
// Ошибка
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
    char s[10];
    cout<<"Имя файла: \n";
    cin>>s;
    ifstream FN(s);
    int i;
```

```

assign(FN, s);
rewrite(FN);
i:=1;
write(FN, i);
i:=10;
write(FN, i);
r:=20;
write(FN, r);
close(FN);
end.

```

```

float r;
i=1;
FN<<i;
i=10;
FN<<i;
r=20;
FN<<r;
FN.close();
return 0;
}

```

1.9. Текстовые файлы

Известно, что данные логических файлов в программах, написанных на языке высокого уровня, представлены последовательностями двоичных цифр. Такой способ представления данных удобен для внешних запоминающих устройств (магнитных дисков и лент, оптических дисков, *флэш*-памяти и др.). В случае устройств ввода-вывода (клавиатура, экран, принтер и т. д.) данные должны быть представлены в понятной человеку форме, т.е. последовательностями символов.

Чтобы облегчить взаимодействие между человеком и компьютером, на языках высокого уровня информация, предназначенная для пользователя, представлена в виде *текстовых файлов*. Текстовый файл состоит из последовательности символов, разделенных на строки (рис. 1.9). Длина строк варьирует. Конец каждой строки обозначается специальным элементом, обозначенным EOL (*End Of Line*). Поскольку длина строк является переменной, положение строки в любом текстовом файле невозможно рассчитать заранее. Следовательно, доступ к компонентам текстовых файлов является *последовательным*. Другими словами, запись или чтение компонента, то есть символа или строки, возможны только после того, как предыдущий компонент был прочитан или записан.

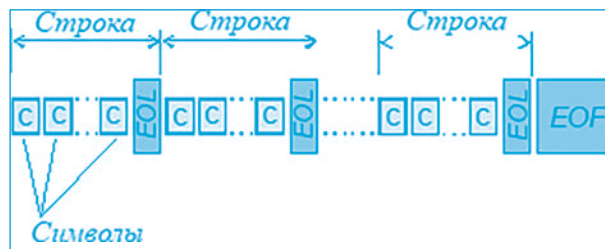


Рис. 1.9. Структура текстового файла

В рамках лицейской учебной дисциплины «Информатика» изучаются следующие способы записи/чтения *текстовых файлов*:

- на уровне символа;
- на уровне лексемы;
- на уровне строк.

В случае обработки на уровне символов данные во внешнем файле читаются / записываются посимвольно, каждый из символов сохраняется в переменной типа `char`. Объединение этих символов в числа или строки является обязанностью программиста.

Программист также должен позаботиться об обработке специальных элементов *EOL* (*end of line* — конец строки).

В случае **обработки на уровне лексем** данные записываются и читаются с использованием функций и процедур, которые «собирают» данные из внешнего файла в числа и строки и сохраняют их, в зависимости от ситуации, в переменных целого, вещественного, строкового типа. Очевидно, что рассматриваемые лексемы должны быть разделены белыми символами (пробел, табуляция или конец строки).

Например, предположим, что входной файл содержит следующие данные:

```
82 3.14 ПРИНЯТ <EOL><EOF>
```

Интуитивно понятно, что входной файл содержит целое число 82, вещественное число 3,14 и строку символов *ПРИНЯТ*.

Обработка этого файла на уровне символов заключается в последовательном чтении каждого из символов, в нем содержащихся:

```
8 2 3 . 1 4 П Р И Н Я Т
```

и помещая их в переменные типа `char`, объявленные в программе. Далее программист должен будет дополнительно написать последовательность операторов, преобразующих считанные символы в нужные ему значения.

Обработка того же файла на уровне лексем включает чтение символов во входном файле и их группировку в три лексемы:

```
82 3.14 ПРИНЯТ
```

Впоследствии полученные таким образом лексемы автоматически преобразуются во внутреннее представление и хранятся в переменных соответствующего типа: `integer/int`, `real/float` и *строка символов*.

В случае **обработки на уровне строк** данные во внешнем файле читаются/записываются построчно, причем каждая строка сохраняется в переменной тип *строка символов*.

Обработка на уровне символов полезна при программировании алгоритмов редактирования текстов, например при построении последовательности слов и предложений, делении слов на слоги и т. д., а обработка на уровне лексем — в случае программирования алгоритмов обработки преимущественно числовых данных. Обработка на уровне строк дает пользователю возможность манипулировать как отдельными символами, так и лексемами, используя предопределенные функции для обработки строк.

PASCAL

В языке ПАСКАЛЬ текстовый файл задается описанием вида:

```
var f : text;
```

где предопределенный тип `text` известен любой программе на языке ПАСКАЛЬ. Отметим, что типы `text` и `file of char` различны, так как файл

file of char не содержит символов *EOL*. Чтобы в этом убедиться, достаточно сравнить соответствующие структуры, представленные на *рис.1.8* и *рис. 1.9*.

Обработка файлов типа *text* может осуществляться с помощью известных процедур, применимых к любым типам файлов: *assign*, *reset*, *rewrite*, *read*, *write*, *close*. В дополнение к ним в языке есть специальные процедуры для обработки элементов *EOL*:

writeln(f)— записывает в файл элемент *EOL* (конец строки);

readln(f)— переход на следующую строку.

Конец строки определяется с помощью булевой функции *eofln(f)*, которая принимает значение *true* после считывания последнего символа строки.

Оператор вида

```
writeln(f, e1, e2, ..., en)
```

эквивалентен последовательности операторов:

```
write(f, e1, e2, ..., en); writeln(f).
```

Оператор вида

```
readln(f, v1, v2, ..., vn)
```

эквивалентен последовательности операторов:

```
read(f, v1, v2, ..., vn); readln(f).
```

Для ввода и вывода данных используются, как правило, предопределенные текстовые файлы *Input* и *Output*, известные любой программе на языке ПАСКАЛЬ. Файл *Input* предназначен только для чтения и связан со стандартным устройством ввода операционной системы (как правило, с клавиатурой). Файл *Output* предназначен только для записи и связан со стандартным устройством вывода (как правило, экраном). Эти файлы открываются и закрываются автоматически в начале и соответственно в конце выполнения программы. Если при вызове процедур ввода/вывода имя файла не указано в списке параметров, то предполагается, что *текстовым файлом* является файл *Input* или файл *Output*, в зависимости от ситуации. Например, *read(c)* эквивалентен *read(Input, c)*, а *write(c)* эквивалентен *write(Output, c)*.

Для примера представим программу, которая создает внешний файл *LISTA.TXT*, содержащий фамилию и имя *n* учеников, количество которых неизвестно на момент написания программы.

```
Program P93;  
{ Создание текстового файла LISTA.TXT способом }  
{ обработки на уровне строк }  
var F : text;      { логический файл }  
    n : integer;    { количество учеников }  
    NP : string;    { фамилия и имя ученика }  
    i : integer;    { счетчик цикла }
```

```

begin
  { связывание логического файла F с внешним файлом }
  assign(F, 'LISTA.TXT');
  { Открытие файла F для записи }
  rewrite(F);
  { Считывание с клавиатуры количества учеников }
  write('Введите количество учеников n=');
  readln(n);
  { Запись количества учеников в файл F }
  writeln(F, n);
  { цикл по количеству учеников }
  for i:=1 to n do
    begin
      { Считывание с клавиатуры фамилии и имени }
      writeln('Введите фамилию и имя ученика ', i, ':');
      readln(NP);
      {Запись фамилии и имени в файл F }
      writeln(F, NP);
    end;
  { Закрытие файла F }
  close(F);
  writeln('Были введены данные ', n, ' учеников');
  readln;
end.

```

В продолжение предыдущего примера представим программу, считывающую данные из внешнего файла LISTA.TXT и отображающую их на экране.

```

Program P94;
{ Считывание текстового файла LISTA.TXT способом }
{ обработки на уровне строк }
var F : text;          { логический файл F }
    n : integer;       { количество учеников }
    NP : string;       { фамилия и имя }
    i : integer;       { счетчик цикла }
begin
  { Связывание логического файла F с внешним файлом }
  assign(F, 'LISTA.TXT');
  { Открытие файла F для чтения }
  reset(F);
  { Считывание количества учеников n из файла F }
  readln(F, n);
  { Отображение на экране количества учеников n }
  writeln('Количество учеников n=', n);
  { Цикл по количеству учеников }
  for i:=1 to n do

```

```

begin
  readln(F, NP); { считывание NP из файла F }
  writeln(NP);   { отображение NP на экране }
end;
close(F);        { закрытие файла F          }
readln;
end.

```

Обработка на уровне символов

Текстовые файлы могут быть считаны и записаны посимвольно. Для этой цели переменная *v* в вызываемых процедурах `read(f, v)` и `write(f, v)` должна быть типа `char`. Очевидно, что вставка специальных символов *EOL* и *EOF* является обязанностью программиста.

В качестве примера приведем программу P95, которая создает на текущем диске текстовый файл `FILE.TXT`. Строки файла вводят с клавиатуры (файл `Input`). Признаком конца строки является нажатие клавиши `<ENTER>`, а признаком конца файла является нажатие клавиш `<CTRL+Z>`, `<ENTER>`.

```

Program P95;
{ Создание текстового файла FILE.TXT методом обработки }
{ на уровне символов                                     }
var F : text;      { логический файл F                  }
    C : char;
begin
  assign(F, 'FILE.TXT'); { связывает файл F с FILE.TXT }
  rewrite(F);           { открывает F для записи        }
  while not eof do      { проверяет, нажаты ли клавиши <CTRL+Z> }
  begin
    while not eoln do { проверяет, нажата ли клавиша <ENTER> }
    begin
      read(C);        { считывает C с клавиатуры        }
      write(F, C);    { записывает C в файл F            }
    end;
    writeln(F);       { записывает EOL в файл F          }
    readln;           { переходит на следующую строку   }
  end;
  close(F);           { записывает EOF в файл F          }
end.

```

Отметим, что в вышеприведенной программе компоненты входного файла `Input` (клавиатура) считываются посимвольно, а конец каждой строки (специальный символ *EOL*) распознается функцией `eoln`. Аналогично конец входного файла (специальный символ *EOF*) распознается функцией `eof`.

Следующая программа отображает на экране содержимое внешнего файла `FILE.TXT`.

```

Program P96;
{ Считывание текстового файла FILE.TXT путем обработки }
{ на уровне символов }
var F : text;           { логический файл F }
      C : char;
begin
  assign(F, 'FILE.TXT'); { связывает файл F с FILE.TXT }
  reset(F);              { открывает F для чтения }
  while not eof(F) do    { проверяет, есть ли EOF в F }
    begin
      while not eoln(F) do { проверяет, есть ли EOL в F }
        begin;
          read(F, C);      { считывает C из F }
          write(C);        { отображает на экране C }
        end;
        readln(F);        { переходит на следующую строку в F }
        writeln;          { переходит на следующую строку на экране }
      end;
      close(F);           { закрывает файл F }
      readln;
    end.

```

В вышеприведенной программе компоненты выходного файла Output (экран) записываются на уровне символов. Конец *EOL* каждой строки записывается с помощью вызова процедуры `readln`, а конец файла *EOF* — с помощью вызова процедуры `close`.

Обработка на уровне лексем

Посимвольная обработка текстовых файлов является обременительной, если символьные последовательности из текста следует интерпретировать как данные типа `integer`, `real`, `boolean`, *строка символов*. Конверсия из внешней формы этих данных во внутреннее представление соответственно их типам является ответственностью программиста. Чтобы облегчить ему работу, процедуры чтения/записи были расширены следующим образом.

В случае текстовых файлов переменная v в вызове `read(f, v)` может быть целого, вещественного, символьного или строкового типа. При чтении последовательность символов, представляющая переменную v , будет преобразована во внутреннее представление соответствующего типа.

За выражением e в процедуре `write(f, e)` может следовать указатель формата. Значение выражения может относиться к типу `integer`, `real`, `char` или *строка символов*. При записи соответствующее значение переводится из внутреннего представления в последовательность символов.

Последовательности символов, считываемые/записываемые с помощью процедур `read/write`, соответствуют синтаксису констант типа переменной/выражения v/e .

В качестве примера приведем программу P97, которая:

- 1) считывает с клавиатуры строку, содержащую три вещественных числа a , b , c — длины сторон треугольника;
- 2) записывает считанные числа в файл `IN.TXT`;
- 3) повторяет шаги 1) и 2) до момента нажатия пользователем клавиш `<CTRL+Z>` `<ENTER>`;

- 4) считывает из текущей строки файла IN.TXT вещественные числа a , b , c ;
- 5) вычисляет полупериметр p и площадь s соответствующего треугольника;
- 6) записывает вещественные числа a , b , c , p , s в текстовый файл OUT.TXT;
- 7) повторяет шаги 4)—6) до тех пор, пока не будет считана последняя строка текстового файла OUT.TXT;
- 8) отображает на экране содержимое файла OUT.TXT.

Program P97;

```
{ Обработка треугольников }
{ Запись и считывание на уровне лексем }
var F, G : text;           { логические файлы }
    a, b, c, p, s : real;
    Str : string;
begin

    { Считывание данных с клавиатуры и их сохранение в файле F }
    assign(F, 'IN.TXT'); { связывает F с IN.TXT }
    rewrite(F);          { открывает F для записи }
    writeln('Введите вещественные числа a, b, c:');
    while not eof do
        begin
            readln(a, b, c); { считывает с клавиатуры a, b, c }
            writeln(F, a:8:2, b:8:2, c:8:2); { scribe a, b, c in F }
        end;
    close(F);              { закрывает F }

    { Считывание данных из F, выполнение вычислений и }
    { сохранение результатов в G }
    reset(F);              { открывает F для считывания }
    assign(G, 'OUT.TXT'); { связывает G с OUT.TXT }
    rewrite(G);            { открывает G для записи }
    while not eof(F) do
        begin
            readln(F, a, b, c); { считывает a, b, c из F }
            write(G, a:8:2, b:8:2, c:8:2); { записывает a, b, c в G }
            p:=(a+b+c)/2;
            s:=sqrt(p*(p-a)*(p-b)*(p-c));
            writeln(G, p:15:2, s:15:4); { scribe p, s in G }
        end;
    close(F);              { закрывает F }
    close(G);              { закрывает G }

    { Считывание результатов из G и отображение их на экране }
    reset(G);              { открывает G для считывания }
    while not eof(G) do
        begin
            readln(G, Str); { считывает Str из G }
            writeln(Str);   { отображает Str на экране }
        end;
    close(G);              { закрывает G }
    readln;
end.
```

При вводе данных

```
1 1 1 <ENTER>
3 4 6 <ENTER>
<CTRL+Z><ENTER>
```

программа P97 отобразит на экране:

```
1.00 1.00 1.00 1.50 0.4330
3.00 4.00 6.00 6.50 5.3327
```

Отметим, что вышеприведенная программа предназначена для обработки данных о неизвестном количестве треугольников, что придает ей большую гибкость для применения. Такая гибкость обусловлена использованием функции `eof`. С помощью этой функции проверяется, был ли в процессе считывания достигнут конец каждого из файлов `Input` (клавиатура) и `IN.TXT`.

C++

В C++ текстовый файл содержит только символы ASCII, расположенные в строках разной длины. Элемент *EOL* представлен символом *escape* `"\n"` (new line, новая строка). Конец файла указывается элементом *EOF*. Поскольку длина строк является переменной, положение строки в файле невозможно рассчитать заранее. В результате доступ к компонентам *текстового файла* является последовательным.

Поскольку язык C++ не навязывает никакой структуры содержимому файлов, разбиение их на лексемы (числа, строки символов) и на строки является обязанностью программиста. Для этого могут быть использованы все функции, предназначенные для обработки файлов с последовательным доступом, рассмотренные в параграфе 1.8.

В качестве примера представлена программа P93, которая создает на текущем диске *текстовый файл*, имя которому задается с клавиатуры. Строки файла разделяют нажатием клавиши `<ENTER>`, а конец файла нажатием клавиш `<CTRL + Z>` `<ENTER>`.

```
// Программа P93
// Создание текстового файла
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
    char nume_fisier[80];
    char c;
    cout << "\n\tПрограмма создает текстовый файл\n";
    cout << "\n\tВведите имя файла: ";
    cin.getline(nume_fisier, 80);
    ofstream f(nume_fisier);
    cout << "\n\tВводите символы. В конце строк нажимайте En-
ter, в конце ввода CTRL+Z\n";
    while(cin.get(c))
        f << c;
    f.close();
    return 0;
}
```

Обработка на уровне символов

Напомним, что при считывании оператором >> игнорируются *белые символы* (пробел, табуляция, конец строки). Поэтому, если хотим прочитать один за другим все символы в файле, включая белые, мы воспользуемся функцией `get()`.

```
// Программа P94
// Считывание из текстового файла
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
    char nume_fisier[80];
    char c ;
    cout << "\n\tПрограмма считывает текстовый файл\n";
    cout << "\n\tВведите имя файла: ";
    cin.getline(nume_fisier,80);
    ifstream f(nume_fisier);
    f.get(c);
    while(!f.eof())
    {
        f.get(c);
        cout<<c;
    }
    f.close();
    return 0;
}
```

Чтобы проверить, достигла ли конца файла текущая позиция чтения/записи, применяют функцию `eof()`, которая возвращает значение 0, если текущая позиция не находится в конце файла, и значение, отличное от 0, если текущая позиция указывает на конец файла.

Нижеследующая программа копирует символьные данные из *текстового файла* IN.TXT в *текстовый файл* OUT.TXT.

```
// Программа P95
// Посимвольное копирование текстовых файлов
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
    char c ;
    ifstream f("in.txt"); // поток f связывается с in.txt
    ofstream g("out.txt"); // поток g связывается с out.txt
    f.get(c); // из f считывается первый символ
    while(!f.eof()) // цикл по символам из f
    {
```

```

        g.put(c);           // считанный символ записывается в g
        f.get(c);           // из f считывается следующий символ
    }
f.close();                  // закрытие файла in.txt
g.close();                  // закрытие файла out.txt
return 0;
}

```

Обработка на уровне строк

Для построчного считывания символов из входного файла, включая белые символы, применяется функция `getline`.

Следующая программа копирует строку за строкой данные из *текстового файла* IN.TXT в *текстовый файл* OUT.TXT.

```

// Программа P96
// Построчное копирование текстовых файлов
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
    char linie[80];
    ifstream f("in.txt");           // поток f связывается с in.txt
    ofstream g("out.txt");           // поток g связывается с out.txt
    while(!f.eof())                  // цикл по строкам из f
    {
        f.getline(linie,80);         // из f считывается текущая строка
        g<<linie<<endl;              // считанная строка записывается в g
    }
    f.close();                       // закрытие файла in.txt
    g.close();                       // закрытие файла out.txt

    return 0;
}

```

Обработка на уровне лексем

Нижеследующая программа применяет функции обработки файлов C++ для их разбиения на лексемы. Точнее, эта программа:

- 1) считывает с клавиатуры строку, содержащую вещественные числа a , b , c — длины сторон треугольника;
- 2) записывает считанные из текущей строки числа в текстовый файл `in.txt`;
- 3) повторяет шаги 1)–2) до тех пор, пока не будут нажаты клавиши `<CTRL+Z>` `<ENTER>`;
- 4) считывает из текущей строки *текстового файла* `in.txt` вещественные числа a , b , c ;
- 5) вычисляет полупериметр p и площадь s соответствующего треугольника;
- 6) записывает вещественные числа a , b , c , p , s в текстовый файл `out.txt`;

7) повторяет шаги 4)–6) до тех пор, пока не считается последняя строка текстового файла out.txt;

8) отражает на экране содержимое файла out.txt.

```
//Программа P97
//Обработка треугольников
//Запись и считывание на уровне лексем
#include <iostream>
#include <iomanip>
#include <cmath>
#include <fstream>
using namespace std;
int main()
{
    float a,b,c;
    float p,s;
    int i,n;
    char linie[80];

    //Считывание данных с клавиатуры и их запись в f
    ofstream f("in.txt"); //открывает f для записи
    cout<<"открывает f для записи a,b,c \n";
    cin>>a>>b>>c; // считывает a, b, c с клавиатуры
    while (!cin.eof()) //повторяет до нажатия с клавиатуры <CTRL+Z>
        <ENTER>
    {
        f<<a<<" "<<b<<" "<<c<<endl; // пишет a, b, c в файл f
        cin>>a>>b>>c; // считывает a, b, c с клавиатуры
    }
    f.close(); // закрывает f

    //Считывание данных из f, проведение вычислений и
    // сохранение результатов в g
    ifstream f("in.txt"); // открывает f для чтения
    ofstream g("out.txt"); // открывает g для записи
    f>>a>>b>>c; // считывает a,b,c из f
    while (!f.eof())
    {
        g<<a<<" "<<b<<" "<<c<<" "; // запись a,b,c в g
        p=(a+b+c)/2;
        s=sqrt(p*(p-a)*(p-b)*(p-c));
        g<<p<<" "<<s<<"\n"; // запись p и s в g
        f>>a>>b>>c; // считывание a,b,c из f
    }
    f.close(); // закрывает f
    g.close(); // закрывает g
}
```

```
// Считывание результатов из g и их отображение на экране
ifstream g("out.txt"); // открывает g для считывания
while(!g.eof())
{
    g.getline(linie,80); // считывает текущую строку из g
    cout<<linie<<endl; // отображает на экране считанную строку
}
g.close(); // закрывает g
return 0;
}
```

Для входных данных

```
1 1 1 <ENTER>
3 4 6 <ENTER>
<CTRL+Z><ENTER>
```

программа P97 отобразит на экране:

```
1 1 1 1.5 0.433013
3 4 6 6.5 5.332684
```

Подчеркнем, что вышеприведенная программа предназначена для обработки данных в отношении неизвестного количества треугольников. Это придает программе гибкость в ее применении. Достигается эта гибкость благодаря применению функции `eof`. С помощью этой функции проверяется, были ли считаны до конца данные из обоих файлов `Input` (клавиатура) и `IN.TXT`.

Вопросы и упражнения

- 1 УПРАЖНЯЙТЕСЬ! Нарисуйте схематическое представление структуры *текстового файла*. Объясните назначение каждого графического значка соответствующей схемы.
- 2 ПРОАНАЛИЗИРУЙТЕ! Дан следующий текстовый входной файл:

```
Данные 9.8 12 23.67 СТОП<EOL><EOF>
```

Объясните, каким образом считаются данные из этого файла и какие при этом произойдут преобразования для случаев обработки: (а) на уровне символов; (b) на уровне лексем.

- 3 ПРОАНАЛИЗИРУЙТЕ! Дан следующий текстовый входной файл:

```
5.1 9.3 Отклонен<EOL>6.4 4.3 Допущен <EOL><EOF>
```

Объясните, каким образом считаются данные из этого файла и какие при этом произойдут преобразования для случаев обработки: (а) на уровне символов; (b) на уровне лексем.

- 4 СОЗДАВАЙТЕ! Приведите примеры входных файлов с начальными данными задачи по физике и задачи по биологии и опишите процесс считывания этих данных на уровне символов и на уровне лексем.

- ⑤ ПРОАНАЛИЗИРУЙТЕ! Чем отличаются способы обработки текстового файла на уровне символов и на уровне лексем? Приведите примеры алгоритмов, в которых возникает необходимость реализации обработки на уровне символов и на уровне лексем.
- ⑥ УЧИТЕСЬ УЧИТЬСЯ! Исходя из двух способов считывания входных файлов на уровне символов и на уровне лексем, объясните эти способы для выходных текстовых файлов. Приведите примеры.
- ⑦ ОБЪЯСНИТЕ! Объясните значение элементов *EOL* и *EOF*. Каким образом обрабатываются эти элементы?
- ⑧ ОБЪЯСНИТЕ! Объясните, как записываются и считываются данные в случае текстовых файлов. Какие процедуры/функции применяют для этого?
- ⑨ ОБРАТИТЕ ВНИМАНИЕ! Запустите на выполнение следующую программу:

П А С К А Л Ь

```

Program P98;
{ Связывание файла FN с
  устройством ввода }
type FisierNumere = file of
integer;
var FN : FisierNumere;
    i : integer;
begin
  assign(FN, 'CON');
  rewrite(FN);
  i:=1;
  write(FN, i);
  i:=2;
  write(FN, i);
  i:=3;
  write(FN, i);
  close(FN);
  readln;
end.

```

C ++

```

// Программа P98
// Связывание файла FN с
// устройством ввода
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
  int i;
  ofstream FN („CON");
  i=1;
  FN<<i;
  i=2;
  FN<<i;
  i=3;
  FN<<i;
  FN.close();
  return 0;
}

```

Объясните результаты, выводимые на экран.

- ⑩ ПРОАНАЛИЗИРУЙТЕ! Дан логический файл f. Какие из нижеследующих операторов, предназначенные для обнаружения конца файла, записаны корректно?

П А С К А Л Ь

- a) **if** eoln(f) **then**
 write ('Конец файла')
else write ('Не конец файла');
- b) **if** eof(f) **then**
 write ('Конец файла')
else write ('Не конец файла');
- c) **if** eoln(f)=false **then**
 write ('Конец файла')
else write ('Не конец файла');
- d) **if not** eof(f) **then**
 write ('Конец файла')
else write ('Не конец файла');

- a) `if (eof(f))`
 `cout << "Конец файла";`
`else cout << "Не конец файла";`
- b) `if (f.eof())`
 `cout << "Конец файла";`
`else cout << "Не конец файла";`
- c) `if (!eof(f))`
 `cout << "Конец файла";`
`else cout << "Не конец файла";`
- d) `if (!f.eof())`
 `cout << "Не конец файла";`
`else cout << "Конец файла";`

- ❶ ПРИМЕНИТЕ! Напишите программу, которая выводит на экран содержание любого текстового файла.
- ❷ ПРОГРАММИРУЙТЕ! Напишите программу, которая выводит на экран количество гласных, содержащихся в текстовом файле.
- ❸ ПРОГРАММИРУЙТЕ! Входные данные некоторой программы записаны в *текстовый файл*. В каждой строке файла содержатся два целых и три вещественных числа, разделенные пробелами. Напишите программу, которая выводит на экран сумму целых и сумму вещественных чисел из каждой строки.
- ❹ ПРОГРАММИРУЙТЕ! Входные данные некоторой программы записаны в *текстовый файл*. В каждой строке файла содержатся по три вещественных числа, разделенных пробелами, и по одному из слов ДОПУЩЕН, ОТКЛОНЕН. Напишите программу, которая:
- a) выводит содержимое данного файла на экран;
 - б) создает резервную копию файла;
 - в) создает *текстовый файл*, строки которого содержат среднее арифметическое трех вещественных чисел, взятых из соответствующих строк входного файла;
 - г) выводит на экран строки входного файла таким образом, что перед каждой строкой пишется ее порядковый номер: 1, 2, 3 и т. д.
- ❺ ПРОГРАММИРУЙТЕ! Каждая строка *текстового файла* содержит следующие данные, разделенные пробелами:
- порядковый номер (целое число);
 - фамилия (последовательность символов, не содержащая пробелов);
 - имя (последовательность символов, не содержащая пробелов);
 - отметка по 1-му школьному предмету (вещественное число);
 - отметка по 2-му школьному предмету (вещественное число);
 - отметка по 3-му школьному предмету (вещественное число).
- Напишите программу, которая:
- a) создает резервную копию текстового файла;
 - б) выводит на экран содержимое файла;
 - в) создает текстовый файл, строки которого содержат следующие данные, разделяемые пробелами:
 - порядковый номер (целое число);
 - фамилия (последовательность символов, не содержащая пробелов);
 - имя (последовательность символов, не содержащая пробелов);
 - средний балл (вещественное число);

Файл, созданный в пункте в, необходимо вывести на экран.

2.1. Количество информации

Обычный смысл слова **информация** — «новости, устное, письменное или переданное другими способами сообщение об определенных фактах, событиях, деятельности и т. п.», — конкретизируется в специальном разделе математики, который называется **теорией информации**. В соответствии с этой теорией **источник информации** описывается переменной S , которая может принимать значения из конечного множества различных элементов $\{s_1, s_2, \dots, s_n\}$. Предполагается, что текущие значения переменной S априори не известны. Известно только множество $\{s_1, s_2, \dots, s_n\}$, называемое **множеством возможных сообщений**.

Например, дорожный светофор можно рассматривать как источник информации, множество возможных сообщений которого: {зеленый, желтый, красный}. Телеграфный аппарат представляет собой источник информации, множество возможных сообщений которого включает буквы A, B, C, \dots, Z , цифры $0, 1, 2, \dots, 9$ и знаки препинания. Возможными сообщениями клавиатуры являются «Нажата клавиша A », «Нажата клавиша B », ..., «Нажата клавиша $F1$ », «Нажата клавиша $F2$ », ..., «Одновременно нажаты клавиши $CTRL$ и $BREAK$ » и т. д.

Сообщения передаются от источника к приемнику информации через физическую среду, называемую **каналом передачи** (рис. 2.1). Например, телеграфные сообщения передаются по проводам, радиосообщения передаются через эфир, сообщения клавиатуры — через группу проводников. **Помехи** (шумы) упомянутой физической среды могут искажать передаваемые сообщения.

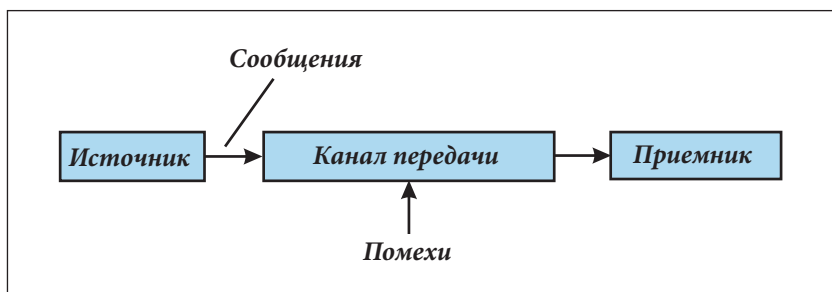


Рис. 2.1. Общая схема системы передачи информации

Очевидно, что текущее значение переменной S становится известным приемнику только после приема соответствующего сообщения.

Количество/объем информации I , которое содержится в сообщении, переданном источником, определяется соотношением:

$$I = \log_a n,$$

где n — это количество возможных сообщений источника. Конкретное значение константы a устанавливается выбором **единицы измерения количества информации**. Обычно в качестве единицы измерения используется **бит**.

Бит — это количество информации в отдельном сообщении от источника с множеством только из двух возможных сообщений.

Следовательно, как и в случае других величин (длины, массы, температуры и т. д.), количество информации измеряется сравнением с **эталоном**. Так как для эталонного источника $n = 2$, из уравнения:

$$\log_a 2 = 1 \text{ (бит)}$$

получаем $a = 2$. Следовательно, количество информации I , измеряемое в битах, определяется соотношением:

$$I = \log_2 n \text{ (бит)}.$$

В таблице 2.1 представлены часто используемые значения функции $\log_2 n$.

Таблица 2.1

Значения функции $\log_2 n$

n	$\log_2 n$	n	$\log_2 n$
1	0,000	21	4,392
2	1,000	22	4,459
3	1,585	23	4,524
4	2,000	24	4,585
5	2,322	25	4,644
6	2,585	26	4,700
7	2,807	27	4,755
8	3,000	28	4,807
9	3,170	29	4,858
10	3,322	30	4,907
11	3,459	31	4,954
12	3,585	32	5,000
13	3,700	33	5,044
14	3,807	34	5,087
15	3,907	35	5,129
16	4,000	36	5,170
17	4,087	37	5,209
18	4,170	38	5,248
19	4,248	39	5,285
20	4,322	40	5,322

Проанализируем несколько примеров. Количество информации в одном сообщении светофора:

$$I = \log_2 3 \approx 1,585 \text{ бит}.$$

Количество информации одной буквы латинского алфавита $\{A, B, C, \dots, Z\}$, $n = 26$, составляет:

$$I = \log_2 26 \approx 4,700 \text{ бит}.$$

Количество информации одной буквы греческого алфавита $\{A, B, \Gamma, \Delta, \dots, \Omega\}$, $n = 24$, составляет:

$$I = \log_2 24 \approx 4,585 \text{ бит}.$$

Если известно количество информации I , содержащееся в отдельном сообщении, то общее **количество информации, переданное источником**, определяется соотношением:

$$V = N I,$$

где N — количество переданных сообщений.

Большие объемы информации выражаются с помощью единиц, производных от бита:

$$1 \text{ Килобит (Kbit)} = 2^{10} = 1\,024 \text{ бит } (\approx 10^3 \text{ бит});$$

$$1 \text{ Мегабит (Mbit)} = 2^{20} = 1\,048\,576 \text{ бит } (\approx 10^6 \text{ бит});$$

$$1 \text{ Гигабит (Gbit)} = 2^{30} \approx 10^9 \text{ бит};$$

$$1 \text{ Терабит (Tbit)} = 2^{40} \approx 10^{12} \text{ бит};$$

$$1 \text{ Петабит (Pbit)} = 2^{50} \approx 10^{15} \text{ бит}.$$

Вопросы и упражнения

- ❶ Как определяется источник информации? Приведите несколько примеров.
- ❷ Для чего предназначен канал передачи?
- ❸ Как определяется количество информации в одном сообщении? В N сообщениях?
- ❹ Какая единица используется для измерения количества информации и в чем ее смысл?
- ❺ Определите количество информации в отдельном сообщении источников со следующими возможными сообщениями:
 - а) прописные и строчные буквы латинского алфавита;
 - б) прописные и строчные буквы греческого алфавита;
 - в) прописные и строчные буквы румынского алфавита;
 - г) прописные и строчные буквы русского алфавита;
 - д) десятичные цифры 0, 1, 2, ..., 9;
 - е) цифры 0, 1, 2, ..., 9, знаки +, −, ×, / и скобки ();
 - жс) числовые показания в виде $hh:mm$ (hh — часы, mm — минуты) электронных часов;
 - з) числовые показания в виде $hh:mm:ss$ (ss — секунды) электронных часов;
 - и) числовые показания в виде $zz.ll.aa$ (zz — день, ll — месяц, aa — год) электронного календаря.
- ❻ Для каждого из источников, приведенных в упражнении 5, определите количество информации, содержащееся в 1000 сообщений, переданных источником.
- ❼ Напишите программу, вычисляющую количество информации в N сообщениях, переданных источником с n возможными сообщениями.

2.2. Кодирование и декодирование информации

Назовем **знаком** элемент конечного множества объектов, которые могут различаться. Линейно упорядоченное множество знаков называется **алфавитом**.

Представим некоторые из бесчисленного множества **алфавитов**:

- а) алфавит десятичных цифр: 0, 1, 2, ..., 9;
- б) алфавит прописных латинских букв: A, B, C, ..., Z;
- в) множество знаков зодиака;
- г) множество фаз Луны.

Особое значение представляют алфавиты, состоящие только из двух знаков. Такие алфавиты называются **двоичными алфавитами**, а их знаки соответственно — **двоичными знаками**.

Приведем несколько примеров двоичных алфавитов:

- а) цифры {0, 1};
- б) пара цветов {красный, желтый};
- в) пара состояний {закрит, открыт};
- г) пара ответов {да, нет};
- д) пара напряжений {0 В, 2 В};
- е) пара состояний {намагничен, не намагничен};
- ж) пара знаков {+, −} и т. д.

Удобно, чтобы знаки двоичного алфавита были представлены цифрами {0, 1}, которые называются **двоичными цифрами**. Последовательность, состоящая из m знаков, некоторые из которых могут повторяться, образует **слово**, а m представляет **длину слова**. Слова, образованные из двоичных знаков, называются **двоичными словами**. Очевидно, что слова могут иметь переменную или постоянную длину. В последнем случае они называются **m -позиционными словами**. Приведем некоторые множества слов постоянной длины:

- 1-позиционные: {0, 1};
- 2-позиционные: {00, 01, 10, 11};
- 3-позиционные: {000, 001, 010, 011, 100, 101, 110, 111};
- 4-позиционные: {0000, 0001, ..., 1110, 1111}.

Заметим, что $(m + 1)$ -позиционные слова образуются по два из m -позиционных слов добавлением двоичных цифр 0 и 1. Следовательно, множество m -позиционных слов включает 2^m различных слов.

Двоичные слова применяют для представления, передачи, хранения и обработки сообщений s_1, s_2, \dots, s_n источника информации (рис. 2.2).

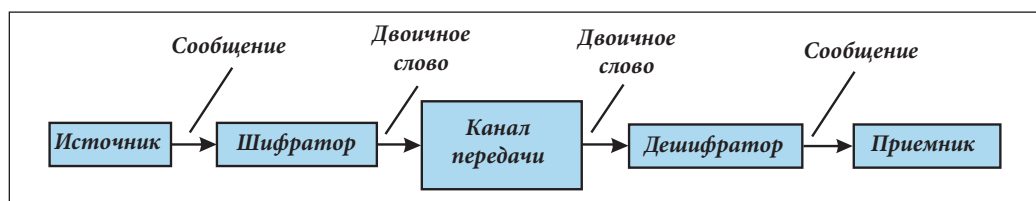


Рис. 2.2. Кодирование и декодирование сообщений в системах передачи информации

Правило преобразования сообщений в слова называется кодом, а соответствующая операция — кодированием. Операция, обратная кодированию, называется декодированием. Технические устройства, которые выполняют соответствующие операции, называются шифратор и дешифратор.

Самым простым является код, в котором возможным сообщениям s_1, s_2, \dots, s_n соответствуют двоичные слова постоянной длины m . Такой код, называемый **m -позиционным кодом**, может быть определен с помощью таблицы, содержащей возможные сообщения и соответствующие им слова. На рис. 2.3 представлены соответствующие таблицы для источника с $n = 2, 3, 4, \dots, 8$ возможными сообщениями.

$n=2, m=1$		$n=3, m=2$		$n=4, m=2$	
s_1	0	s_1	00	s_1	00
s_2	1	s_2	01	s_2	01
		s_3	10	s_3	10
				s_4	11

$n=5, m=3$		$n=6, m=3$		$n=7, m=3$		$n=8, m=3$	
s_1	000	s_1	000	s_1	000	s_1	000
s_2	001	s_2	001	s_2	001	s_2	001
s_3	010	s_3	010	s_3	010	s_3	010
s_4	011	s_4	011	s_4	011	s_4	011
s_5	100	s_5	100	s_5	100	s_5	100
		s_6	101	s_6	101	s_6	101
				s_7	110	s_7	110
						s_8	111

Рис. 2.3. Коды, состоящие из слов постоянной длины (m -позиционные коды)

Операции кодирования и декодирования состоят в извлечении необходимых данных из таблицы. Очевидно, что декодирование будет однозначным только тогда, когда двоичные слова, содержащиеся в таблице, различны. Это возможно, если длина m слов кода удовлетворяет неравенству

$$2^m \geq n.$$

После логарифмирования получаем:

$$m \geq \log_2 n.$$

Поскольку выражение $\log_2 n$ представляет количество информации, можно утверждать:

Длина слов любого позиционного кода должна быть больше или равна количеству информации в кодируемом сообщении.

Например, длина слов для кодирования прописных букв латинского алфавита $\{A, B, C, \dots, Z\}$, $n = 26$, определяется соотношением

$$m \geq \log_2 26 \approx 4,700.$$

Устанавливая $m = 5$, можем формировать двоичные 5-позиционные кодовые слова:

$$A - 00000$$

$B - 00001$
 $C - 00010$
 $D - 00011$
 $E - 00100$
 \dots
 $Z - 11001.$

Подобный код был предложен английским философом и государственным деятелем *Фрэнсисом Бэконом* еще в 1580 году.

Алгоритмы составления **кодов, состоящих из слов с переменной длиной**, значительно сложнее и изучаются в углубленных курсах информатики.

Вопросы и упражнения

- ❶ Что такое алфавит? Приведите примеры двоичных алфавитов.
- ❷ Как представляют знаки любого двоичного алфавита?
- ❸ Объясните, как могут быть получены двоичные $(m + 1)$ -позиционные слова. Чему равно число различных двоичных m -позиционных слов?
- ❹ Для чего предназначен код? Как определяется m -позиционный код?
- ❺ Как осуществляются кодирование и декодирование сообщений в случаях, когда код определен с помощью таблицы?
- ❻ Закодируйте сообщения s_3 , s_4 и s_6 источника с семью возможными сообщениями. Используйте 3-позиционный код с *рис. 2.3*.
- ❼ Декодируйте сообщения 100, 000 и 010, представленные в 3-позиционном коде на *рис. 2.3*, $n = 5$.
- ❽ Как определяется количество двоичных знаков, необходимых для формирования слов постоянной длины любого кода?
- ❾ Используя 3-позиционный код с *рис. 2.3*, $n = 6$, закодируйте сообщения s_1 , s_2 , s_6 , s_5 , s_3 , s_6 , s_3 , s_2 , s_1 .
- ❿ Как влияет количество информации некоторого сообщения на длину слова кода?
- ⓫ Объясните смысл терминов **количество информации** и **информация**.
- ⓬ Напишите программу, которая кодирует и декодирует буквы латинского алфавита. Используйте код, предложенный *Фрэнсисом Бэконом*.
- ⓭ Напишите программу, которая составляет таблицу m -позиционного кода для источника с n возможными сообщениями.

2.3. Часто используемые коды

Любой код, применяемый для представления, передачи, хранения и обработки информации, должен быть экономичным и нечувствительным к помехам, а соответствующие устройства кодирования и декодирования — простыми. По мере развития вычислительной техники разработано много кодов. Эти коды классифицируются на *числовые* и *алфавитно-числовые*.

Числовые коды обеспечивают возможность представления цифр $\{0, 1, 2, \dots, 9\}$ с помощью двоичных 4-позиционных слов. Примеры числовых кодов представлены в *таблице 2.2*.

Числовые коды

Цифра	Название кода			
	Прямой	Грея	Айкена	С избытком 3
0	0000	0000	0000	0011
1	0001	0001	0001	0100
2	0010	0011	0010	0101
3	0011	0010	0011	0110
4	0100	0110	0100	0111
5	0101	0111	1011	1000
6	0110	0101	1100	1001
7	0111	0100	1101	1010
8	1000	1100	1110	1011
9	1001	1101	1111	1100

Алфавитно-числовые коды представляют с помощью двоичных слов цифры 0, 1, 2, ..., 9, строчных и прописных букв алфавита, знаков препинания, знаков арифметических операций и т. д. В таблице 2.3 представлен код **ASCII** (*American Standard Code for Information Interchange* — Американский стандартный код для информационного обмена), изобретенный в 1968 году.

Код ASCII

Символ	Двоичное слово	Десятичный эквивалент	Символ	Двоичное слово	Десятичный эквивалент
Пробел	0100000	32	P	1010000	80
!	0100001	33	Q	1010001	81
"	0100010	34	R	1010010	82
#	0100011	35	S	1010011	83
\$	0100100	36	T	1010100	84
%	0100101	37	U	1010101	85
&	0100110	38	V	1010110	86
'	0100111	39	W	1010111	87
(0101000	40	X	1011000	88
)	0101001	41	Y	1011001	89
*	0101010	42	Z	1011010	90
+	0101011	43	[1011011	91
,	0101100	44	\	1011100	92
-	0101101	45]	1011101	93
.	0101110	46	^	1011110	94

Символ	Двоичное слово	Десятичный эквивалент	Символ	Двоичное слово	Десятичный эквивалент
/	0101111	47	—	1011111	95
0	0110000	48	`	1100000	96
1	0110001	49	a	1100001	97
2	0110010	50	b	1100010	98
3	0110011	51	c	1100011	99
4	0110100	52	d	1100100	100
5	0110101	53	e	1100101	101
6	0110110	54	f	1100110	102
7	0110111	55	g	1100111	103
8	0111000	56	h	1101000	104
9	0111001	57	i	1101001	105
:	0111010	58	j	1101010	106
;	0111011	59	k	1101011	107
<	0111100	60	l	1101100	108
=	0111101	61	m	1101101	109
>	0111110	62	n	1101110	110
?	0111111	63	o	1101111	111
@	1000000	64	p	1110000	112
A	1000001	65	q	1110001	113
B	1000010	66	r	1110010	114
C	1000011	67	s	1110011	115
D	1000100	68	t	1110100	116
E	1000101	69	u	1110101	117
F	1000110	70	v	1110110	118
G	1000111	71	w	1110111	119
H	1001000	72	x	1111000	120
I	1001001	73	y	1111001	121
J	1001010	74	z	1111010	122
K	1001011	75	{	1111011	123
L	1001100	76		1111100	124
M	1001101	77	}	1111101	125
N	1001110	78	~	1111110	126
O	1001111	79	Del	1111111	127

Данный код является 7-позиционным и включает $2^7 = 128$ символов. Первые 32 символа (двоичные слова 0000000, 0000001, 0000010, ..., 0011111) определяют технические детали передачи информации и не включены в таб-

лицу. Двоичные слова 0100000, 0100001, 0100010, ..., 1111110 представляют печатные символы текстов на английском языке. Слово 1111111 представляет непечатаемый символ *Delete* (Стирание).

Кодирование сообщений осуществляется путем замены символов соответствующими двоичными словами. Например, слово START представляется в коде ASCII следующей последовательностью двоичных слов:

1010011 1010100 1000001 1010010 1010100.

Очевидно, что декодирование осуществляется в обратном порядке. Например, последовательность двоичных слов

1010011 1010100 1001111 1010000

представляет в коде ASCII слово STOP.

Как правило, языки программирования оперируют не с самими двоичными словами, а с их десятичными эквивалентами. В программах на языке ПАСКАЛЬ десятичные эквиваленты символов могут быть найдены с помощью предопределенной (встроенной) функции ord. Например:

ord('S')=83; ord('T')=84; ord('A')=65; ord('R')=82

и т. д. Предопределенная функция chr возвращает символ, соответствующий указанному десятичному эквиваленту. Так,

chr(83)='S'; chr(84)='T'; chr(65)='A'; chr(82)='R'.

Подобным образом в программах на языке C++ записываем:

int('S')=83; int('T')=84; int('A')=65; int('R')=82
char(83)='S'; char(84)='T'; char(65)='A'; char(82)='R'.

Ориентированный на английские тексты код ASCII не включает буквы с диакритическими знаками и специальные графические символы, встречаемые в разных европейских языках и в научных работах. Поэтому для современных компьютеров разработаны специальные версии кода ASCII, называемые **расширенными кодами ASCII**. Расширенные коды являются 8-позиционными и включают $2^8 = 256$ символов. Структура соответствующих кодов представлена в *таблице 2.4*.

Таблица 2.4

Структура расширенных ASCII кодов

Символ	Двоичное слово	Десятичный эквивалент	Замечания
Пробел	00100000	32	Часть 1: - символы кода ASCII
!	00100001	33	
"	00100010	34	
#	00100011	35	
...	
}	01111101	125	
~	01111110	126	
Del	01111111	127	

Символ	Двоичное слово	Десятичный эквивалент	Замечания
A	10000000	128	Часть 2: - специфические символы национальных языков; - псевдографические символы; - научные символы
B	10000001	129	
B	10000010	130	
...	
≡	11110000	240	
Ă	11110001	241	
ă	11110010	242	
Â	11110011	243	
â	11110100	244	
Î	11110101	245	
î	11110110	246	
Ș	11110111	247	
ș	11111000	248	
'	11111001	249	
-	11111010	250	
√	11111011	251	
Ț	11111100	252	
ț	11111101	253	
□	11111110	254	
	11111111	255	

Часть 1 каждого расширенного кода включает символы от 0 до 127, содержащиеся в обычном коде ASCII. **Часть 2** определена для каждой страны в отдельности и включает символы от 128 до 255. Эти символы применяют как для представления букв национальных алфавитов, так и для часто используемых научных символов. Для примера в *таблице 2.4* представлены коды букв *Ă, ă, Â, â, Î, î, Ș, ș, Ț, ț* из алфавита румынского языка и коды букв *A, B, B, ...* из алфавита русского языка, предложенные в 1992 году фирмой *TISH* (Кишинэу). Очевидно, что использование расширенных кодов обеспечивает обработку информации, представленной на различных языках.

Другим примером алфавитно-числового кода является двоичный 8-позиционный код **EBCDIC** (*Extended Binary Coded Data Interchange Code* — Расширенный двоичный код для обмена данными), который применяют для больших электронно-вычислительных машин.

Необходимо подчеркнуть, что расширение области применения 8-позиционных кодов способствует использованию байта и производных от него единиц для измерения количества информации:

$$1 \text{ байт} = 2^3 = 8 \text{ бит};$$

$$1 \text{ Килобайт} = 2^{10} \approx 10^3 \text{ байт};$$

$$1 \text{ Мегабайт} = 2^{20} \approx 10^6 \text{ байт};$$

$$1 \text{ Гигабайт} = 2^{30} \approx 10^9 \text{ байт};$$

$$1 \text{ Терабайт} = 2^{40} \approx 10^{12} \text{ байт};$$

$$1 \text{ Петабайт} = 2^{50} \approx 10^{15} \text{ байт}.$$

В специальной литературе байт обозначается *B (byte)*, а соответствующие производные единицы – *KB, MB, GB, TB* и *PB*.

Известно, что информатика в значительной степени подвержена явлению глобализации, поэтому программные продукты и цифровое оборудование разрабатывают таким образом, что они могут обрабатывать информацию, представленную на разных языках. Для компьютерного представления символов практически всех языков мира чаще всего используется код UNICODE, длина слов которого может достигать 32 двоичных цифр.

Вопросы и упражнения

- ❶ Сколько возможных сообщений может быть закодировано с помощью *m*-позиционного кода?
- ❷ Определите коды, которые применены в операционной системе, с которой вы работаете.
- ❸ Закодируйте в коде Грея следующие последовательности десятичных цифр: 123, 461, 952, 783, 472.
- ❹ Декодируйте сообщения, представленные в коде Айкена:
 - a) 0011 1111 0100 d) 1110 0010 1101
 - b) 1111 0000 0100 e) 0011 1100 1111
 - c) 0010 0001 1011 f) 1111 1101 0000
- ❺ Закодируйте в ASCII-коде выражения:
 - a) A+B d) NEXT I
 - b) FOR I=1 TO N e) PAUSE
 - c) PRINT A\$ f) PROGRAM
- ❻ Декодируйте сообщения, представленные в ASCII-коде:
 - a) 1000010 1100101 1100111 1101001 1101110;
 - b) 1010011 1110100 1101111 1110000;
 - c) 1000101 1101110 1100100;
 - d) 1101001 0111010 0111101 0110001 0111011.
- ❼ Разработайте программу, которая выводит на экран коды следующих символов, введенных с клавиатуры:
 - a) десятичные цифры 0, 1, 2, ..., 9;
 - b) прописные латинские буквы A, B, C, ..., Z;
 - c) строчные латинские буквы a, b, c, ..., z;
 - d) знаки арифметических операций;
 - e) специальные символы ;, <, =, >, ?, [,], {, }, /, \.
- ❽ ТЕМАТИЧЕСКОЕ ИССЛЕДОВАНИЕ. С помощью поисковой системы найдите в Интернете описания часто используемых кодов. Установите исторический курс соответствующих кодов, их достоинства и недостатки, сферы их применения.

2.4. Информация непрерывных сообщений

Источники информации, изученные ранее, определялись с помощью переменной S , которая может принимать значения из конечного множества различных элементов $\{s_{n1}, s_{n2}, \dots, s_n\}$, называемого **множеством возможных сообщений**. Практика показывает, что не все источники информации могут быть определены подобным образом. Для примера приведем ртутные либо спиртовые термометры, спидометры автомобилей, микрофоны, видеокамеры и др. Такие источники могут быть определены с помощью переменной S (температура, мгновенная скорость, напряжение на выходных клеммах микрофона и т. д.), которая может принимать любые значения из заданного интервала $[s_{\min}, s_{\max}]$.

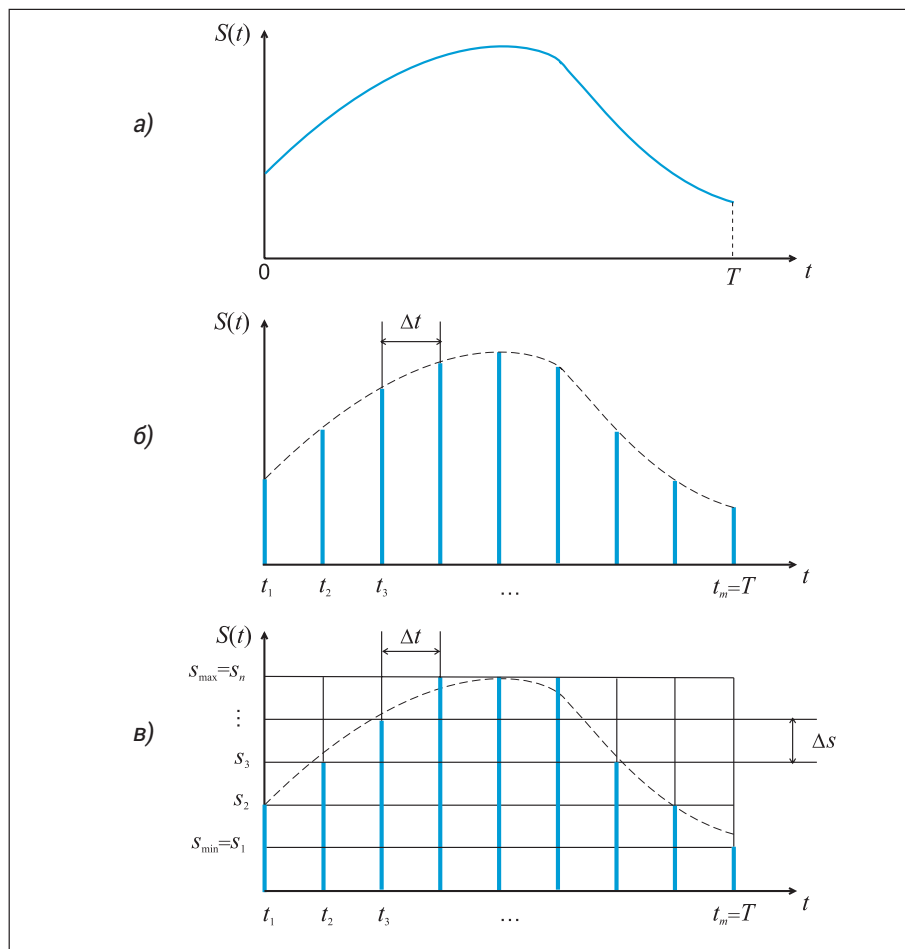


Рис. 2.4. Дискретизация непрерывных сообщений: а — непрерывное сообщение; б — сообщение, дискретизированное во времени; в — сообщение, дискретизированное во времени и по уровню

Источники информации, которые определены с помощью переменной S , принимающей значения из конечного множества различных элементов, на-

зываются источниками с **дискретными сообщениями**. Источники, определяемые с помощью переменной, которая может принимать любые значения из заданного интервала, называются **источниками с непрерывными (аналоговыми) сообщениями**.

Как и дискретные, непрерывные сообщения осуществляются во времени. Следовательно, S — функция времени, $S = S(t)$. С целью оценки количества/объема информации в непрерывных сообщениях будем рассматривать значения функции $S(t)$ только в моменты времени t_1, t_2, \dots, t_m (рис. 2.4). Множество соответствующих значений (отсчетов), обозначаемое $\{S(t_1), S(t_2), \dots, S(t_m)\}$, называется **выборкой**.

Как правило, моменты времени t_1, t_2, \dots, t_m определяются в соответствии с соотношением:

$$t_i = t_{i-1} + \Delta t.$$

Величина Δt называется **интервалом дискретизации**. Конкретное значение интервала дискретизации Δt выбирается исходя из скорости, с которой $S(t)$ изменяется во времени.

Например, в метеорологии изменения температуры происходят на протяжении часов, и $\Delta t = 1$ час, а в технике обработки звуковых сигналов $\Delta t = 5 \cdot 10^{-5}$ с.

Операция преобразования непрерывных сообщений в выборку называется дискретизацией во времени.

Очевидно, что до приема непрерывного сообщения конкретные отсчеты $S(t_1), S(t_2), \dots, S(t_m)$ не известны получателю. Известен только интервал $[s_{\min}, s_{\max}]$, к которому принадлежат рассматриваемые значения. Для оценки количества информации в каждом из отсчетов выборки округлим значения $S(t_i)$, $i = 1, 2, \dots, m$ до одного из заранее заданных значений s_1, s_2, \dots, s_n (рис. 2.4).

Заранее выбранные значения s_1, s_2, \dots, s_n называются **квантами**, а операция преобразования текущих значений непрерывных сообщений в кванты — **дискретизацией по уровню, или квантованием**.

Обычно

$$s_1 = s_{\min}, \quad s_2 = s_{\min} + \Delta s, \quad \dots, \quad s_i = s_{i-1} + \Delta s, \quad \dots, \quad s_n = s_{\max}.$$

Величина Δs представляет шаг или **интервал квантования**. Конкретное значение интервала квантования зависит от физической природы источника информации, точности измерений, разрешающей способности приемника и т.д.

Например, для медицинского термометра $s_{\min} = 34^\circ$, $s_{\max} = 42^\circ$ и $\Delta s = 0,1^\circ$. В метеорологии $s_{\min} = -60^\circ$, $s_{\max} = +60^\circ$, $\Delta s = 1^\circ$. Для спидометра автомобиля $s_{\min} = 0$, $s_{\max} = 150$ км/ч, $\Delta s = 5$ км/ч.

Количество отсчетов выборки m и количество квантов n определяются следующими соотношениями (рис. 2.4 б и с):

$$m = \frac{T}{\Delta t} + 1; \quad n = \frac{|s_{\max} - s_{\min}|}{\Delta s} + 1,$$

где T — длительность непрерывного сообщения.

Поскольку кванты s_1, s_2, \dots, s_n могут рассматриваться как дискретные сообщения, количество информации в одном отсчете:

$$I = \log_2 n = \log_2 \left(\frac{|s_{\max} - s_{\min}|}{\Delta s} + 1 \right),$$

а количество информации в непрерывном сообщении:

$$V = mI = \left(\frac{T}{\Delta t} + 1 \right) \log_2 \left(\frac{|s_{\max} - s_{\min}|}{\Delta s} + 1 \right).$$

Например, для медицинского термометра:

$$I = \log_2 \left(\frac{|42 - 34|}{0,1} + 1 \right) \approx 6,34 \text{ бит}$$

Для спидометра автомобиля:

$$I = \log_2 \left(\frac{|150 - 0|}{5} + 1 \right) \approx 4,95 \text{ бит}$$

Количество информации в аудиозаписи для $n = 256$, $\Delta t = 5 \cdot 10^{-5}$ и $T = 45$ мин составляет:

$$V = \frac{45 \cdot 60}{5 \cdot 10^{-5}} \log_2 256 = 4,32 \cdot 10^8 \text{ бит} = 432 \text{ Мбит}$$

Если точность измерения и разрешающая способность приемника возрастают, то интервал дискретизации Δt и шаг квантования Δs должны быть уменьшены. Следовательно, возрастает и количество информации, содержащейся в непрерывном сообщении.

Информация непрерывных сообщений может быть представлена с помощью последовательности двоичных слов. Для этого кванты s_1, s_2, \dots, s_n кодируются точно так же, как и любое дискретное сообщение. Например, показания медицинского термометра ($I \approx 6,34 \text{ бит}$) могут быть закодированы с помощью 7-позиционного кода. Чаще всего используют прямые числовые коды (таблица 2.2), причем кодовое слово представляет количество соответствующих квантов. В некоторых приложениях применён код Грея, который нечувствителен к помехам.

Устройство, которое преобразовывает непрерывное сообщение, поданное на его вход, в последовательность кодовых слов, называется **аналого-цифровым преобразователем (АЦП)**. Обратная операция, состоящая в преобразовании кодовых слов в непрерывное сообщение, осуществляется с помощью **цифро-аналогового преобразователя (ЦАП)**. Использование преобразователей необходимо в случаях, когда обрабатываемая информация представлена непрерывными сообщениями: при контроле технологических процессов, в управлении движущимися объектами, в мониторинге физиологических параметров в медицине, для фильтрации и микширования аудиосигналов и т.д.

Вопросы и упражнения

- 1 В чем отличие источников с дискретными и с непрерывными сообщениями?
- 2 Приведите несколько примеров источников непрерывных сообщений. Уточните интервал изменения переменной, описывающей источник.
- 3 Объясните, как осуществляется операция дискретизации во времени. Как выбирается интервал дискретизации?

- ④ Объясните, как осуществляется операция квантования. Как выбирается шаг квантования?
- ⑤ Как влияют интервал дискретизации и шаг квантования на количество информации, извлекаемой из непрерывного сообщения?
- ⑥ Импульсный альтиметр (высотомер) самолета может измерять высоту от 100 м до 20 км. Погрешность измерения не превышает 1 м. Чтобы выполнить одно измерение, нужно 10^{-3} с. Определите количество информации, выдаваемой альтиметром за 5 часов полета.
- ⑦ Температура внутри химического реактора записывается на бумажной миллиметровой ленте. По оси абсцисс отображается время (1 мм соответствует 1 часу), а по оси ординат — температура (1 мм соответствует 10°C). Сколько информации содержит запись, осуществленная за 30 дней, если температура может изменяться от 80 до 1000°C ?
- ⑧ Для записи звука используется микрофон, напряжение на выходе которого изменяется от 0 до 100 мкВ. Устройство записи не различает уровни напряжений, отличающиеся менее чем на 0,1 мкВ. Для обеспечения качественного воспроизведения каждую секунду берутся 40000 отсчетов. Сколько информации вырабатывает микрофон на протяжении 3 часов?
- ⑨ Для чего предназначены аналого-цифровые и цифро-аналоговые преобразователи?
- ⑩ Напишите программу, которая вводит с клавиатуры текущие значения отсчетов и выводит на экран коды соответствующих квантов.
- ⑪ Напишите программу, которая моделирует работу цифро-аналогового преобразователя.
- ⑫ ИССЛЕДУЙТЕ! Пользуясь поисковой системой, найдите в Интернете описание цифровых звукозаписывающих устройств. Узнайте период дискретизации и шаг квантования, используемые для оцифровки звуковых сообщений, выполняемых каждым из этих устройств.
- ⑬ ТЕМАТИЧЕСКОЕ ИССЛЕДОВАНИЕ. С помощью поисковой системы найдите в Интернете описания цифровых звукозаписывающих устройств, предлагаемых коммерческими/торговыми агентами. Сравните эти устройства с точки зрения способа оцифровки аудиосообщений. Узнайте, как параметры дискретизации влияют на стоимость звукозаписывающих устройств, каковы их преимущества, недостатки и области применения.

2.5. Квантование изображений

Изображением называется представление некоторого объекта, выполненное на поверхности самим пользователем напрямую или с помощью определенных устройств. Для примера вспомним рисунки, фотографии, изображения, полученные с помощью различных систем — оптических, оптико-механических или оптико-электронных: микроскопа, телескопа, кинопроектора, телевизора и др.

Для измерения количества информации изображение делится на **микроразоны**, называемые чаще всего **точками**, или **пикселями**. Разложение изображения на точки осуществляется с помощью **растра** (от латинского слова *raster* — «грабли»). Растр представляет плоскую поверхность, чаще прямоугольную, на которую нанесены два ряда параллельных линий, перпендикулярных между собой (рис. 2.5). Плотность линий и соответственно плотность точек

характеризуют разрешающую способность оборудования для воспроизведения или формирования изображений.

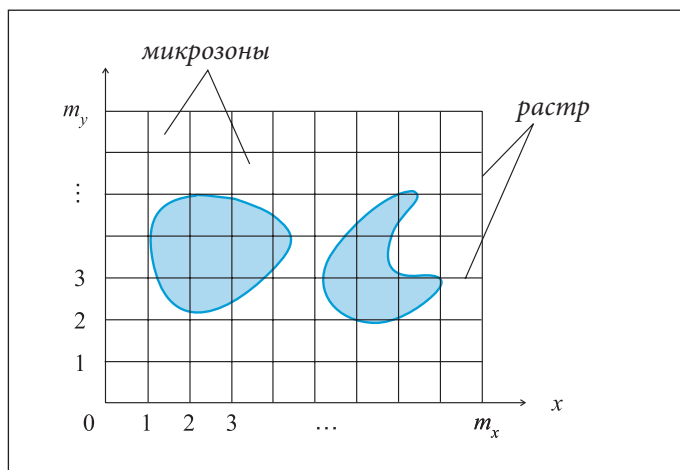


Рис. 2.5. Разложение изображения на микрозоны

Например, для газетных иллюстраций используют растры с разрешением 24–30 *линий/см* (576–900 точек на 1 см^2), а для воспроизведения картин — растры с 54–60 *линий/см*. Растр монитора, то есть рисунок, который формирует электронный луч на экране электронно-лучевой трубки (кинескопа), может включать 640×480 , 800×600 , 720×400 , ..., 1024×1024 точек.

Разложение изображения на точки (микрозоны) представляет собой операцию дискретизации в пространстве.

В случае монохромных (черно-белых) изображений каждая микрозона описывается ее яркостью, которая в общем случае представляет собой непрерывную величину. Эта величина может быть дискретизована (квантована) по уровню. Количество квантов n будет характеризовать разрешающую способность оборудования для воспроизведения или формирования изображений. Следовательно, количество информации в одном монохромном изображении:

$$I = m_x m_y \log_2 n,$$

где m_x и m_y представляют количество микрозон соответствующего растра по горизонтали и вертикали (рис. 2.5).

Поскольку цвета могут быть переданы путем совмещения трех представлений одного и того же изображения в красном, зеленом и синем, то количество информации в цветном изображении определяется соотношением:

$$I = 3 m_x m_y \log_2 n.$$

Изображения движущихся объектов дискретизируются во времени, представляя обычно 24 (в кинематографии) или 25 (в телевидении) кадров в секунду. Следовательно, количество информации в одном фильме продолжительностью T секунд определяется соотношением:

$$V = T f I,$$

где f — это частота кадров, а I — количество информации в одном кадре.

Например, в телевидении $m_x \approx m_y = 625$, $n = 32$ и $f = 25$ кадров в секунду. Один кадр будет содержать:

$$I = 3 \cdot 625 \cdot 625 \cdot \log_2 32 \approx 5,6 \text{ Мбит}.$$

Цветной фильм продолжительностью 1,5 часа будет содержать:

$$V = 1,5 \cdot 3\,600 \cdot 25 \cdot I \approx 791 \text{ Гбит}.$$

Набор двоичных слов, содержащих информацию о микрizonaх, называется цифровым изображением. Операция преобразования изображения в набор двоичных слов называется квантованием (оцифровыванием) изображения.

Изображения, полученные с видеокамер, квантуются с помощью аналого-цифровых преобразователей. Изображения на бумаге могут быть квантованы с помощью специального устройства — сканера. Это устройство содержит фоточувствительные ячейки, аналого-цифровые преобразователи и механизм относительного перемещения бумаги и фотоячеек.

Цифровые изображения преобразуются в видимые изображения с помощью цифро-аналоговых преобразователей и устройства формирования раstra: электронно-лучевой трубки и системы отклонения монитора, матрицы иглонок в механических принтерах и т. д.

Вопросы и упражнения

- 1 Назовите операции, необходимые для квантования изображений.
- 2 Каково назначение раstra? Чем руководствуются при выборе плотности линий раstra?
- 3 Как оценить количество информации, содержащейся в монохромном изображении?
- 4 Как можно передать цвета многоцветного изображения? Как измерить количество информации в цветном изображении?
- 5 Оцените количество информации в газетной фотографии размерами 10×10 см, переданной с помощью раstra, содержащего 24 точек/см. Каждая точка может содержать оттенки: белый, светло-серый, темно-серый, черный.
- 6 Сколько информации содержит цветная фотография размерами 20×20 см, воспроизведенная с помощью раstra, содержащего 60 точек/см. Может быть передано до 256 уровней яркости соответствующих точек.
- 7 Растр видеокамеры состоит из 1024×1024 точек. Может быть передано до 64 уровней яркости соответствующих точек. Сколько информации будет содержаться в фильме продолжительностью 3 часа?
- 8 Цифровое изображение содержит по одному двоичному слову для каждой точки раstra монитора. Сколько уровней яркости можно отобразить на экране, если слова цифрового изображения являются 3-позиционными? 5-позиционными? 8-позиционными?
- 9 ИССЛЕДУЙТЕ! Пользуясь поисковиками, найдите в Интернете описание цифровых фотоаппаратов. Узнайте мощность разрешения каждого из устройств при оцифровке статических изображений.
- 10 ТЕМАТИЧЕСКОЕ ИССЛЕДОВАНИЕ. С помощью поисковой системы найдите в Интернете описания цифровых фотоаппаратов, предлагаемых коммерческими агентами. Сравните эти устройства с точки зрения способа

оцифровки статических изображений. Узнайте, как параметры дискретизации влияют на стоимость цифровых фотоаппаратов, каковы их преимущества, недостатки и области применения.

❶ **ИССЛЕДУЙТЕ!** Пользуясь поисковиками, найдите в Интернете описания цифровых видеокамер. Узнайте мощность разрешения каждого из устройств при оцифровке динамической информации.

❷ **ТЕМАТИЧЕСКОЕ ИССЛЕДОВАНИЕ.** С помощью поисковой системы найдите в Интернете описания цифровых видеокамер, предлагаемых коммерческими агентами. Сравните эти камеры с точки зрения способа оцифровки динамических изображений. Узнайте, как параметры дискретизации влияют на стоимость цифровых видеокамер, каковы их преимущества, недостатки и области применения.

2.6. Представление и передача информации

Материальный объект, используемый для хранения, передачи или обработки информации, называется **носителем информации**. Различают статические и динамические носители информации.

Статические носители используются для хранения информации или, другими словами, для передачи ее во времени. Информация, записанная на статический носитель, может быть считана с целью последующей обработки или использования. Первыми носителями информации, использованными людьми, были камни, таблички из обожженной глины, папирус. Другим носителем информации является бумага. Сообщения, записанные на бумаге в рукописной форме, в виде рисунков или печатных текстов, могут сохраняться очень продолжительное время. В компьютерах в качестве статических носителей информации используются:

- бумага для механических, струйных, лазерных и других принтеров;
- активные слои магнитных лент и дисков;
- отражающие среды оптических дисков и т. п.

Передача информации в пространстве осуществляется с помощью **динамических носителей**. В качестве динамических носителей современная техника использует:

- акустические волны в газах (воздухе) или жидкостях;
- электрические напряжение и ток;
- электромагнитные волны и т. п.

Поскольку передача информации производится в пространстве и времени, как минимум хотя бы одна физическая величина носителя информации должна изменяться.

Изменение физической величины, обеспечивающее передачу сообщений, называется сигналом. Характеристика сигнала, используемая для представления (описания) сообщений, называется информационным параметром.

Например, в радио- и телевидении в качестве носителя информации используются электромагнитные волны. Амплитуда или частота этих волн может изменяться во времени (рис. 2.6). В первом случае информационным параметром является амплитуда колебаний, а во втором — их частота.

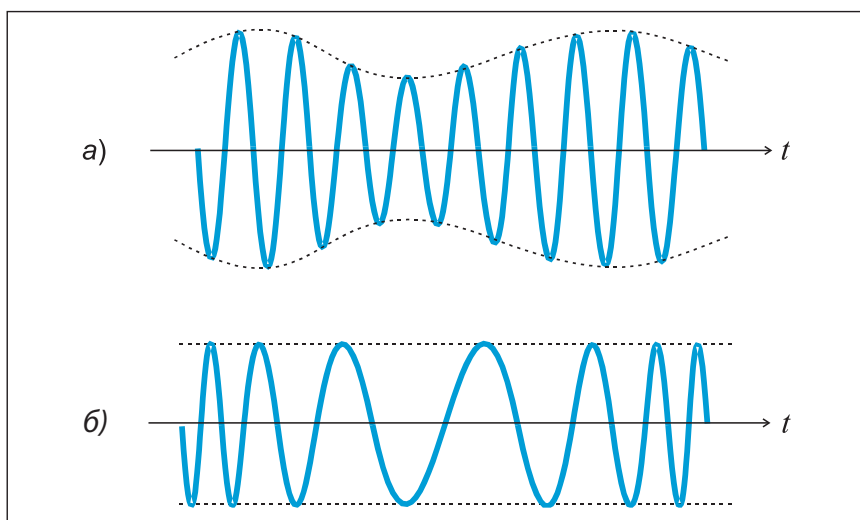


Рис. 2.6. Информационные параметры электромагнитных волн:
а — амплитуда; б — частота

В электронных компьютерах в качестве носителя информации обычно используется электрический ток, а напряжение и сила тока являются информационными параметрами сигнала. Форма рассматриваемых сигналов представлена на рис. 2.7. В случае уровней напряжения с одним значением напряжения связывается двоичная цифра 0, а с другим — двоичная цифра 1. Двоичные цифры 0 и 1 могут также быть связаны соответственно с отсутствием или наличием импульса, с отрицательным или положительным импульсом и др.

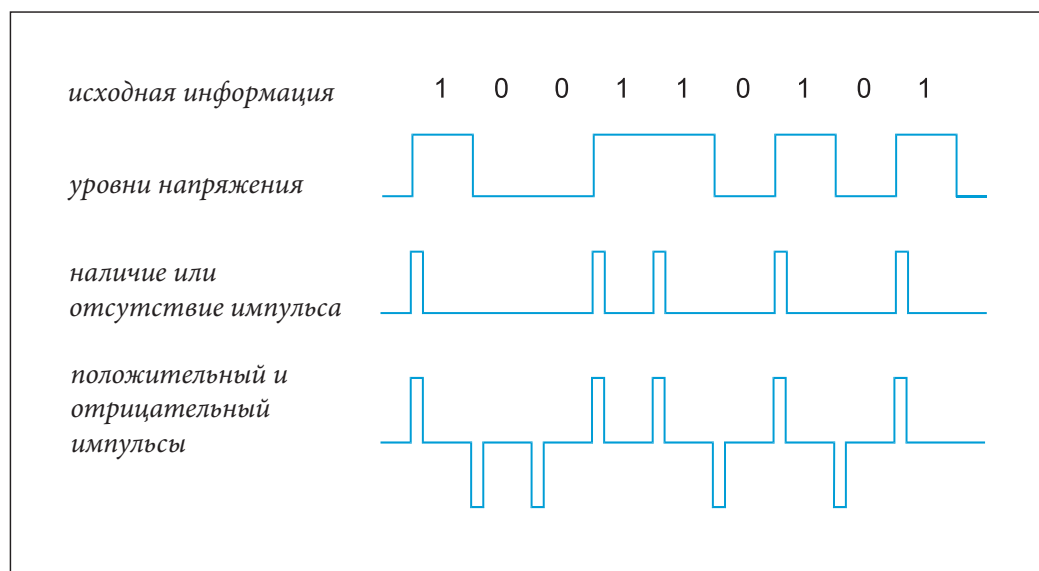


Рис. 2.7. Сигналы, используемые в вычислительной технике

Сигнал называется дискретным, если соответствующий информационный параметр может принимать конечное число значений. Сигнал называется непрерывным, если информационный параметр может принимать произвольные значения из заданного интервала.

Например, сигналы на рис. 2.6 — непрерывные, а сигналы на рис. 2.7 — дискретные. Очевидно, что дискретные и непрерывные сигналы являются формой представления соответствующих сообщений.

Любая техническая система использует те сигналы, которые обеспечивают наилучшую реализацию функций, для которых она была спроектирована. Современные компьютеры используют уровни напряжения, телефонные сети — электрический ток, а радио и телевидение — электромагнитные волны и т. д. Практика показывает, что непрерывные сигналы могут быть переданы на значительно большие расстояния, чем дискретные сигналы. Следовательно, для обеспечения связи между компьютерами, находящимися на расстоянии, при передаче дискретных сигналов они должны быть преобразованы в непрерывные. При приеме осуществляется обратное преобразование: непрерывный сигнал преобразуется в дискретный.

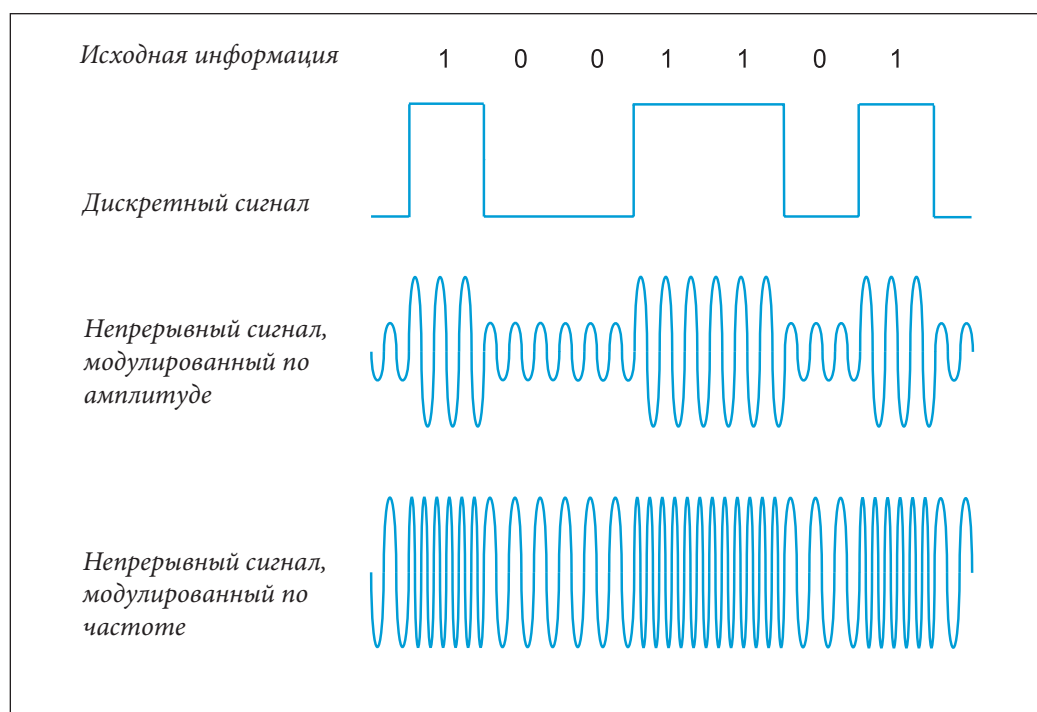


Рис. 2.8. Модуляция непрерывных сигналов

Операция, с помощью которой информационный параметр непрерывного сигнала изменяется в зависимости от значений дискретного сигнала, называется модуляцией.

Техническое устройство, которое реализует указанную операцию, называется **модулятором**. Для примера на рис. 2.8 представлены операции модуляции по амплитуде и частоте.

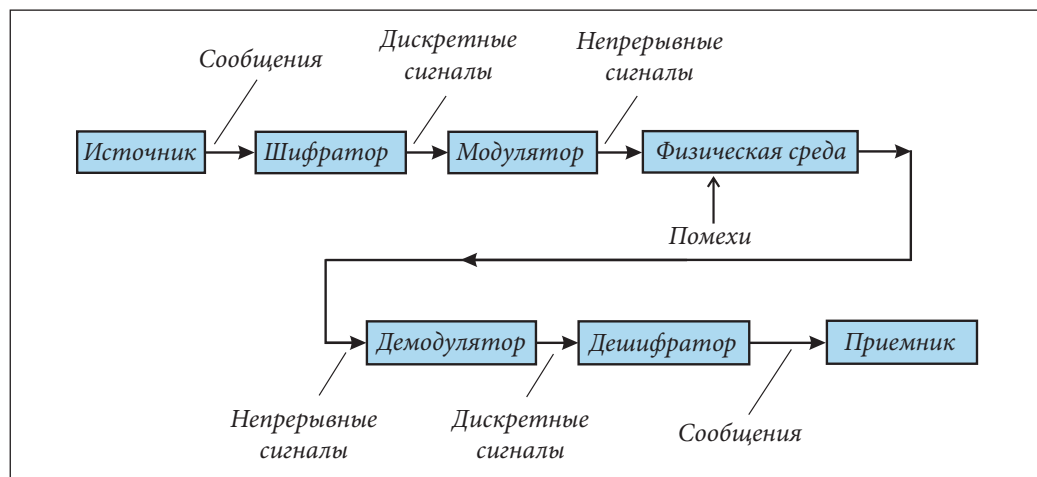


Рис. 2.9. Подробная схема системы передачи информации

Операция извлечения дискретного сигнала из непрерывного в соответствии с применяемым способом модуляции называется демодуляцией.

Техническое устройство, которое реализует соответствующую операцию, называется **демодулятором**.

Вспомним назначение компонент рассматриваемой системы:

шифратор — преобразует сообщения, вырабатываемые источником в двоичные слова;

модулятор — преобразует дискретные сигналы, представляющие собой двоичные слова, в непрерывные сигналы;

физическая среда — представляет собой проводники, оптические волокна, эфир и т.п., через которые распространяются непрерывные сигналы;

демодулятор — преобразует непрерывные сигналы в дискретные;

дешифратор — преобразует двоичные слова в сообщения.

Очевидно, что в случае использования кодов для коррекции и обнаружения ошибок система передачи информации будет помехоустойчивой.

Главной характеристикой любой системы передачи информации является ее **пропускная способность**, которая выражается в **битах в секунду**. Пропускная способность зависит от физических характеристик компонент системы, методов модуляции-демодуляции, статистических характеристик помех. Например, пропускная способность телефонного канала составляет около 34 Кбит/с ; пропускная способность радиоканала с сантиметровыми волнами — около 1 Гбит/с ; пропускная способность оптического канала — 1 Тбит/с .

Вопросы и упражнения

- ❶** В чем отличия статических и динамических носителей информации?
- ❷** Определите тип следующих носителей информации:
а) перфокарты; д) фото пленка;
б) ультразвуковые волны; е) гравитационные волны;
в) перфоленты; ж) фотобумага.
з) акустические волны;
- ❸** Назовите информационные параметры сигналов от следующих источников:
а) микрофон;
б) радиостанция, волны — длинные, средние или короткие;
в) музыкальный инструмент;
г) радиостанция, ультракороткие волны;
д) видеокамера.
- ❹** Опишите носители информации и сигналы, используемые в современных компьютерах.
- ❺** Объясните операции модуляции и демодуляции сигналов.
- ❻** Каково назначение модулятора? Демодулятора?
- ❼** На рис. 2.8 приведены непрерывные сигналы, представляющие двоичное слово 1001101. В коде ASCII (таблица 2.3) это слово соответствует символу М. Нарисуйте непрерывные сигналы, соответствующие следующим символам:

а)	>;		д)	<;	
б)	т;		е)	к;	
в)	ш;		ж)	а;	
з)	9;		з)	@.	
- ❽** От чего зависит пропускная способность канала? В каких единицах она измеряется?
- ❾** Как влияют помехи на пропускную способность канала?
- ❿ ИССЛЕДУЙТЕ!** Узнайте пропускную способность каналов информационных систем, к которым у вас есть доступ дома, в школе, в общественных местах.

Глава 3

АРИФМЕТИЧЕСКИЕ ОСНОВЫ ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

3.1. Системы счисления

В цифровых компьютерах информация любого вида представляется, хранится и обрабатывается в числовой форме. Числа представляются с помощью элементарных символов, называемых **цифрами**.

Совокупность правил представления чисел вместе с множеством используемых цифр носит название системы счисления. Количество цифр определяет основание системы счисления.

Приведем несколько примеров систем счисления:

- **десятичная система** — это система счисления по основанию 10 с количеством используемых цифр, равным 10, соответственно 0, 1, 2, ..., 9;
- **двоичная система** — это система счисления по основанию 2 с количеством используемых цифр, равным 2, то есть 0 и 1. Соответствующие цифры называются **двоичными цифрами**, или **битами**;
- **троичная система** — это система счисления по основанию 3 с количеством используемых цифр, равным 3, соответственно 0, 1 и 2;
- **восьмеричная система** — это система счисления по основанию 8, содержащая 8 цифр: 0, 1, 2, ..., 7;
- **шестнадцатеричная система** — это система счисления по основанию 16, которая содержит 16 цифр: 0, 1, 2, ..., 9, *A* (десять), *B* (одиннадцать), *C* (двенадцать), *D* (тринадцать), *E* (четырнадцать), *F* (пятнадцать).

В *таблице 3.1* представлены одни и те же числа в различных системах счисления.

Таблица 3.1

Представление чисел в различных системах счисления

Десятич- ная	Двоич- ная	Восьме- ричная	Шестнад- цатеричная	Десятич- ная	Двоич- ная	Восьме- ричная	Шестнад- цатеричная
0	0	0	0	7	111	7	7
1	1	1	1	8	1000	10	8
2	10	2	2	9	1001	11	9
3	11	3	3	10	1010	12	A
4	100	4	4	11	1011	13	B
5	101	5	5	12	1100	14	C
6	110	6	6	13	1101	15	D

Десятич- ная	Двоич- ная	Восьме- ричная	Шестнад- цатеричная	Деся- тичная	Двоич- ная	Восьме- ричная	Шестнад- цатеричная
14	1110	16	E	50	110010	62	32
15	1111	17	F
16	10000	20	10	60	111100	74	3C
17	10001	21	11
18	10010	22	12	70	1000110	106	46
19	10011	23	13
20	10100	24	14	80	1010000	120	50
...
30	11110	36	1E	90	1011010	132	5A
...
40	101000	50	28	100	1100100	144	64
...

Правило представления чисел в десятичной системе видно из следующего примера:

$$(3856,43)_{10} = 3 \cdot 10^3 + 8 \cdot 10^2 + 5 \cdot 10^1 + 6 \cdot 10^0 + 4 \cdot 10^{-1} + 3 \cdot 10^{-2}.$$

Заметим, что в этом представлении значение (вес) каждой цифры зависит от положения, которое она занимает в числе. Например, цифра 3 встречается два раза: первый раз она имеет значение 3000, а второй раз — 0,03.

Системы, в которых значение цифр зависит от занимаемого ими положения в представлении (записи) чисел, называются позиционными.

Предположим, что число N имеет целую часть, состоящую из $n + 1$ цифр, а дробную часть — из m цифр:

$$N = c_n c_{n-1} \dots c_1 c_0, c_{-1} c_{-2} \dots c_{-m}.$$

Значение этого числа вычисляется в зависимости от основания системы следующим образом:

$$(N)_b = c_n b^n + c_{n-1} b^{n-1} + \dots + c_1 b^1 + c_0 b^0 + c_{-1} b^{-1} + c_{-2} b^{-2} + \dots + c_{-m} b^{-m}.$$

Путем соответствующих вычислений выполняется **преобразование** числа $(N)_b$ по основанию b в десятичную систему счисления.

Например:

$$(101,1)_{10} = 1 \cdot 10^2 + 0 \cdot 10^1 + 1 \cdot 10^0 + 1 \cdot 10^{-1} = 101,1;$$

$$(101,1)_2 = 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 + 1 \cdot 2^{-1} = 5,5;$$

$$(101,1)_3 = 1 \cdot 3^2 + 0 \cdot 3^1 + 1 \cdot 3^0 + 1 \cdot 3^{-1} = 10,333...;$$

$$(101,1)_8 = 1 \cdot 8^2 + 0 \cdot 8^1 + 1 \cdot 8^0 + 1 \cdot 8^{-1} = 65,125;$$

$$(101,1)_{16} = 1 \cdot 16^2 + 0 \cdot 16^1 + 1 \cdot 16^0 + 1 \cdot 16^{-1} = 257,0625.$$

Формально десятичная система не имеет никаких особых преимуществ перед другими системами счисления. Предполагается, что эта система была принята в древние времена благодаря тому факту, что в процессе счета в качестве инструмента использовались пальцы рук.

Компьютер может работать в любой системе счисления. В ходе развития вычислительной техники было установлено, что наиболее удобна двоичная система. Предпочтение ей можно отдать по следующим причинам:

- простота правил для арифметических и логических операций;
- физическое представление цифр с целью хранения и обработки чисел осуществляется гораздо легче для двух символов, чем для десяти: перфорирован – не перфорирован, контакт замкнут – контакт разомкнут, наличие или отсутствие тока и т. д.;

- схемы, которые должны различать только два состояния, более надежны в работе, чем схемы, которые должны различать десять состояний.

Отметим, что в процессе развития цивилизации были созданы и **непозиционные системы счисления**. Ярким примером может служить римская система, использующая цифры *I, V, X, L, C, D, M*. Поскольку правила представления чисел и выполнения арифметических операций в таких системах очень сложны, они имеют ограниченное применение.

Вопросы и упражнения

- ❶ Как определяется система счисления?
- ❷ В чем отличие между позиционными и непозиционными системами счисления?
- ❸ Приведите примеры позиционных систем счисления. Как определяется основание системы счисления?
- ❹ Переведите в десятичную систему счисления число $(101,1)_b$, записанное в следующих системах счисления:

$$b = 4, 5, 6, 7, 9, 11, 12, 13, 14 \text{ и } 15.$$

- ❺ Переведите в десятичную систему счисления следующие числа:

a) $(328)_9;$	i) $(1010,01)_3;$	q) $(341,02)_8;$
b) $(516)_7;$	j) $(201,12)_4;$	r) $(ABCD)_{16};$
c) $(1010,01)_2;$	k) $(341,02)_6;$	s) $(328)_{16};$
d) $(201,12)_3;$	l) $(1111)_{16};$	t) $(516)_{16};$
e) $(341,02)_5;$	m) $(328)_{11};$	u) $(1010,01)_{16};$
f) $(FFFF)_{16};$	n) $(516)_9;$	v) $(201,12)_{16};$
g) $(328)_{10};$	o) $(1010,01)_8;$	w) $(341,02)_{12};$
h) $(516)_8;$	p) $(201,12)_8;$	x) $(F001)_{16}.$

- ❻ Какие факторы способствовали использованию в вычислительной технике двоичной системы счисления?
- ❼ Напишите программу, переводящую числа, записанные в системах счисления по основанию b , $b \leq 10$, в десятичную систему счисления.
- ❽ Напишите программу, переводящую числа, записанные в системах счисления по основанию b , $10 \leq b \leq 36$, в десятичную систему счисления.

- 9 ИССЛЕДУЙТЕ! Найдите в Интернете, какие системы счисления использовались великими древними цивилизациями. Как эти системы повлияли на современные цивилизации?

3.2. Перевод чисел из одной системы счисления в другую

Перевод числа $(N)_b$ в его десятичный эквивалент осуществляется в соответствии с формулой из предыдущего параграфа:

$$(N)_b = c_n b^n + c_{n-1} b^{n-1} + \dots + c_1 b^1 + c_0 b^0 + c_{-1} b^{-1} + c_{-2} b^{-2} + \dots + c_{-m} b^{-m}.$$

Перевод десятичного числа $(N)_{10}$ в его эквивалент по основанию b осуществляется по следующим правилам:

– делением на соответствующее основание целой части и целочисленных частных после каждой операции деления до получения нулевого частного; результат перевода целой части составляется из целочисленных остатков, записанных в порядке, обратном их вычислению;

– умножением на основание дробной части, а затем и дробных частей, полученных в предшествующих умножениях, до тех пор пока дробная часть очередного произведения не станет равной нулю или до получения требуемого количества цифр дробной части результата; результат преобразования дробной части состоит из целых частей произведений, записанных в порядке их вычисления.

Проанализируем несколько примеров.

1. Перевести десятичное число 53,40625 в его двоичный эквивалент.

$$\begin{aligned} 53 : 2 &= 26 + \frac{1}{2}; \\ 26 : 2 &= 13 + \frac{0}{2}; \\ 13 : 2 &= 6 + \frac{1}{2}; \\ 6 : 2 &= 3 + \frac{0}{2}; \\ 3 : 2 &= 1 + \frac{1}{2}; \\ 1 : 2 &= 0 + \frac{1}{2}. \end{aligned}$$

Следовательно, целая часть двоичного числа будет 110101.

$$\begin{aligned} 0,40625 \times 2 &= 0,8125; \\ 0,8125 \times 2 &= 1,625; \\ 0,625 \times 2 &= 1,25; \\ 0,25 \times 2 &= 0,5; \\ 0,5 \times 2 &= 1,0. \end{aligned}$$

Дробная часть двоичного числа будет 01101. Следовательно,

$$(53,40625)_{10} = (110101,01101)_2.$$

2. Перевести число 23,7 из десятичной системы в двоичную.

$$\begin{aligned} 23 : 2 &= 11 + \frac{1}{2}; \\ 11 : 2 &= 5 + \frac{1}{2}; \\ 5 : 2 &= 2 + \frac{1}{2}; \end{aligned}$$

$$2 : 2 = 1 + {}^0/2;$$

$$1 : 2 = 0 + {}^1/2;$$

$$0,7 \times 2 = 1,4;$$

$$0,4 \times 2 = 0,8;$$

$$0,8 \times 2 = 1,6;$$

$$0,6 \times 2 = 1,2;$$

$$0,2 \times 2 = 0,4;$$

$$0,4 \times 2 = 0,8;$$

...

Заметим, что операция может быть продолжена до бесконечности, то есть не существует точного перевода анализируемого числа. Следовательно,

$$(23,7)_{10} = (10111,101100\dots)_2.$$

3. Осуществить перевод числа 1996,0625 из десятичной системы в восьмеричную.

$$1996 : 8 = 249 + {}^4/8;$$

$$249 : 8 = 31 + {}^1/8;$$

$$31 : 8 = 3 + {}^7/8;$$

$$3 : 8 = 0 + {}^3/8;$$

$$0,0625 \times 8 = 0,5;$$

$$0,5 \times 8 = 4.$$

Следовательно,

$$(1996,0625)_{10} = (3714,04)_8.$$

4. Перевести число 2914,25 из десятичной системы в шестнадцатеричную.

$$2914 : 16 = 182 + {}^2/16;$$

$$182 : 16 = 11 + {}^6/16;$$

$$11 : 16 = 0 + {}^{11}/16;$$

$$0,25 \times 16 = 4.$$

Следовательно,

$$(2914,25)_{10} = (B62,4)_{16}.$$

Вопросы и упражнения

- ❶ Как осуществляется перевод числа из системы по основанию b в десятичную систему?
- ❷ Переведите в десятичную систему следующие числа:

a) $(100001,01111)_2;$

b) $(328,678)_9;$

c) $(100,100)_{16};$

- | | | |
|----------------------|---------------------|---------------------|
| d) $(10,01)_{16};$ | g) $(1221,1112)_3;$ | j) $(4231,124)_5;$ |
| e) $(AAA,BBB)_{16};$ | h) $(1321,1312)_4;$ | k) $(50,505050)_6;$ |
| f) $(EE,00F)_{16};$ | i) $(124,521)_7;$ | l) $(7777,001)_8.$ |

③ Как выполняется перевод десятичного числа в его эквивалент по основанию b ?

④ Переведите в двоичную систему следующие десятичные числа:

- | | | |
|------------|------------|------------|
| a) 13,889; | e) 93,447; | i) 58,749; |
| b) 38,668; | f) 70,212; | j) 4,345; |
| c) 53,536; | g) 8,347; | k) 3,156; |
| d) 29,261; | h) 39,764; | l) 91,428. |

Проверьте результаты, осуществляя перевод из двоичной в десятичную систему счисления.

⑤ Переведите в восьмеричную систему следующие десятичные числа:

- | | | |
|-------------|-------------|-------------|
| a) 358,932; | e) 886,526; | i) 795,128; |
| b) 479,093; | f) 971,258; | j) 680,895; |
| c) 591,241; | g) 515,914; | k) 256,453; |
| d) 649,113; | h) 347,607; | l) 838,261. |

Проверьте результаты, выполняя перевод из восьмеричной системы в десятичную.

⑥ Переведите в шестнадцатеричную систему следующие десятичные числа:

- | | | |
|--------------|--------------|--------------|
| a) 1424,699; | e) 5818,961; | i) 4985,995; |
| b) 3517,315; | f) 9336,491; | j) 9721,678; |
| c) 9607,201; | g) 3442,722; | k) 5292,837; |
| d) 8974,664; | h) 4521,449; | l) 2734,592. |

Проверьте результаты, выполняя перевод из шестнадцатеричной системы в десятичную.

⑦ Напишите программу, которая переводит десятичные числа в эквиваленты из соответствующих систем счисления по основанию b , $b < 10$.

⑧ Напишите программу, которая переводит десятичные числа в числа систем по основанию b , $10 < b \leq 36$.

3.3. Перевод чисел из двоичной системы счисления в восьмеричную, шестнадцатеричную и обратно

Поскольку $8 = 2^3$, двоично-восьмеричный перевод и восьмерично-двоичный перевод может быть осуществлен напрямую. Любая восьмеричная цифра представляется тремя двоичными цифрами:

$$0 = 000;$$

$$1 = 001;$$

$$2 = 010;$$

$$5 = 101;$$

$$3 = 011;$$

$$6 = 110;$$

$$4 = 100;$$

$$7 = 111.$$

Если дано восьмеричное число, то для его перевода в двоичное каждая восьмеричная цифра заменяется тремя двоичными.

Примеры:

$$(247,315)_8 = (010\ 100\ 111, 011\ 001\ 101)_2;$$

$$(512,07)_8 = (101\ 001\ 010, 000\ 111)_2;$$

$$(3,146)_8 = (011, 001\ 100\ 110)_2.$$

Если рассматривается двоичное число, то для перевода его в восьмеричное группируем по три двоичные цифры, начиная с позиции запятой влево для целой части и вправо соответственно для дробной части, находя эквиваленты этих троек цифр в восьмеричной системе. При дополнении некоторой группы до трех двоичных цифр добавление нулей в начале числа для целой части и соответственно в конце числа для дробной части не меняет значение числа.

Примеры:

$$(11,011101)_2 = (011,011\ 101)_2 = (3,35)_8;$$

$$(10,11011)_2 = (010,110\ 110)_2 = (2,66)_8;$$

$$(1001,01011)_2 = (001\ 001,010\ 110)_2 = (11,26)_8.$$

Аналогично поступаем и в случае шестнадцатеричной системы, основание которой $16 = 2^4$. Любая шестнадцатеричная цифра представляется четырьмя двоичными цифрами:

$$0 = 0000;$$

$$8 = 1000;$$

$$1 = 0001;$$

$$9 = 1001;$$

$$2 = 0010;$$

$$A = 1010;$$

$$3 = 0011;$$

$$B = 1011;$$

$$4 = 0100;$$

$$C = 1100;$$

$$5 = 0101;$$

$$D = 1101;$$

$$6 = 0110;$$

$$E = 1110;$$

$$7 = 0111;$$

$$F = 1111.$$

Примеры:

$$(6AF3,B2)_{16} = (0110\ 1010\ 1111\ 0011, 1011\ 0010)_2;$$

$$(6F1,3CA)_{16} = (0110\ 1111\ 0001, 0011\ 1100\ 1010)_2;$$

$$(11,01101)_2 = (0011, 0110\ 1000)_2 = (3,68)_{16};$$

$$(10001,01011)_2 = (0001\ 0001, 0101\ 1000)_2 = (11,58)_{16}.$$

Вопросы и упражнения

- ❶ Как осуществляется перевод из восьмеричной в двоичную систему счисления и обратно?
- ❷ Переведите в двоичную систему следующие восьмеричные числа:
- | | | |
|------------|------------|------------|
| a) 15,006; | e) 21,626; | i) 42,322; |
| b) 13,06; | f) 6,3415; | j) 44,523; |
| c) 43,15; | g) 771,25; | k) 32,271; |
| d) 10,01; | h) 12,121; | l) 73,536. |
- ❸ Переведите в восьмеричную систему следующие двоичные числа:
- | | | |
|-------------------|------------------|-------------------|
| a) 1,1; | e) 1101,1; | i) 10110,001011; |
| b) 101,10101; | f) 1,000001; | j) 11111,0010001; |
| c) 1111,000101; | g) 11110001,101; | k) 11001,00101; |
| d) 10101110,1001; | h) 0,00001; | l) 1011,0001011. |
- ❹ Напишите программу для перевода чисел из двоичной в восьмеричную систему счисления и из восьмеричной в двоичную систему счисления.
- ❺ Как осуществляется шестнадцатерично-двоичное и двоично-шестнадцатеричное преобразование?
- ❻ Переведите в двоичную систему следующие шестнадцатеричные числа:
- | | | |
|-------------|----------------|-------------|
| a) FFF,AAA; | e) 3,1AB; | i) C,DC1; |
| b) F,1A; | f) AABB,0000F; | j) 942,14A; |
| c) 1,009; | g) 81,91A; | k) CAA,B; |
| d) 0,00F; | h) 10,01; | l) DAD,ABA. |
- ❼ Переведите в двоичную систему следующие шестнадцатеричные числа:
- | | | |
|-----------------------|--------------------|----------------|
| a) 1001011101,01101; | e) 1011101,011; | i) 1011,0101; |
| b) 111,01; | f) 11,001011101; | j) 11001,0110; |
| c) 1,1; | g) 11110,01111; | k) 0,00001; |
| d) 10101110111,00101; | h) 111011,0010000; | l) 101,01011. |
- ❽ Напишите программу для перевода чисел из двоичной в шестнадцатеричную систему счисления и из шестнадцатеричной в двоичную систему счисления.
- ❾ Переведите в шестнадцатеричную систему следующие восьмеричные числа:
- | | | |
|-------------|-------------|-------------|
| a) 13,146; | e) 451,35; | i) 644,031; |
| b) 613,12; | f) 12,357; | j) 5,4312; |
| c) 541,723; | g) 53,627; | k) 675,542; |
| d) 104,527; | h) 572,004; | l) 372,271. |

10 Переведите в восьмеричную систему следующие шестнадцатеричные числа:

a) AA;

e) 15F,6A1;

i) BBB;

b) A2B,1F;

f) 3,281;

j) AF,31B;

c) F,3A;

g) 9,AF;

k) 2C,ACB;

d) FFFF;

h) 3,418F;

l) A1B,39E.

11 Напишите программу для восьмерично-шестнадцатеричного и шестнадцатерично-двоичного перевода чисел.

3.4. Арифметические операции в двоичной системе счисления

Арифметические операции над двоичными числами очень просты. Правила действий в двоичной системе представлены в *таблицах 3.2, 3.3 и 3.4*.

Таблица 3.2
Двоичное сложение

$0 + 0 = 0$
$0 + 1 = 1$
$1 + 0 = 1$
$1 + 1 = 10$

Таблица 3.3
Двоичное вычитание

$0 - 0 = 0$
$1 - 0 = 1$
$1 - 1 = 0$
$10 - 1 = 1$

Таблица 3.4
Двоичное умножение

$0 \times 0 = 0$
$0 \times 1 = 0$
$1 \times 0 = 0$
$1 \times 1 = 1$

Примеры:

1. Выполните сложение десятичных чисел 29 и 43 в двоичной системе счисления:

$$(29)_{10} = (11101)_2;$$

$$(43)_{10} = (101011)_2;$$

11101 +
101011
1001000

Проверка: $(1001000)_2 = (72)_{10}$, результат верен, поскольку $(29)_{10} + (43)_{10} = (72)_{10}$.

2. Выполните вычитание десятичного числа 37 из десятичного числа 46 в двоичной системе счисления:

$$(37)_{10} = (100101)_2;$$

$$(46)_{10} = (101110)_2;$$

101110 _
100101
1001

Проверка: $(1001)_2 = (9)_{10}$, результат верен, поскольку $(46)_{10} - (37)_{10} = (9)_{10}$.

3. Выполните умножение десятичных чисел 3,25 и 7,125 в двоичной системе счисления:

$$(3,25)_{10} = (11,01)_2;$$

$$(7,125)_{10} = (111,001)_2;$$

11,01 ×
111,001

1101
0000
0000
1101
1101
1101

10111,00101

Проверка: $(10111,00101)_2 = (23,15625)_{10}$, результат верен, поскольку $(3,25)_{10} \times (7,125)_{10} = (23,15625)_{10}$.

4. Выполните деление десятичного числа 211 на десятичное число 3 в двоичной системе счисления:

$$(211)_{10} = (11010011)_2;$$

$$(3)_{10} = (11)_2;$$

11010011	11
11	-----
00100	1000110
11	
11	
11	

01	

Следовательно $11010011 : 11 = 1000110$, остаток 1.

Проверка: $(1000110)_2 = (70)_{10}$, результат верен, поскольку $(211)_{10} : (3)_{10} = (70)_{10} + (1)_{10}$.

Заметим, что как при умножении, так и при делении установка запятой, отделивающей целую часть от дробной, осуществляется так же, как и в десятичной системе счисления.

Вопросы и упражнения

- Как выполняются арифметические операции в двоичной системе счисления?
- Вычислите в двоичной системе:

a) $34 + 251;$

f) $1996 - 51;$

k) $0,5 \times 0,5;$

b) $68 - 7;$

g) $2015 + 1995;$

l) $1 : 0,5;$

c) $1512 + 620;$

h) $28,5 + 0,75;$

m) $40 : 0,125;$

d) $14 \times 8;$

i) $63,125 - 4,125;$

n) $32 : 2;$

e) $63 : 3;$

j) $3,0625 \times 2,125;$

o) $32 : 16;$

- p) $401 \times 8;$ r) $401 \times 4;$ t) $401 \times 2;$
 q) $32 : 8;$ s) $32 : 4;$ u) $933 : 3.$

Числа в приведенных выражениях записаны в десятичной системе счисления.

- ③ Напишите программу для сложения и вычитания двоичных чисел.
- ④ Напишите программу для умножения и деления двоичных чисел.

3.5. Представление натуральных чисел в компьютере

Современные компьютеры используют двоичную систему счисления. Представление натуральных чисел $N = \{0, 1, 2, \dots\}$ реализуется на фиксированном числе двоичных позиций, как правило, 8, 16, 32 или 64 (рис. 3.1).

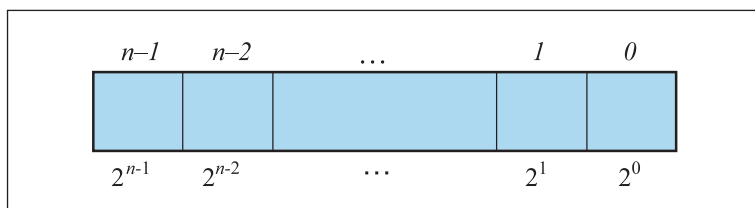


Рис. 3.1. Представление натуральных чисел на n двоичных позициях

В позициях $0, 1, \dots, n - 1$ записаны двоичные цифры натурального числа, представленного в двоичной системе счисления. Выравнивание двоичных чисел выполняется вправо, возможные незначащие нули размещаются перед двоичным числом.

Для примера на рис. 3.2 приведено представление натурального числа

$$1039 = (10000001111)_2$$

на 16 двоичных позициях.

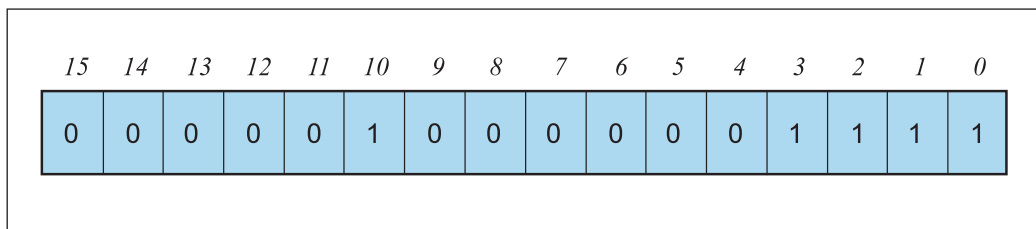


Рис. 3.2. Представление натурального числа 1039 на 16 двоичных позициях

Максимальное число, которое может быть представлено на n двоичных позициях (рис. 3.3),

$$1 \cdot 2^0 + 1 \cdot 2^1 + 1 \cdot 2^2 + \dots + 1 \cdot 2^{n-1} = 2^n - 1.$$

Следовательно, на n двоичных позициях могут быть представлены натуральные числа из интервала $[0; 2^n - 1]$.

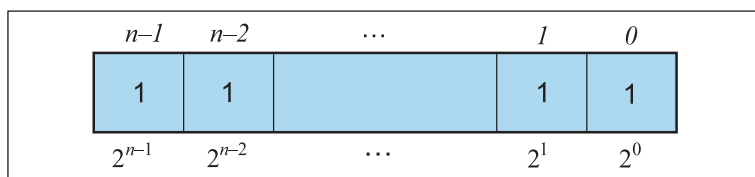


Рис. 3.3. Представление максимального натурального числа на n двоичных позициях

Вопросы и упражнения

- ❶ Как представляются натуральные числа в компьютере?
- ❷ Представьте натуральные числа 3, 112, 191, 204, 255 на 8 двоичных позициях.
- ❸ Представьте натуральные числа 3, 255, 1024, 2048, 4096, 65535 на 16 двоичных позициях.
- ❹ Вычислите максимальные натуральные числа, которые могут быть представлены на 4, 8, 12, 16, 24, 32 и 64 двоичных позициях.
- ❺ ИССЛЕДУЙТЕ! Определите, на скольких двоичных позициях представляются натуральные числа в цифровых устройствах, с которыми вы работаете.

3.6. Представление целых чисел

В компьютере отсутствует возможность прямого представления знаков $+$ и $-$, принадлежащих положительным и отрицательным числам. По этой причине представление знака числа x осуществляется с помощью особой двоичной цифры, именуемой **цифрой знака** (или знаковым битом), расположенной в позиции $n - 1$ (рис. 3.4):

$$S = \begin{cases} 0, & x > 0; \\ 1, & x < 0. \end{cases}$$

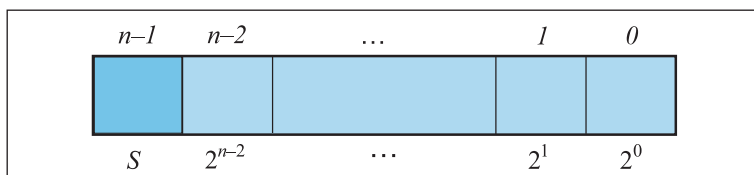


Рис. 3.4. Представление целых чисел на n двоичных позициях

Как правило, двоичные числа со знаком представляются в компьютерах не в исходном виде, а в специальных кодах, дающих определенные преимущества при выполнении арифметических операций. С этой точки зрения известны три способа представления, называемые **двоичными кодами для алгебраических чисел**.

Прямой код (код *величина и знак*). Представление любого числа в этом коде очень простое: в цифре знака записывается 0, если число положительное, и 1, если оно отрицательное; в значащей части записывается число (по модулю) в обычной двоичной системе.

Для примера на рис. 3.5 (с. 126) приведены представления чисел $+52$ и -52 в прямом коде на 8 двоичных позициях.

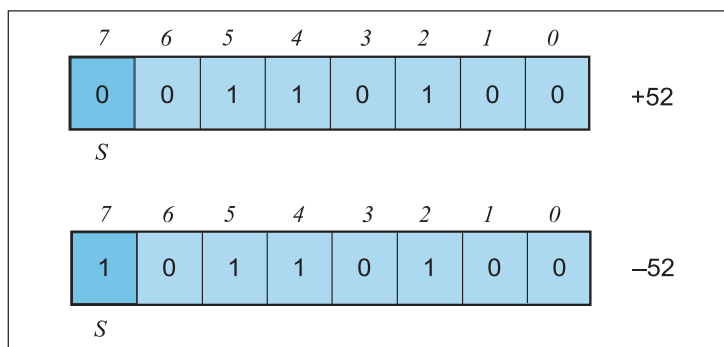


Рис. 3.5. Представление чисел +52 и -52 в прямом коде

В прямом коде на n двоичных позициях могут быть представлены целые положительные и отрицательные числа N , для которых:

$$-(2^{n-1} - 1) \leq N \leq (2^{n-1} - 1).$$

Заметим, что в прямом коде существуют два двоичных представления для числа 0: 00...0 и 10...0. Прямой код редко используется в компьютерах, поскольку требует сложных алгоритмов выполнения арифметических операций и проверки результатов.

Обратный (инверсный) код. Положительные числа записываются в инверсном коде точно так же, как и в прямом. Если число отрицательное, то сначала оно записывается как положительное, а затем инвертируется каждая двоичная цифра, то есть 1 становится 0 и 0 становится 1. Отсюда и название — обратный код.

На рис. 3.6 приведены представления чисел +52 и -52 в обратном коде на 8 двоичных позициях.

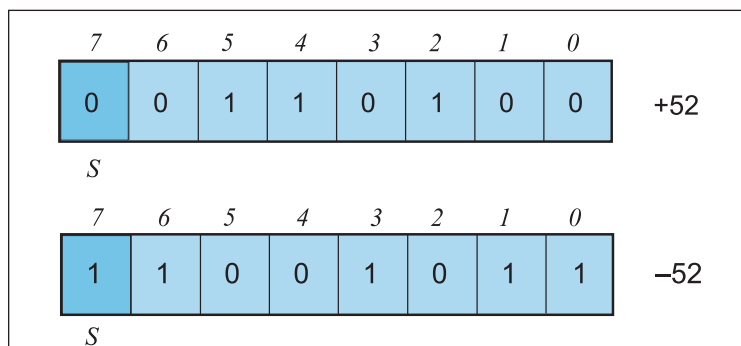


Рис. 3.6. Представление чисел +52 и -52 в обратном коде

Можно легко установить, что интервал целых чисел N , которые могут быть представлены в обратном коде, такой же, как и для прямого кода, а число 0 имеет два двоичных представления: 00...0 и 11...1.

Дополнительный код. В этом коде положительные числа имеют точно такое же представление, как в прямом и инверсном кодах. Если число отрицательное, оно сначала записывается в обратном коде, а затем к младшей значащей цифре (двоичная позиция 0) прибавляется 1.

Для примера на *рис. 3.7* приведены представления чисел +52 и -52 в дополнительном коде.

7	6	5	4	3	2	1	0	
0	0	1	1	0	1	0	0	+52
S								
7	6	5	4	3	2	1	0	
1	1	0	0	1	0	1	1	-52 в обратном коде
S								
7	6	5	4	3	2	1	0	
1	1	0	0	1	1	0	0	-52 в дополнительном коде
S								

Рис. 3.7. Представление чисел +52 и -52 в дополнительном коде

Дополнительный код используется в большинстве компьютеров благодаря простоте выполнения в нем арифметических операций и легкости проверки результатов. В этом коде на n двоичных позициях могут быть представлены целые числа из интервала $[-2^{n-1}, 2^{n-1} - 1]$.

Вопросы и упражнения

- Как могут быть представлены в компьютере целые числа? Объясните, как записываются отрицательные числа в прямом, инверсном и дополнительном кодах.
- Представьте в прямом коде на 8 двоичных позициях:

a) +12;	d) -12;	g) +21;
b) -21;	e) -64;	h) -68;
c) +68;	f) +105;	i) -112.
- Представьте в обратном коде на 8 двоичных позициях:

a) +10;	d) -10;	g) +65;
b) -65;	e) +101;	h) -101;
c) +112;	f) -112;	i) -105.
- Представьте в дополнительном коде на 8 двоичных позициях:

a) +40;	c) +109;	e) -40;
b) +27;	d) -16;	f) -27;

g) -109;

i) +16;

k) +111;

h) -101;

j) +101;

l) -111.

- 5 Напишите программу, которая считывает с клавиатуры целые числа и выводит на экран их представления в прямом, обратном и дополнительном кодах для $n = 8, 16$ и 32 .
- 6 Как представляется число 0 в прямом, обратном и дополнительном кодах? Сколько представлений есть у числа 0 в рассматриваемых кодах?
- 7 Определите максимальное число, которое может быть представлено на n двоичных позициях в прямом, обратном и дополнительном кодах.
- 8 ИССЛЕДУЙТЕ! Узнайте, на скольких позициях и какими двоичными кодами для алгебраических чисел представляются целые числа в цифровых устройствах, с которыми вы работаете.

3.7. Представление вещественных чисел

Вещественные числа представляются в компьютере в формате с фиксированной или плавающей запятой (в экспоненциальной форме).

Представление с фиксированной запятой. В этом представлении считается, что у всех чисел запятая расположена в одной и той же позиции, хотя это и не соответствует внешней форме представления чисел. Процесс преобразования из внешней формы во внутреннюю и обратно реализуется путем выбора программистом соответствующих масштабных коэффициентов.

Обычно считается, что запятая размещается сразу после цифры знака — случай, когда числа содержат только дробную часть (рис. 3.8).

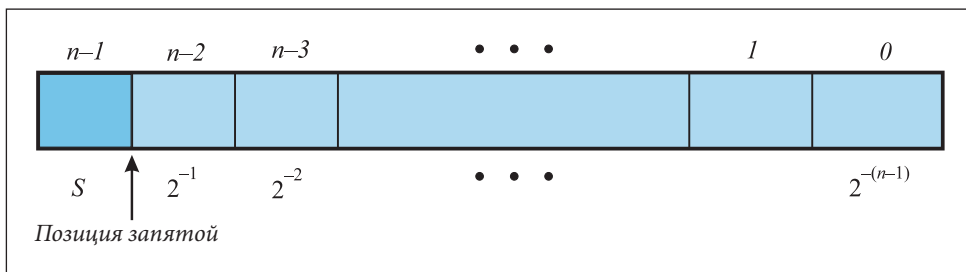


Рис. 3.8. Представление вещественных чисел в формате с фиксированной запятой

Собственно, запятая не материализована физически в компьютере. Из рис. 3.8 следует, что на n двоичных позициях можно представить вещественные числа, абсолютное значение которых

$$0,00\dots0 \leq |x| \leq 0,11\dots1$$

или в десятичной системе счисления,

$$0 \leq |x| \leq 1 - 2^{-(n-1)}.$$

Как и в случае целых чисел, вещественные числа, меньшие единицы, могут быть представлены с некоторыми изменениями в прямом, обратном или дополнительном кодах.

Для примера на рис. 3.9 приведено представление чисел

$$\begin{aligned} +0,9375 &= +15/16 = (0,1111)_2, \\ -0,9375 &= -15/16 = (-0,1111)_2 \end{aligned}$$

в формате с фиксированной запятой на 8 двоичных позициях в прямом коде.

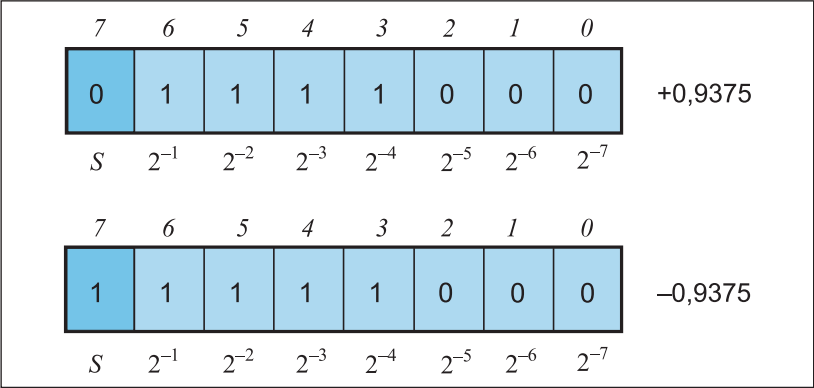


Рис. 3.9. Представление чисел +0,9375 и -0,9375 в формате с фиксированной запятой в прямом коде

Главное преимущество представления с фиксированной запятой состоит в том, что арифметические операции над вещественными числами могут быть выполнены арифметическим устройством, предназначенным для обработки целых чисел.

Представление с плавающей запятой. Операции над числами с фиксированной запятой удобны для однородных потоков данных, когда все вещественные числа по модулю меньше 1. Однако такое представление не эффективно в научных расчетах при одновременной работе с очень большими и очень малыми числами, порядок которых заранее непредсказуем. Для таких задач используется представление с плавающей запятой.

Числа, представляемые в формате с плавающей запятой, могут быть как целыми, так и дробными, а их значение задается соотношением:

$$x = M \times b^E,$$

где b — основание, M — мантисса, а E — экспонента. В современных компьютерах используется $b = 2$ или 16 , а $|M| \leq 1$.

Число, представленное в формате с плавающей запятой, является нормализованным, если первая после запятой цифра мантиссы отлична от нуля.

Примеры:

1. Число 23 выражается в формате с плавающей запятой так:

$$23 = (10111)_2 = 0,10111 \times 2^5,$$

где $M = 0,10111$; $b = 2$; $E = 5$.

2. Число 4,9375 записывается в формате с плавающей запятой следующим образом:

$$4,9375 = (100,1111)_2 = 0,1001111 \times 2^3,$$

$$M = 0,1001111; b = 2; E = 3.$$

3. Число $-0,375$ записывается в формате с плавающей запятой так:

$$-0,375 = (-0,011)_2 = -0,11 \times 2^{-1},$$

$$M = -0,11; b = 2; E = -1.$$

Отметим, что реальная позиция запятой внутри числа зависит от значения экспоненты, то есть запятая меняет свое положение (отсюда и название плавающая запятая).

Существует несколько вариантов представления мантиисы и экспоненты на n двоичных позициях. На рис. 3.10 приведено представление с плавающей запятой **в формате экспонента-мантииса**.

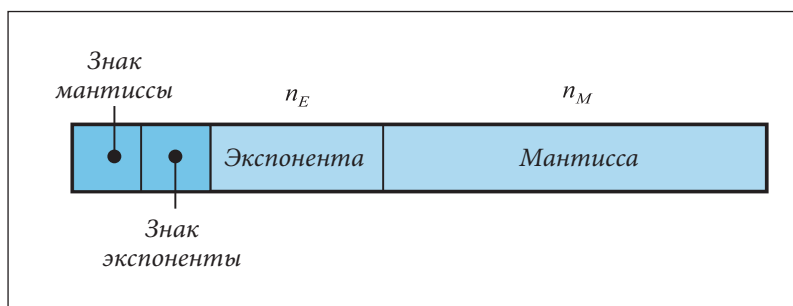


Рис. 3.10. Представление с плавающей запятой в формате экспонента-мантииса

Количество двоичных позиций n_E , выделенных экспоненте, устанавливает область значений чисел, которые могут быть представлены, тогда как количество битов для мантиисы n_M определяет точность представления числа. В современных компьютерах $n_E = 6 \dots 15$ и $n_M = 24 \dots 64$, что обеспечивает представление чисел в очень широком диапазоне.

На рис. 3.11 приведено представление чисел в формате **характеристика-мантииса**.

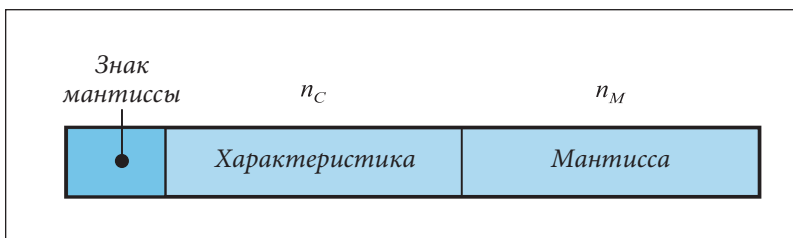


Рис. 3.11. Представление с плавающей запятой в формате характеристика-мантииса

Характеристика C — это форма представления экспоненты E . Она определяется соотношением

$$C = E + K.$$

В персональных компьютерах

$$K = 2^{n_c - 1} - 1,$$

где n_c — это количество двоичных позиций, выделенных для характеристики. В этом представлении характеристика в отличие от экспоненты не может принимать отрицательных значений, что упрощает арифметические устройства.

В частности, для $n = 8$ характеристика вычисляется так:

$$C = E + 127,$$

указывая практически знак экспоненты:

– если $C \geq 127$, тогда $E \geq 0$;

– если $C < 127$, тогда $E < 0$.

В таблице 3.5 представлены форматы характеристика–мантисса, используемые в большинстве персональных компьютеров.

Таблица 3.5

Форматы характеристика–мантисса, используемые в персональных компьютерах

Название формата	n	n_c	n_m	Точность мантиссы, десятичные цифры	Область возможных значений чисел
Обычная точность	32	8	23	6 sau 7	$10^{-37} \dots 10^{38}$
Двойная точность	64	11	52	15 sau 16	$10^{-307} \dots 10^{308}$
Расширенная точность	80	15	64	19	$10^{-4932} \dots 10^{4932}$

Поскольку в соответствии с условием нормализации первая цифра мантиссы всегда равна 1, эта цифра может быть представлена или не представлена в памяти компьютера. В случае, когда она не представлена, указанная цифра называется **скрытым битом**. Отметим, что техника скрытого бита относится только к представлению чисел в памяти, но не к операциям, выполняемым арифметическим устройством.

Вопросы и упражнения

- Какие различия имеют представления с фиксированной и плавающей запятой?
- Представьте в формате с фиксированной запятой на 8 двоичных позициях в прямом коде:

a) +0,875;

e) -0,875;

i) +0,125;

b) -0,125;

f) +0,3;

j) -0,3;

c) +0,4;

g) -0,4;

k) +0,15625;

d) -0,15625;

h) +0,21875;

l) -0,21875.

- Укажите преимущества и недостатки у представления с фиксированной запятой.

- 4 Как представляются вещественные числа в формате с плавающей запятой? Какие числа, представленные в формате с плавающей запятой, удовлетворяют условию нормализации?
- 5 Как выполняется нормализация чисел, представленных в формате с плавающей запятой?
- 6 Представьте в формате с плавающей запятой следующие числа:
- | | | |
|-------------|------------|-------------|
| a) +1,5; | e) -1,5; | i) -0,0625; |
| b) +0,0625; | f) +3,25; | j) +2,7; |
| c) -3,25; | g) -32,87; | k) -2,7; |
| d) -6,25; | h) +6,25; | l) -0,147. |
- 7 От чего зависят точность и область значений чисел, представленных в формате с плавающей запятой?
- 8 Каковы различия между форматами *экспонента-мантисса* и *характеристика-мантисса*? Как рассчитываются характеристики чисел, представленных в формате с плавающей запятой?
- 9 Вычислите характеристики чисел из упражнения 6. Предполагается, что $n_c = 8$.
- 10 Объясните термин *скрытый бит*. Каковы преимущества такого представления?
- 11 **ИССЛЕДУЙТЕ!** Узнайте, как представляются натуральные, целые и вещественные числа в цифровых устройствах, с которыми вы работаете? Определите количество двоичных позиций, используемых каждым представлением.

4.1. Логические переменные и выражения

Булева алгебра, или **алгебра логики** является составной частью математики. В ней законы мышления — объект изучения классической логики — изучаются с помощью символических методов. Такое название она получила в честь английского математика *Джорджа Буля*, который в своей работе *The Laws of Thought* (Законы мышления), опубликованной в 1853 году, заложил основы алгебры логики. Долгие годы булева алгебра считалась просто математическим курьезом, неприменимым к практическим областям науки. Она снова стала актуальной с появлением автоматических телефонных сетей и цифровых компьютеров.

С формальной точки зрения булева алгебра может быть определена множеством элементов $\{0, 1\}$, множеством элементарных операций $\neg, \&, \vee$ и определенным набором постулатов. Следовательно, в алгебре логики любая переменная может иметь только одно из двух возможных значений, обозначаемых условно 0 и 1, причем другие значения являются недопустимыми.

Переменные алгебры логики обозначаются через x, y, z, x_1, x_2, \dots с индексами или без них, а элементы 0 и 1 называются **логическими константами**.

Элементарные операции алгебры логики имеют следующие названия:

- \neg — отрицание (инверсия, логическая операция *НЕ*);
- $\&$ — конъюнкция (логическое произведение, логическая операция *И*);
- \vee — дизъюнкция (логическая сумма, логическая операция *ИЛИ*).

Элементарные операции определяются с помощью специальных таблиц, называемых **таблицами истинности**.

Таблица истинности — это таблица, включающая все возможные комбинации значений переменных, для которых определен оператор, и результаты соответствующей операции.

На рис. 4.1 представлены таблицы истинности отрицания, конъюнкции и дизъюнкции.

x	\bar{x}
0	1
1	0

x	y	$x \& y$
0	0	0
0	1	0
1	0	0
1	1	1

x	y	$x \vee y$
0	0	0
0	1	1
1	0	1
1	1	1

Рис. 4.1. Таблицы истинности элементарных логических операторов

Отметим, что в булевой алгебре отрицание обозначается горизонтальной линией над соответствующей переменной.

Поскольку переменная x может принимать значения 0 или 1, таблица истинности отрицания содержит только две строки. В случае конъюнкции и дизъюнкции таблицы истинности содержат четыре строки — по одной строке для каждой возможной комбинации 00, 01, 10 и 11 значений переменных x и y .

Логические переменные и константы, объединенные с помощью операторов \neg , $\&$ и \vee , образуют логические выражения, например:

$$1) \quad x \& y \vee z;$$

$$4) \quad 1 \& x \vee y;$$

$$2) \quad x \vee y \& z;$$

$$5) \quad 0 \vee x \vee y.$$

$$3) \quad \bar{x} \& y \vee z;$$

$$6) \quad 1 \vee 0.$$

Значения логических выражений могут быть вычислены с помощью таблиц истинности элементарных операторов. Для вычисления выражений устанавливается следующий **приоритет логических операторов**:

- 1) отрицание;
- 2) конъюнкция;
- 3) дизъюнкция.

Например, в выражении

$$x \vee y \& z$$

сначала выполняется операция $\&$, а после нее — операция \vee . В выражении

$$x \& y \vee \bar{z}$$

выполняется отрицание, затем — конъюнкция и в конце — дизъюнкция.

Порядок выполнения логических операторов может быть изменен с помощью скобок „(“ и „)”. Ясно, что в первую очередь выполняются операции, заключенные в скобки.

Например, в случае логического выражения

$$x \vee y \& \bar{x} \vee z$$

выполняются отрицание, конъюнкция и затем соответствующие операции дизъюнкции. В случае выражения

$$(x \vee y) \& (\bar{x} \vee z)$$

после отрицания выполняются дизъюнкции и в конце — конъюнкция.

Для систематизации вычисление логических выражений выполняется с помощью специальных таблиц, называемых **таблицами истинности логических выражений**.

Таблица истинности логического выражения включает все возможные комбинации значений переменных рассматриваемого выражения и результаты логических операций в порядке их вычисления.

Для примера на рис. 4.2 представлена таблица истинности выражения

$$\bar{x} \& y \vee z,$$

а на рис. 4.3 — таблица истинности выражения

$$\overline{x \& y \vee z}.$$

x	y	z	\bar{x}	$\bar{x} \& y$	$\bar{x} \& y \vee z$
0	0	0	1	0	0
0	0	1	1	0	1
0	1	0	1	1	1
0	1	1	1	1	1
1	0	0	0	0	0
1	0	1	0	0	1
1	1	0	0	0	0
1	1	1	0	0	1

Рис. 4.2. Таблица истинности логического выражения $\bar{x} \& y \vee z$

x	y	z	$x \& y$	$x \& y \vee z$	$x \& y \vee \bar{z}$
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	0	1	0
1	0	0	0	0	1
1	0	1	0	1	0
1	1	0	1	1	0
1	1	1	1	1	0

Рис. 4.3. Таблица истинности логического выражения $x \& y \vee \bar{z}$

Отметим, что с целью упрощения записей в логических выражениях разрешается опускать символ оператора $\&$. Например, логические выражения

$$\bar{x} \& y \vee z,$$

$$(x \vee y) \& (\bar{x} \vee z)$$

могут быть записаны в виде:

$$\bar{x}y \vee z,$$

$$(x \vee y)(\bar{x} \vee z).$$

Вопросы и упражнения

- ❶ Какие значения могут принимать логические переменные?
- ❷ Какие элементарные операторы используются в алгебре логики и как они определяются?
- ❸ Запомните таблицы истинности отрицания, конъюнкции и дизъюнкции.
- ❹ Как составляются логические выражения? Укажите приоритет логических операций?
- ❺ Объясните роль скобок при вычислении логических выражений.

6 Составьте таблицы истинности следующих логических выражений:

- | | |
|----------------------------|------------------------------|
| a) $\bar{x}y$; | h) $\bar{x} \vee \bar{y}$; |
| b) $\bar{x} \vee y$; | i) xyz ; |
| c) $\overline{x \vee y}$; | j) $x \vee y \vee z$; |
| d) $\bar{\bar{x}}$; | k) $xy \vee z$; |
| e) $\bar{x}x$; | l) $x(y \vee z)$; |
| f) \overline{xy} ; | m) $\bar{x} \vee y \vee z$; |
| g) $\bar{x} \vee x$; | n) $\bar{x}(y \vee z)$. |

7 Напишите программу, которая считывает с клавиатуры значения логических переменных x , y , z и выводит на экран значения следующих выражений:

- | | |
|-------------------------------|-----------------------------------|
| a) \bar{x} ; | f) $x \vee \bar{y} \vee z$; |
| b) xy ; | g) $xy \vee z$; |
| c) $x \vee y$; | h) $\overline{x \vee y \vee z}$; |
| d) $\overline{(x \vee y)}z$; | i) $x \vee xy$; |
| e) $\bar{x}y \vee z$; | j) $\bar{x} \vee y \vee z$. |

8 Напишите программу, которая выводит на экран таблицы истинности конъюнкции и дизъюнкции.

9 Составьте циклический алгоритм, который формирует все возможные комбинации значений переменных x , y и z . Выведите соответствующие комбинации на экран.

10 Для каждого из нижеприведенных логических выражений напишите программу, которая выводит на экран соответствующую таблицу истинности:

- | | |
|----------------------------|-----------------------------------|
| a) $x \vee \bar{x}$; | f) $\bar{x}y$; |
| b) $x\bar{x}$; | g) $x \vee y \vee z$; |
| c) $\overline{x \vee y}$; | h) xyz ; |
| d) $\bar{x}y$; | i) $(x \vee y)(\bar{x} \vee y)$; |
| e) $\bar{x} \vee y$; | j) $(x \vee y)(x \vee \bar{y})$. |

11 ТВОРИТЕ! Соберите необходимую информацию и напишите эссе о жизни, учебе и профессиональной карьере английского математика *Джорджа Буля*.

12 УЗНАЙТЕ! В отличие от двоичной логики, то есть логики, которая работает только с двумя возможными значениями, существуют ее расширения, которые работают с тремя и даже более значениями. Исследуйте различные источники информации, узнайте эти расширения и области, в которых они используются.

4.2. Логические функции

Понятие **логической**, или **булевой функции** определяется таким же образом, как и в случае классической алгебры.

Обозначим через x_1, x_2, \dots, x_n произвольную группу булевых переменных, где $n = 1, 2, 3, \dots$.

Поскольку каждая булева переменная может иметь только значения 0 или 1, количество всевозможных комбинаций значений переменных x_1, x_2, \dots, x_n составляет 2^n .

Очевидно, для $n = 1$ имеем две комбинации (0 и 1); для $n = 2$ существуют $2^2 = 4$ комбинации (00, 01, 10 и 11); для $n = 3$ существуют $2^3 = 8$ комбинаций (000, 001, 010, 011, 100, 101, 110 и 111) и т. д.

Логическая функция n переменных $y = f(x_1, x_2, \dots, x_n)$ есть отображение, которое ставит в соответствие каждой комбинации значений переменных x_1, x_2, \dots, x_n значение 0 или 1 переменной y .

Переменные x_1, x_2, \dots, x_n называются **независимыми переменными**, или **аргументами**, а переменная y – **зависимая переменная**, или **функция** аргументов x_1, x_2, \dots, x_n .

Следовательно, **область определения** функции $y = f(x_1, x_2, \dots, x_n)$ – это множество всевозможных комбинаций

0 0 ... 0
0 0 ... 1
...
1 1 ... 1

значений аргументов x_1, x_2, \dots, x_n (всего 2^n комбинаций), а **областью значений** логической функции является множество $\{0, 1\}$.

Как и в случае классической алгебры, логические функции могут быть определены с помощью таблиц, формул и графических методов.

Таблица истинности логической функции $y = f(x_1, x_2, \dots, x_n)$ – это таблица, включающая все возможные комбинации значений аргументов x_1, x_2, \dots, x_n и соответствующие им значения зависимой переменной y .

Например, на рис. 4.4 представлена таблица истинности некоторой логической функции трех переменных.

x_1	x_2	x_3	y
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

Рис. 4.4. Таблица истинности некоторой логической функции трех переменных

Таблица состоит из $2^3 = 8$ строк и 2 столбцов: первого – для всевозможных комбинаций значений аргументов x_1, x_2, x_3 и второго – для соответствующих значений зависимой переменной y . В рассматриваемой таблице комбинации $x_1 = 0, x_2 = 0, x_3 = 0$ соответствует значение $y = f(0,0,0) = 1$; комбинации $x_1 = 0, x_2 = 0, x_3 = 1$ – значение $y = f(0,0,1) = 0$ и т. д.

Определение логических функций с помощью формул осуществляется присваиванием зависимой переменной y значений логического выражения, содержащего аргументы x_1, x_2, \dots, x_n .

Например:

$$1) \quad y = x;$$

$$2) \quad y = x_1 x_2;$$

$$3) \quad y = \bar{x}_1 x_2 \vee x_3;$$

$$4) \quad y = x_1 x_2 \vee x_3.$$

Очевидно, что, зная формулу логической функции, можно составить ее таблицу истинности. Например, таблица истинности функции

$$y = \overline{x_1 x_2 \vee x_3}$$

будет такой же, как и приведенная на рис. 4.4. Чтобы удостовериться в этом, достаточно составить таблицу истинности логического выражения

$$\overline{x_1 x_2 \vee x_3},$$

представленную в других обозначениях на рис. 4.3 из параграфа 4.1.

Существуют также **графические методы определения логических функций**. Эти методы основываются на диаграммах, применяемых в теории множеств, и изучаются в углубленных курсах информатики.

Вопросы и упражнения

- ❶ Сформулируйте определение понятия *логическая функция*.
- ❷ Укажите область определения и область значений логической функции.
- ❸ Какими способами может быть определена логическая функция n переменных?
- ❹ Как составляется таблица истинности логической функции n переменных? Сколько строк содержит такая таблица?
- ❺ Как можно составить таблицу истинности логической функции, если известна ее формула?
- ❻ Логическая функция трех переменных $y = f(x_1, x_2, x_3)$ определена с помощью таблицы истинности, приведенной на рис. 4.4. Укажите комбинации значений аргументов x_1, x_2, x_3 , для которых рассматриваемая функция имеет значение $y = 1$. Укажите соответствующие комбинации для значения функции $y = 0$.
- ❼ Составьте таблицы истинности следующих функций:

$$a) \quad y = x;$$

$$b) \quad y = \bar{x};$$

$$c) \quad y = x_1 x_2;$$

$$d) \quad y = x_1 \vee x_2;$$

$$e) \quad y = \overline{x_1 x_2};$$

$$f) \quad y = \overline{x_1 \vee x_2};$$

$$g) \quad y = \overline{x_1 x_2 x_3};$$

$$h) \quad y = x_1 (x_2 \vee \bar{x}_3);$$

$$i) \quad y = \bar{x}_1 \vee x_2 x_3;$$

$$j) \quad y = x_1 \vee \overline{x_2 x_3}.$$

- 8 Напишите программы, которые считывают с клавиатуры значения логических переменных x_1, x_2, x_3, x_4 и выводят на экран значения одной из следующих функций:

a) $y = x_1 x_2 \vee x_3 x_4;$

g) $y = x_1 \bar{x}_2 \vee x_3 \bar{x}_4;$

b) $y = (x_1 \vee x_2)(x_3 \vee x_4);$

h) $y = \bar{x}_1 x_2 \vee \bar{x}_3 x_4;$

c) $y = \overline{x_1 x_2 x_3 x_4};$

i) $y = \bar{x}_1 \bar{x}_2 \vee \bar{x}_3 \bar{x}_4;$

d) $y = \overline{x_1 \vee x_2 \vee x_3 \vee x_4};$

j) $y = x_1 x_2 x_3 x_4 \vee \bar{x}_2 x_3 x_4;$

e) $y = \overline{x_1 x_2} \vee \overline{x_3 x_4};$

k) $y = x_1 \vee x_2 x_3 \vee x_4;$

f) $y = \overline{(x_1 \vee x_2)(x_3 \vee x_4)};$

l) $y = x_1 \vee x_2 \vee x_3 \vee \bar{x}_1 \bar{x}_2 \bar{x}_3 \bar{x}_4.$

- 9 Напишите программы, которые составляют таблицы истинности следующих функций:

a) $y = x;$

h) $y = \bar{x}_1 \bar{x}_2;$

b) $y = \bar{x};$

i) $y = \overline{x_1 x_2 x_3};$

c) $y = x_1 x_2;$

j) $y = \bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3;$

d) $y = x_1 \vee x_2;$

k) $y = \overline{x_1 \vee x_2 \vee x_3};$

e) $y = \overline{x_1 x_2};$

l) $y = \bar{x}_1 \bar{x}_2 \bar{x}_3;$

f) $y = \bar{x}_1 \vee \bar{x}_2;$

m) $y = x_1 \vee x_2 \vee \bar{x}_1 \bar{x}_2 \bar{x}_3 \bar{x}_4;$

g) $y = \overline{x_1 \vee x_2};$

n) $y = x_1(x_2 \vee \bar{x}_3 \vee \bar{x}_4).$

4.3. Часто используемые логические функции

Рассмотрим n независимых переменных x_1, x_2, \dots, x_n . Возникает вопрос, сколько логических функций n переменных существует в булевой алгебре? **Количество возможных логических функций** можно получить путем следующих рассуждений.

Поскольку любая логическая функция может быть определена с помощью таблицы истинности, то число возможных функций n переменных совпадает с количеством различных таблиц истинности.

Для того чтобы определить логическую функцию, в столбце y таблицы истинности указываются значения функции – 0 или 1 для каждой из всевозможных 2^n комбинаций значений аргументов. Поскольку в таблице истинности 2^n строк, существует

$$m = 2^{2^n}$$

логических функций n переменных. Соответствующие функции обозначаются $y_j, j = 0, 1, \dots, m - 1$.

Например, в случае, когда $n = 1$, существует $m = 2^{2^1} = 2^2 = 4$ логических функций, представленных на рис. 4.5.

x	y_0	y_1	y_2	y_3
0	0	1	0	1
1	0	0	1	1

Рис. 4.5. Логические функции одной переменной

Очевидно,

$$y_0 = f(x) = 0;$$

$$y_1 = f(x) = \bar{x};$$

$$y_2 = f(x) = x;$$

$$y_3 = f(x) = 1.$$

Функции y_0 и y_3 называются **функция-константа 0** и **функция-константа 1** соответственно. Функция y_1 — это **логическая функция НЕ**, или **отрицание**, а функция y_2 называется **функцией повторения**.

Для $n = 2$ существует

$$m = 2^{2^2} = 2^4 = 16$$

логических функций, представленных на рис. 4.6.

$x_1 x_2$	y_0	y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8	y_9	y_{10}	y_{11}	y_{12}	y_{13}	y_{14}	y_{15}
0 0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
0 1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
0 1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1 1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

Рис. 4.6. Логические функции двух переменных

Функции y_0 и y_{15} нам уже знакомы, константа 0 и константа 1 соответственно:

$$y_0 = f(x_1, x_2) = 0;$$

$$y_{15} = f(x_1, x_2) = 1.$$

Функция y_8 может быть записана в виде:

$$y_8 = f(x_1, x_2) = x_1 x_2.$$

Очевидно, что y_8 имеет название **логическая функция И**, или **конъюнкция**.

Функция y_{14} может быть записана в виде:

$$y_{14} = f(x_1, x_2) = x_1 \vee x_2.$$

Следовательно, функция y_{14} называется **логической функцией ИЛИ** либо **дизъюнкцией**.

Логические функции НЕ, И, ИЛИ, введенные с помощью элементарных операторов $\bar{}$, $\&$, (соответственно называются элементарными логическими функциями.

Из рис. 4.6 следует, что

$$y_1 = \overline{x_1 \vee x_2}.$$

Рассматриваемая функция называется **логической функцией ИЛИ-НЕ**.

Аналогичным образом функция y_7 может быть представлена в виде

$$y_7 = f(x_1, x_2) = \overline{x_1 x_2}.$$

Рассматриваемая функция называется **логической функцией И-НЕ**.

Функция

$$y_9 = f(x_1, x_2) = \overline{x_1} \overline{x_2} \vee x_1 x_2$$

принимает значение 1 только тогда, когда $x_1 = x_2 = 0$ или $x_1 = x_2 = 1$. Эта функция называется **логической функцией СОВПАДЕНИЕ**, или **ЭКВИВАЛЕНТ-НОСТЬ** и обозначается с помощью символа “ \equiv ”.

Анализируя таблицу на рис. 4.6, также заметим, что

$$\begin{aligned}y_3 &= f(x_1, x_2) = \bar{x}_1 \text{ (отрицание переменной } x_1); \\y_{12} &= f(x_1, x_2) = x_1 \text{ (повторение переменной } x_1); \\y_5 &= f(x_1, x_2) = \bar{x}_2 \text{ (отрицание переменной } x_2); \\y_{10} &= f(x_1, x_2) = x_2 \text{ (повторение переменной } x_2).\end{aligned}$$

Для лучшего запоминания на рис. 4.7 представлены таблицы истинности логических функций НЕ, И, ИЛИ, И-НЕ, ИЛИ-НЕ и СОВПАДЕНИЕ.

НЕ		И		ИЛИ	
x	\bar{x}	$x_1 \ x_2$	$x_1 x_2$	$x_1 \ x_2$	$x_1 \vee x_2$
0	1	0 0	0	0 0	0
1	0	0 1	0	0 1	1
		1 0	0	1 0	1
		1 1	1	1 1	1

И-НЕ		ИЛИ-НЕ		СОВПАДЕНИЕ	
$x_1 \ x_2$	$\overline{x_1 x_2}$	$x_1 \ x_2$	$\overline{x_1 \vee x_2}$	$x_1 \ x_2$	$x_1 \vee x_2$
0 0	1	0 0	1	0 0	1
0 1	1	0 1	0	0 1	0
1 0	1	1 0	0	1 0	0
1 1	0	1 1	0	1 1	1

Рис. 4.7. Часто используемые логические функции

Аналогично можно рассматривать логические функции от трех переменных, число которых составляет $m = 2^{2^3} = 2^8 = 256$; логические функции четырех переменных, число которых $m = 2^{2^4} = 2^{16} = 65\ 536$ и т. д. Заметим, что, будучи конечным, число булевых функций все равно огромно. Однако доказано, что **любая логическая функция n переменных, $n \geq 2$, может быть выражена с помощью формулы, содержащей только элементарные операторы $\bar{}$, $\&$, ()** . Указанное свойство упрощает техническую реализацию устройств, предназначенных для вычисления логических функций произвольного числа аргументов.

Вопросы и упражнения

- 1 Определите число логических функций пяти и шести переменных.
- 2 Назовите элементарные логические функции и составьте соответствующие им таблицы истинности.
- 3 Запомните таблицы истинности часто используемых логических функций НЕ, И, ИЛИ, И-НЕ, ИЛИ-НЕ и СОВПАДЕНИЕ.
- 4 Напишите программу, которая выводит на экран таблицу истинности одной из логических функций y_0, y_1, y_2, y_3 двух переменных.
- 5 Напишите программу, которая выводит на экран таблицу истинности одной из логической функций y_j от n переменных.

5.1. Логические элементы

Логическая схема — это устройство, предназначенное для вычисления логических функций. Для реализации логических схем необходимо, чтобы двоичные значения **0, 1** аргументов и соответствующих логических функций были представлены значениями определенных физических величин, например давления, температуры, напряжения или силы электрического тока, освещенности и т. п. В зависимости от используемых величин различаем следующие логические устройства: механические, пневматические, гидравлические, электромеханические, электронные, оптические и т. п. В гидравлических и пневматических устройствах логические значения **0, 1** могут представляться малыми и соответственно большими значениями давления жидкости или газа, в электромеханических и электронных устройствах — присутствием или отсутствием электрического тока, с помощью уровней напряжения и т. д.

Для более ясного понимания принципов работы логических устройств рассмотрим сначала контактные схемы. Базовыми элементами этих схем являются **элементы коммутации** — нормально разомкнутые и нормально замкнутые электрические контакты.

В случае **нормально разомкнутых контактов** электрическая цепь разомкнута, если контакты выключены, и замкнута, если они включены. В случае **нормально замкнутых контактов** электрическая цепь замкнута, если контакты выключены, и разомкнута в противном случае (рис. 5.1).





	Выключены	Включены
Нормально разомкнутые контакты		
Нормально замкнутые контакты		

Рис. 5.1. Нормально разомкнутые и нормально замкнутые контакты

Например, контакты дверной кнопки электрического звонка являются нормально разомкнутыми, а контакты кнопки *Пауза* обычного магнитофона — нормально замкнутыми.

В контактных схемах логические значения аргументов представлены состояниями соответствующих электрических контактов. Логическому значению **1** соответствует состояние *включен*, а логическому значению **0** соответствует состояние *выключен*.

Электрическая схема, реализующая логическую функцию **НЕ** и используемое обозначение, представлена на рис. 5.2.

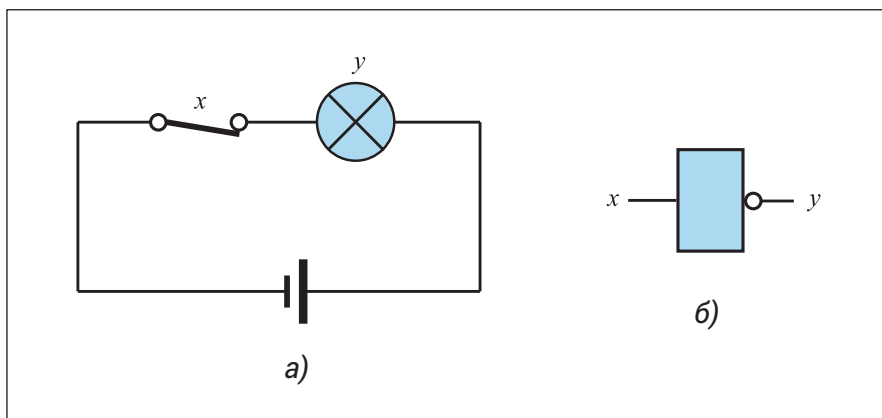


Рис. 5.2. Схема с контактами для реализации логической функции **НЕ** (а) и используемое обозначение (б)

Аргумент x представлен с помощью нормально замкнутого контакта, а значения зависимой переменной y — с помощью состояний электрической лампы: *погашена* (логическое значение **0**) или *горит* (логическое значение **1**). Заметим, что лампа будет гореть ($y = 1$), если нормально замкнутый контакт не включен ($x = 0$).

Логическая функция И реализуется последовательным соединением электрических контактов. Электрическая схема, реализующая функцию **И** двух переменных, и используемое обозначение представлены на рис. 5.3.

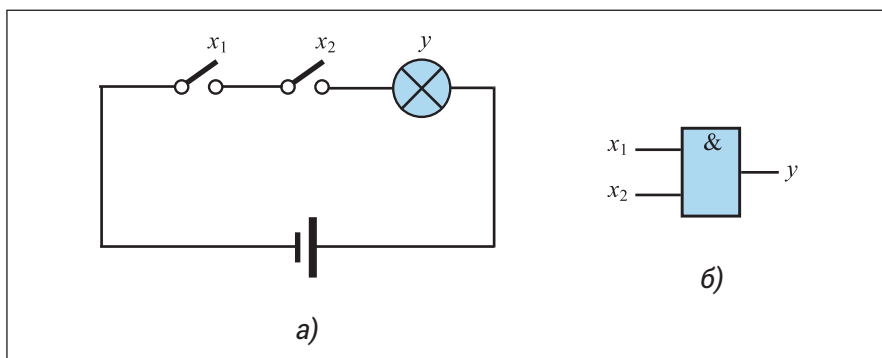


Рис. 5.3. Схема с контактами для реализации логической функции **И** (а) и используемое обозначение (б)

Переменные x_1 и x_2 представлены с помощью двух нормально разомкнутых контактов, а значение переменной y — с помощью лампы. Заметим, что лампа будет гореть ($y = 1$) только тогда, когда оба нормально разомкнутых контакта одновременно включены ($x_1 = 1$ и $x_2 = 1$).

Логическая функция ИЛИ реализуется параллельным соединением электрических контактов. Соответствующая схема представлена на рис. 5.4.

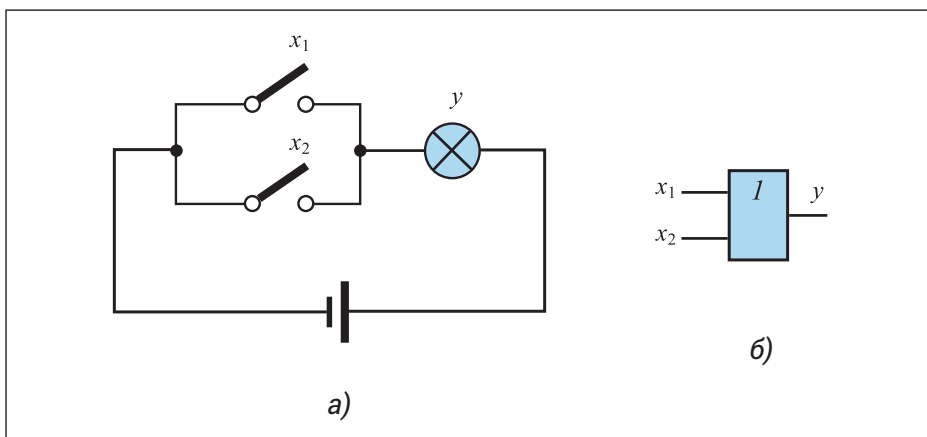


Рис. 5.4. Схема с контактами для реализации логической функции ИЛИ (а) и используемое обозначение (б)

Отметим, что лампа будет гореть ($y = 1$), если хотя бы один из нормально разомкнутых контактов будет включен ($x_1 = 1$ или $x_2 = 1$).

Поскольку скорость замыкания-размыкания электрических контактов очень мала, то в современных компьютерах значения **0**, **1** представлены уровнями напряжения, а в качестве коммутирующего элемента используется транзистор.

Транзистор — это электронный прибор, изготовленный внутри или на поверхности полупроводникового кристалла. Современные технологии позволяют получать $10^6 - 10^7$ транзисторов на 1 см^2 площади кристалла.

В режиме коммутации транзистор можно рассматривать как обычный выключатель, который в одном состоянии проводит электрический ток (контакты замкнуты), а в другом — нет (контакты разомкнуты). Однако в отличие от обычных выключателей открытие и закрытие транзистора осуществляются с помощью электрического тока.

Существуют различные типы транзисторов. На рис. 5.5 представлен транзистор *n-p-n* (сокращение относится к внутренней структуре транзистора) и эквивалентные схемы, иллюстрирующие его работу в режиме коммутации.

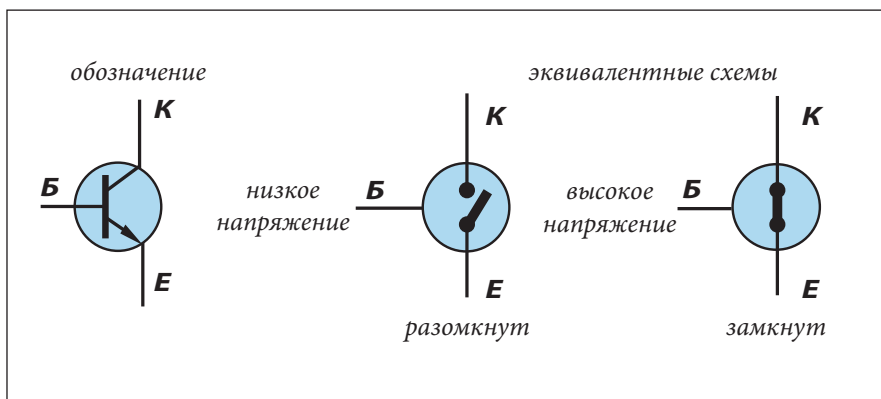


Рис. 5.5. Транзистор *n-p-n*

Транзистор *n-p-n* имеет три вывода: эмиттер Э, база Б и коллектор К. В режиме коммутации эмиттер и коллектор могут рассматриваться как контакты, замыкающиеся и размыкающиеся с помощью напряжения, поданного на базу. Отметим, что современные транзисторы позволяют осуществлять до $10^6 - 10^9$ переключений в секунду. Как и в случае электрических контактов, рассмотренных выше, использование различных типов транзисторов и их последовательное или параллельное соединение позволяют реализовать логические функции *НЕ*, *И*, *ИЛИ*.

Логические схемы, предназначенные для вычисления часто используемых логических функций, называются элементарными логическими схемами, или логическими элементами.

Обозначения логических элементов приведены на рис. 5.6.

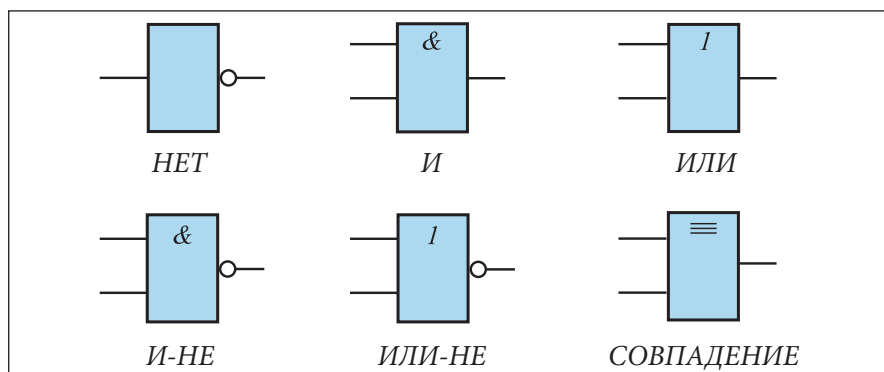


Рис. 5.6. Обозначения логических элементов

Известно, что любая логическая функция может быть выражена с помощью формулы, включающей только элементарные операторы $\bar{}$, $\&$, \vee . Следовательно, любая **логическая функция произвольного количества аргументов может быть реализована путем соединения логических элементов НЕ, И, ИЛИ.**

Например, функция

$$y = x_1 x_2 \vee \bar{x}_2 x_3$$

может быть реализована с помощью следующих логических элементов:

- элемента *НЕ* для вычисления x_2 ;
- двух логических элементов *И* для вычисления конъюнкций $x_1 x_2$ и $\bar{x}_2 x_3$;
- элемента *ИЛИ* для вычисления дизъюнкции $x_1 x_2 (\bar{x}_2 x_3)$.

Схема логической цепи для вычисления рассматриваемой функции представлена на рис. 5.7.

Вопросы и упражнения

- ❶ Как можно представить двоичные значения 0 и 1?
- ❷ Как работают нормально разомкнутые и нормально замкнутые контакты?
- ❸ Какова роль транзистора в современных компьютерах?
- ❹ ЭКСПЕРИМЕНТИРУЙТЕ! Используя оборудование из физической лабо-

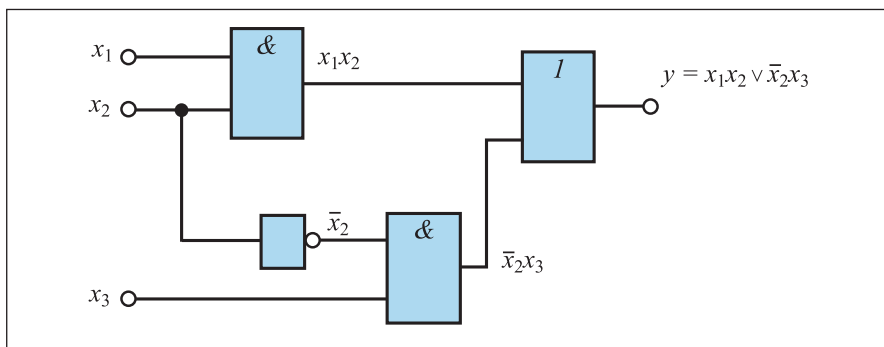


Рис. 5.7. Логическая схема для реализации функции $y = x_1x_2 \vee \bar{x}_2x_3$

ратории, соберите схемы, изображенные на рис. 5.2, 5.3 и 5.4. Проверьте таблицы истинности функций, реализованных данными схемами.

- 5 Как с помощью контактных схем реализуются функции **НЕ**, **И**, **ИЛИ**?
- 6 Какова роль элементов коммутации при реализации логических функций?
- 7 Какова роль транзистора в современных компьютерах?
- 8 Запомните обозначения логических элементов. Объясните, как используются логические элементы для реализации произвольных логических функций.
- 9 **ПРОЕКТИРУЙТЕ!** Используя элементы **НЕ**, **И**, **ИЛИ**, составьте логические схемы для вычисления следующих функций:

- | | |
|---|--|
| a) $y = x_1x_2x_3;$ | i) $y = x_1x_2 \vee \bar{x}_2x_3;$ |
| b) $y = x_1 \vee x_2 \vee x_3;$ | j) $y = (x_1 \vee x_2) (\bar{x}_2 \vee \bar{x}_3);$ |
| c) $y = \bar{x}_1x_2x_3;$ | k) $y = x_1x_2 \vee \bar{x}_1x_3 \vee x_3x_4;$ |
| d) $y = \bar{x}_1 \vee x_2 \vee x_3;$ | l) $y = \bar{x}_1x_2 \vee \bar{x}_1x_3 \vee x_2x_3;$ |
| e) $y = x_1x_2 \vee x_3x_4;$ | m) $y = \bar{x}_1x_2 \vee x_1\bar{x}_2;$ |
| f) $y = (x_1 \vee x_2) (x_3 \vee x_4);$ | n) $y = x_1x_2 \vee \bar{x}_1\bar{x}_2;$ |
| g) $y = x_1x_2;$ | o) $y = x_1(x_2 \vee x_3 \vee x_4);$ |
| h) $y = x_1 \vee x_2;$ | p) $y = x_1 \vee x_2x_3x_4.$ |

- 10 **ИССЛЕДУЙТЕ!** Электромагнитное реле — это устройство, с помощью которого осуществляется замыкание и размыкание электрических контактов. Соответствующие контакты включаются электромагнитом. Как можно использовать реле для реализации логических функций **НЕ**, **И**, **ИЛИ**? **ЭКСПЕРИМЕНТИРУЙТЕ!** Из деталей школьного физического набора соберите электромагнитное реле и проверьте работу схем, предложенных вами.
- 11 **ИССЛЕДУЙТЕ!** Представляя двоичные значения переменных на выходе наличием (значение 1) или отсутствием (значение 0) жидкости, разработайте гидравлическую схему для реализации логических функций **НЕ**, **И**, **ИЛИ**. Соберите соответствующие схемы, используя краны и принадлежности школьного химического набора. Проверьте таблицы истинности реализованных логических функций.

5.2. Классификация логических схем

Логические схемы классифицируются на комбинационные и последовательностные.

В **комбинационной** схеме значения переменных на выходе определяются только текущими значениями переменных на входе в соответствии с логическими функциями схемы.

В **последовательностной** схеме значения переменных на выходе зависят не только от значений входных переменных в текущий момент времени, но и от последовательности их подачи.

Другими словами, комбинационные схемы представляют собой логические устройства без элементов памяти, в то время как последовательностные схемы содержат элементы двоичной памяти. Следовательно, комбинационная схема выполняет числовую обработку информации, которую в общем случае можно выразить с помощью логических функций, не содержащих временных параметров.

Обозначение комбинационных схем приведено на рис. 5.8.

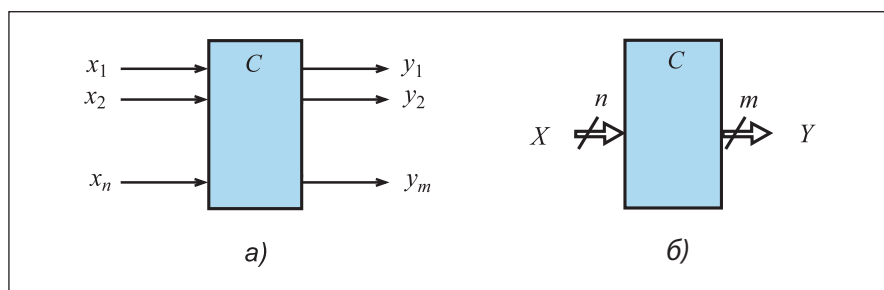


Рис. 5.8. Обозначение комбинационных схем: а — детальное; б — обобщенное

Комбинационная схема имеет n входных переменных $X = \langle x_1, x_2, \dots, x_n \rangle$ и m выходных переменных: $Y = \langle y_1, y_2, \dots, y_m \rangle$. Соединения, осуществляющие передачу (на вход или с выхода) значений группы переменных, обозначаются с помощью двойной линии. В случае необходимости количество переменных указывается возле наклонной черты, пересекающей двойную линию группы переменных. Например, на рис. 5.8, б показано, что группа X содержит n переменных, а группа Y — m переменных.

С точки зрения теории информации комбинационная схема представляет собой преобразователь кодов: на вход подаются двоичные слова исходного кода, а на выходе появляются соответствующие двоичные слова из кода, в который осуществляется преобразование.

Например, исходным кодом может быть *EBCDIC*, а результирующим кодом — *ASCII*.

5.3. Сумматор

При обработке информации одна из главных задач любого компьютера состоит в выполнении арифметических операций и в особенности сложения и вычитания. Устройства, осуществляющие указанные операции, основываются на схемах, выполняющих сложение и вычитание двух двоичных цифр.

Полусумматор — это комбинационная схема, предназначенная для сложения двух двоичных цифр. Таблица истинности, поясняющая работу полусумматора, основывается на правиле сложения двух двоичных цифр и представлена на *рис. 5.9*. Здесь a и b представляют две суммируемые двоичные цифры, s — цифру суммы соответствующего разряда, а t — цифру переноса в следующий разряд.

a	b	s	t
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Рис. 5.9. Таблица истинности для сложения двух двоичных цифр

Для разработки одной из возможных схем полусумматора выразим функции выходных переменных s и t :

$$s = \bar{a}b \vee a\bar{b};$$

$$t = ab.$$

Схема, реализующая функции s , t , и используемое обозначение представлены на *рис. 5.10*.

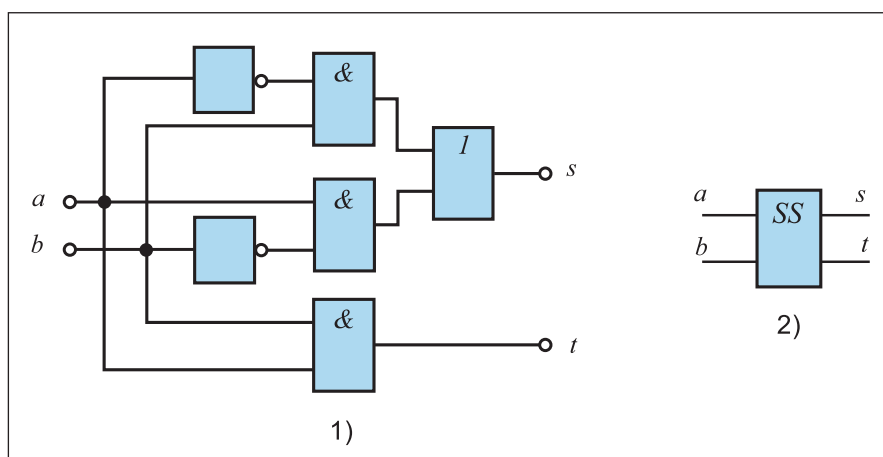


Рис. 5.10. Схема полусумматора (1) и используемое обозначение (2)

Рассмотрим два двоичных числа

$$A = a_{n-1} a_{n-2} \dots a_j \dots a_0$$

и

$$B = b_{n-1} b_{n-2} \dots b_j \dots b_0,$$

где a_j и b_j представляют двоичные цифры разряда j . При суммировании цифр a_j и b_j разряда j нужно учитывать и цифру переноса a_{j-1} из разряда $j-1$:

$$\begin{array}{r}
 t_{j-1} \\
 a_{n-1} a_{n-2} \dots a_j \dots a_0 \\
 + \\
 b_{n-1} b_{n-2} \dots b_j \dots b_0
 \end{array}$$

Таким образом, получаем комбинационную схему, вычисляющую сумму $t_{j-1} + a_j + b_j$, называемую **элементарным сумматором**.

Элементарный сумматор может быть реализован путем каскадного соединения двух полусумматоров SS_1 и SS_2 (рис. 5.11).

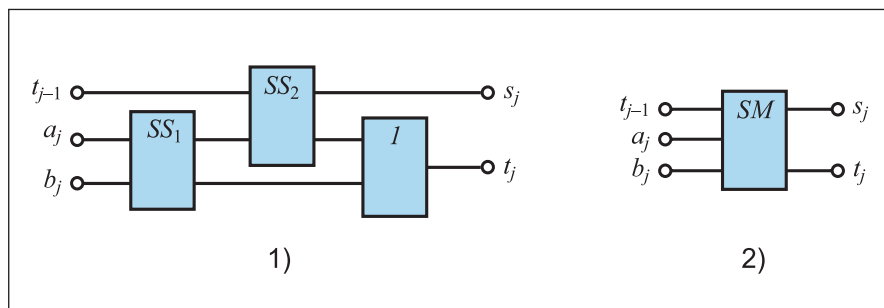


Рис. 5.11. Схема элементарного сумматора (1) и используемое обозначение (2)

Полусумматор SS_1 вычисляет сумму $(a_j + b_j)$, а полусумматор SS_2 суммирует перенос t_{j-1} с суммой $(a_j + b_j)$, вычисленной первым полусумматором. Перенос t_j в старший разряд $j+1$ вычисляется логическим элементом **ИЛИ**, который объединяет промежуточные переносы полусумматоров SS_1 и SS_2 .

Сумма двоичных чисел A и B вычисляется с помощью комбинационной схемы, называемой **сумматором**. Сумматор может быть реализован путем каскадного соединения n элементарных сумматоров (рис. 5.12).

Элементарный сумматор SM_0 , соответствующий самой младшей значащей цифре, может быть заменен полусумматором, поскольку для нулевого разряда перенос из предыдущего разряда отсутствует. Перенос, образующийся на выходе элементарного сумматора SM_{n-1} самого старшего значащего разряда, используется для указания переполнения емкости сумматора на n бит.

Из анализа рис. 5.10, 5.11 и 5.12 следует, что сложное устройство – сумматор на n бит – получен соединением более простых устройств, то есть n элементарных сумматоров. Каждый элементарный сумматор, в свою очередь, получен с использованием двух полусумматоров и логического элемента **ИЛИ**.

Метод разработки сложных устройств (например, сумматора) путем соединения необходимого числа более простых одинаковых устройств (например, элементарных сумматоров) называется **методом иерархического проектирования**. В соответствии с данным методом все компоненты компьютера принадлежат определенным **уровням иерархии**, например:

- уровень 1 — транзисторы;
- уровень 2 — логические элементы;
- уровень 3 — полусумматоры, элементарные сумматоры и т. д.;
- уровень 4 — сумматоры, вычитатели и т. д.;
- уровень 5 — арифметические устройства, устройства управления и т. д.

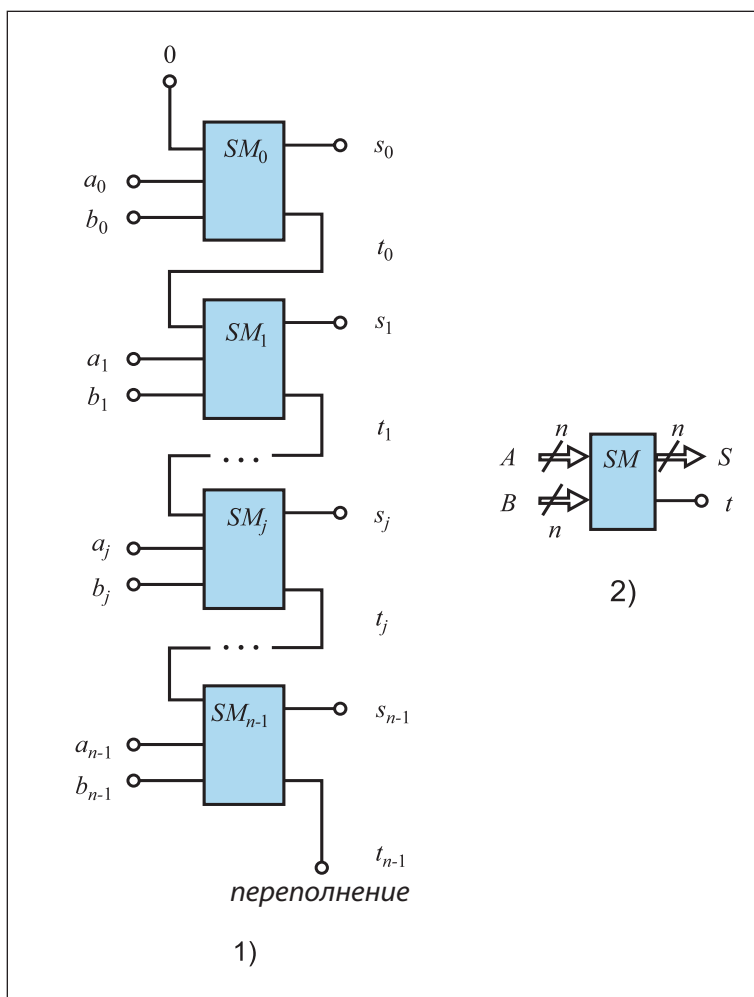


Рис. 5.12. Схема сумматора (1) и используемое обозначение (2)

Компоненты нижнего иерархического уровня используются в качестве простейших «кубиков» для построения компонентов верхнего уровня.

Теоретически устройства любого цифрового компьютера могут быть разработаны и без иерархического метода проектирования. Например, в случае элементарного сумматора использование полусумматоров не обязательно. Достаточно составить таблицу истинности элементарного сумматора, представить выходные функции с помощью формул и использовать соответствующие логические элементы.

Для компонентов более высокого уровня попытка обойтись без метода иерархического проектирования делает невозможной разработку сложных устройств. Например, в случае n -битового сумматора соответствующая таблица истинности состоит из 2^{2n} строк. Для $n = 16$ получаем $2^{2 \cdot 16} = 2^{32} \approx 10^9$ строк. Очевидно, что формулы выходных функций сумматора на 16 битов практически не могут быть записаны. Следовательно, мы вынуждены использовать метод иерархического проектирования и реализовать сумматор путем соединения n элементарных сумматоров.

Аналогичным образом, используя метод иерархического проектирования, можно разработать комбинационные схемы, предназначенные для вычитания двоичных чисел: **полувывчитатель**, **элементарный вычитатель** и **вычитатель**.

Вопросы и упражнения

- 1 Укажите назначение полусумматора; элементарного сумматора; сумматора на n битов.
- 2 ТВОРИТЕ! Составьте таблицу истинности элементарного сумматора. Таблица должна содержать пять столбцов: три для входов a_j, b_j, t_{j-1} и два для выходов s_j, t_j .
- 3 Напишите программу, которая моделирует работу элементарного сумматора. Двоичные цифры a_j, b_j и цифра переноса t_{j-1} от младшего разряда вводятся с клавиатуры, а цифра суммы s_j и цифра переноса к старшему разряду — t_j должны быть выведены на экран.
- 4 Объясните суть метода иерархического проектирования. Обязательно ли применение данного метода? Аргументируйте ваш ответ.
- 5 Сколько логических элементов **НЕ**, **И**, **ИЛИ** содержит сумматор на 16 битов? А на 32 бита?
- 6 ПРОЕКТИРУЙТЕ! **Полувывчитатель** — это комбинационная схема, предназначенная для вычитания двух двоичных цифр. Соответствующая схема имеет входы a, b и выходы d, i . С помощью d обозначена разность $a - b$, а с помощью i — заем из старшего ближайшего разряда. Составьте таблицу истинности и разработайте схему полувывчитателя.
- 7 ПРОЕКТИРУЙТЕ! **Элементарный вычитатель** — это комбинационная схема, способная вычислять разность d_j и заем i_j из ближайшего старшего разряда, если на вход подаются уменьшаемое a_j , вычитаемое b_j и заем i_{j-1} из предыдущего разряда. Используя метод иерархического проектирования, разработайте схему элементарного вычитателя.
- 8 ПРОЕКТИРУЙТЕ! Используя метод иерархического проектирования, разработайте схему вычитателя на n битов.
- 9 Сколько логических элементов **НЕ**, **И**, **ИЛИ** содержит вычитатель на 16 битов? А на 32 бита?
- 10 ПРОАНАЛИЗИРУЙТЕ! Обязательно ли применение метода иерархического проектирования при разработке вычитателя на n битов? Аргументируйте ваш ответ.

5.4. Часто используемые комбинационные схемы

Часто используемые комбинационные схемы представлены на *рис. 5.13*.

Сумматор — это комбинационная схема, предназначенная для сложения двух двоичных чисел. Таблица истинности и схема сумматора были рассмотрены в предыдущем параграфе.

Компаратор — это комбинационная схема, которая сравнивает два двоичных числа A и B , указывая на трех выходах одну из возможных ситуаций: $A < B$, $A > B$ и $A = B$.

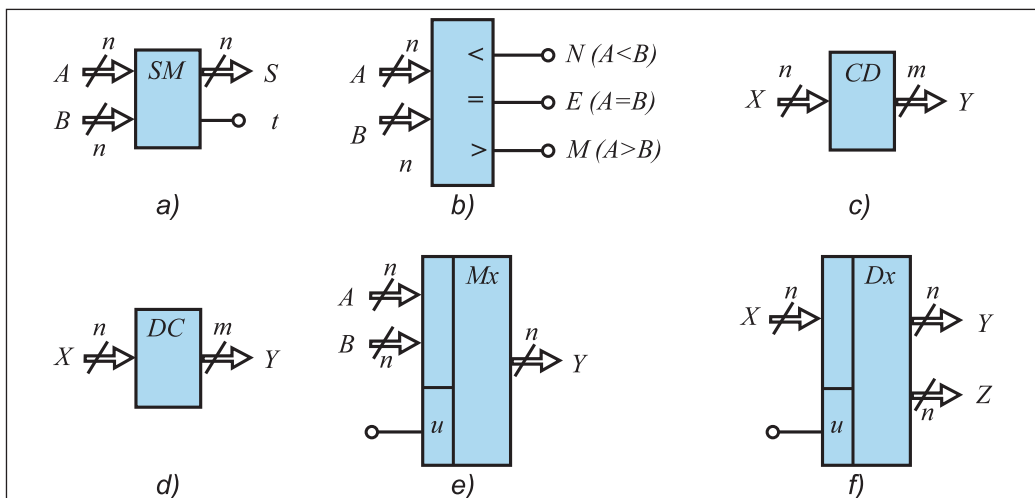


Рис. 5.13. Часто используемые комбинационные схемы:
 а — сумматор; б — компаратор; в — шифратор; г — дешифратор;
 е — мультиплексор; ф — демультиплексор

Шифратор — это комбинационная схема, которая выполняет преобразование сообщений s_1, s_2, \dots, s_n в двоичные слова в соответствующем коде. Каждое сообщение s_i представлено значениями $x_1 = 0, \dots, x_i = 1, \dots, x_n = 0$, поданными на вход шифратора, а закодированное (зашифрованное) слово — с помощью выходных переменных y_1, y_2, \dots, y_m .

Например, переменные x_1, x_2, x_3, \dots могут представлять состояния клавиш $\langle A \rangle, \langle B \rangle, \langle C \rangle, \dots$ клавиатуры. Соответствующий шифратор обеспечит на выходе слово в коде ASCII для нажатой клавиши.

Дешифратор — это комбинационная схема, которая генерирует логический сигнал 1 на выходе, различном для каждой комбинации входных переменных. Другими словами, дешифратор осуществляет операцию, обратную той, которую выполняет шифратор.

Дешифраторы используются для определения операций, которые должен выполнять процессор, для определения состояния устройств ввода-вывода, для синтеза символов и т. п.

Мультиплексор — это комбинационная схема, предназначенная для выбора потока данных. На рис. 5.13, е представлен мультиплексор, который передает на выход биты двоичного числа A ($u = 0$) или B ($u = 1$). В современных компьютерах мультиплексоры используются для передачи данных от нескольких источников к одному приемнику.

Демультиплексор распределяет поток данных с входа X на один из выходов Y ($u = 0$) или Z ($u = 1$). В качестве примера вспомним передачу информации от одного источника к нескольким приемникам.

Вопросы и упражнения

- 1 Объясните назначение часто используемых комбинационных схем: сумматора, компаратора, шифратора, дешифратора, мультиплексора и демультиплексора.
- 2 Составьте таблицу истинности компаратора на 2 бита.

③ Сколько входов и сколько выходов может быть у шифратора? Сколько входов и сколько выходов может быть у дешифратора?

④ ТВОРИТЕ! На панели управления принтера установлены кнопки *ON LINE* (работа под управлением центрального устройства), *OFF LINE* (автономная работа), *LINE FEED* (продвинуть на строку) и *FORM FEED* (продвинуть на страницу). Составьте таблицу истинности шифратора, который вырабатывает на выходе следующие двоичные комбинации:

00 – *ON LINE*;

01 – *OFF LINE*;

10 – *LINE FEED*;

11 – *FORM FEED*.

⑤ ТВОРИТЕ! На панели управления принтера установлены светодиодные индикаторы: *READY* (готов), *PAPER* (отсутствие бумаги), *TEST* (режим тестирования) и *LOAD* (режим загрузки информации). Составьте таблицу истинности дешифратора, который управляет светодиодами. Соответствующие режимы закодированы с помощью следующих двоичных комбинаций:

00 – *READY*;

01 – *PAPER*;

10 – *TEST*;

11 – *LOAD*.

⑥ ТВОРИТЕ! Клавиатура компьютера содержит функциональные клавиши *<F1>*, *<F2>*, ..., *<F12>*. Составьте таблицу истинности шифратора, который выдает на выходе двоичное число, соответствующее нажатой клавише.

⑦ ТВОРИТЕ! Устройства ввода-вывода учебного компьютера имеют следующие адреса:

0000 – клавиатура;

0001 – монитор;

0010 – механический принтер;

0011 – струйный принтер;

0100 – лазерный принтер;

0101 – дисковод для гибких дисков *A*;

0110 – дисковод для гибких дисков *B*;

0111 – привод жесткого диска *C*;

1000 – привод жесткого диска *D*.

Составьте таблицу истинности дешифратора, который выбирает устройство, указываемое соответствующим адресом.

⑧ ТВОРИТЕ! Арифметические операции учебного компьютера закодированы следующим образом:

Сложение	– 000;
Вычитание	– 001;
Умножение	– 010;
Деление	– 011;
Сравнение	– 100.

Составьте таблицу истинности дешифратора арифметических операций рассматриваемого компьютера.

- 9 ТВОРИТЕ! Составьте таблицу истинности мультиплексора с двумя входными линиями.

5.5. RS-триггер

Известно, что в последовательностных схемах значения выходных переменных зависят не только от комбинаций входных переменных, но и от **последовательности их подачи**. Другими словами, последовательностная схема запоминает информацию о двоичных комбинациях, поданных на входы схемы в предыдущие моменты времени. Такое возможно благодаря тому, что последовательностные схемы состоят из комбинационных схем и элементов двоичной памяти.

Элемент двоичной памяти — это схема с двумя различными состояниями, предназначенная для хранения одного бита информации. Соответствующая схема называется триггером.

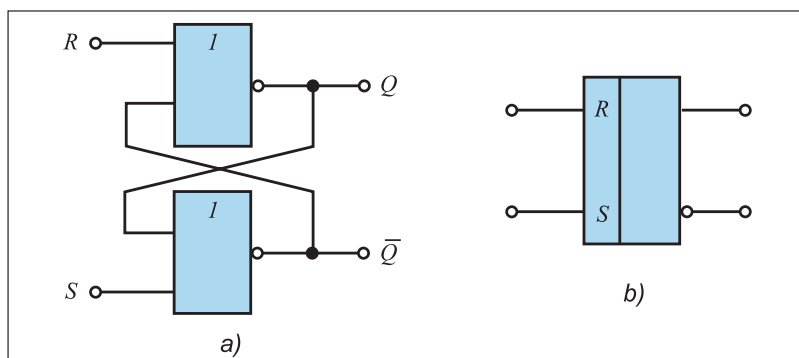


Рис. 5.14. Схема асинхронного RS-триггера (a) и используемое обозначение (b)

На рис. 5.14 представлена схема простейшего триггера, выполненного на логических элементах **ИЛИ-НЕ**, называемого **асинхронным RS-триггером**. У схемы есть два входа, обозначаемые R и S , и два выхода, обозначаемые Q и \bar{Q} . Заметим, что выходные сигналы Q и \bar{Q} подаются на вторые входы элементов **ИЛИ-НЕ**. Соответствующие соединения называются **обратными связями**. Именно благодаря этим соединениям данная схема обладает двумя различными состояниями и, следовательно, обеспечивает запоминание одного бита информации.

В самом деле, пусть входные сигналы $R = S = 0$, а выходы $Q = 1$, $\overline{Q} = 0$. Из-за обратной связи значение $Q = 1$ заставляет другой выход принять значение $\overline{Q} = 0$. В свою очередь, благодаря обратной связи, значение $\overline{Q} = 0$ подтверждает выходной сигнал $Q = 1$. Точно так же в случае, когда выходы $Q = 0$, $\overline{Q} = 1$, из-за обратной связи значение $\overline{Q} = 1$ удерживает другой выход в состоянии $Q = 0$. Следовательно, входные значения $R = S = 0$ не изменяют состояние триггера, обеспечивая хранение ранее записанной двоичной цифры.

Удобно, чтобы состоянию $Q = 1$, $\overline{Q} = 0$ соответствовала двоичная цифра 1, а состоянию $Q = 0$, $\overline{Q} = 1$ — двоичная цифра 0. Выход Q называется **прямым**, или **истинным выходом**, а выход \overline{Q} — **инверсным выходом**, или **выходом отрицания**. Состояние триггера указывается значением прямого выхода Q .

Рассмотрим случай, когда $R = 0$, $S = 1$. Предположим, что триггер находится в состоянии 0, то есть $Q = 0$ и $\overline{Q} = 1$. В этом случае значение $S = 1$ переведет $\overline{Q} = 0$, который в свою очередь, через обратную связь, обеспечит $Q = 1$. Следовательно, триггер переходит в состояние 1 ($Q = 1$, $\overline{Q} = 0$). Это состояние, как установлено ранее, сохранится и после того, как сигнал S изменит свое значение с 1 на 0. Вход S , который обеспечивает установку триггера в состояние 1, называется **входом установки**.

Аналогично можно установить, что в случае, когда триггер находится в состоянии 1 ($Q = 1$, $\overline{Q} = 0$), а $S = 0$, и $R = 1$, триггер перейдет в состояние 0 ($Q = 0$, $\overline{Q} = 1$). Это состояние сохранится и после перехода сигнала R от значения 1 к значению 0. Вход R , который обеспечивает установку триггера в состояние 0, носит название входа сброса.

Входная комбинация $R = 1$ и $S = 1$ заставляет выходы перейти в состояния $Q = 0$ и $\overline{Q} = 0$. Значения $Q = \overline{Q} = 0$ не соответствуют условию противоположности сигналов на выходах. Следовательно, для триггера RS входная комбинация $R = S = 1$ является запрещенной.

Режимы работы рассматриваемого триггера приведены в *таблице 5.1*.

Таблица 5.1

Режимы работы асинхронного RS-триггера			
Входы R S		Режим работы	Выход Q
0	0	хранение	хранимый бит
0	1	установка	1
1	0	сброс	0
1	1	запрещено	—

Прилагательное *асинхронный* в названии RS -триггера указывает на характер влияния управляющих сигналов R и S на состояние триггера. После изучения схемы, приведенной на *рис. 5.14*, приходим к выводу, что управляющие сигналы, поданные на входы R и S , могут изменять состояние триггера в произвольные моменты времени.

Последовательностные схемы, состояние которых может быть изменено управляющими сигналами в произвольные моменты времени, называются асинхронными схемами.

Современный компьютер содержит десятки тысяч триггеров. Изменение их состояний в произвольные моменты времени трудно контролировать, и это может привести к ошибкам в работе. Для исключения ошибок необходимо, чтобы поведение последовательностных схем контролировалось значениями управляющих сигналов, подаваемых на входы в точно определенные дискретные моменты времени. Эти моменты времени задаются с помощью специальных импульсов, называемых **сигналами синхронизации**.

Последовательностные схемы, состояние которых может быть изменено управляющими сигналами только в моменты времени, определяемые сигналами синхронизации, называются синхронными схемами.

Обычно сигнал синхронизации обозначается с помощью буквы C (от английского *clock* — «часы») и вырабатывается специальным устройством, называемым **системными часами**.

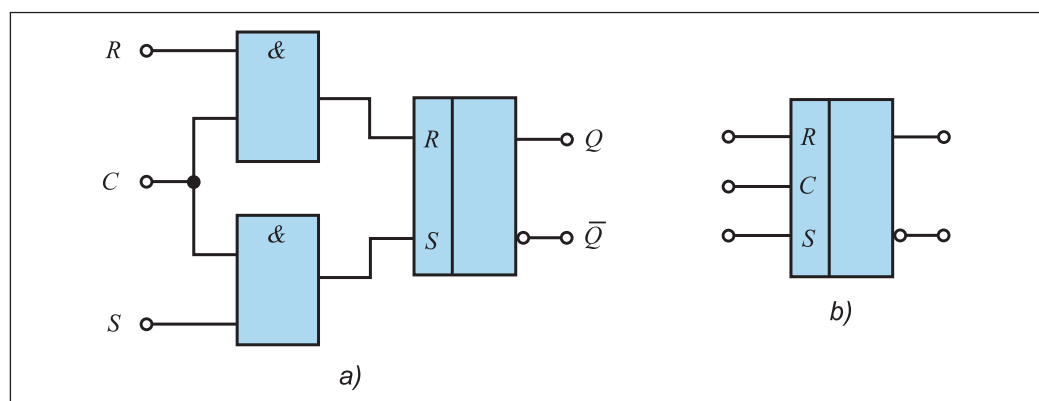


Рис. 5.15. Схема синхронного RS-триггера (a) и его обозначение (b)

На рис. 5.15 представлена схема **синхронного RS-триггера**. Данная схема состоит из асинхронного RS-триггера (рис. 5.14) и двух логических элементов И, которые разрешают подачу управляющих сигналов на входы асинхронного триггера только тогда, когда сигнал синхронизации C принимает значение 1.

Вопросы и упражнения

- 1 Чем отличаются комбинационные и последовательностные схемы?
- 2 В чем назначение триггера?
- 3 Как работает триггер на основе логических элементов **ИЛИ-НЕ**? Для чего предназначены обратные связи?
- 4 Объясните режимы работы асинхронного RS-триггера. Почему комбинация $R = S = 1$ не может быть подана на входы рассматриваемого триггера?
- 5 **ТЕМАТИЧЕСКОЕ ИССЛЕДОВАНИЕ.** Чем отличаются асинхронные и синхронные последовательностные схемы?
- 6 Объясните, как работает синхронный RS-триггер. Каково назначение логических элементов И, входящих в состав данного триггера?

- 7 На рис. 5.16 представлена схема простейшего триггера, реализованного на основе логических элементов *И-НЕ*, называемого **асинхронным $\overline{R}\overline{S}$ -триггером**.

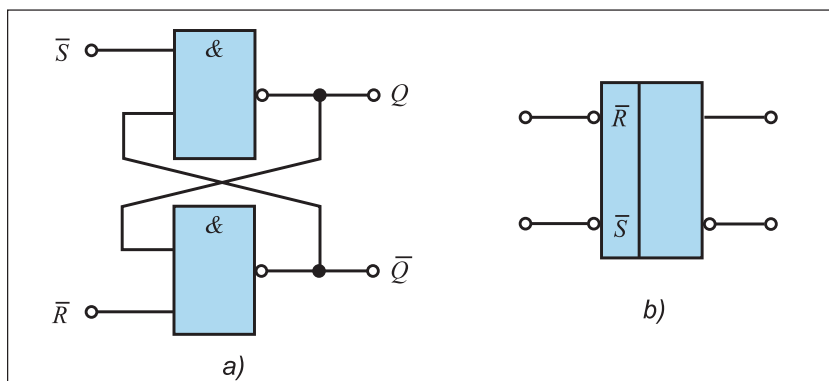


Рис. 5.16. Схема асинхронного $\overline{R}\overline{S}$ -триггера (a) и используемое обозначение (b)

Схема имеет следующие режимы функционирования:

- хранение ($\overline{R} = 1, \overline{S} = 1$);
- установка ($\overline{R} = 1, \overline{S} = 0$);
- сброс ($\overline{R} = 0, \overline{S} = 1$).

Объясните, как работает рассматриваемый триггер. Почему входная комбинация $\overline{R} = 0, \overline{S} = 0$ является запрещенной?

- 8 **ТВОРИТЕ!** Используя асинхронный $\overline{R}\overline{S}$ -триггер, разработайте схему **синхронного $\overline{R}\overline{S}$ -триггера**.
- 9 **ПРОЕКТИРУЙТЕ!** Используя схемы рис. 5.14a и 5.15a, нарисуйте подробную схему (на уровне логических элементов *И*, *ИЛИ-НЕ*) синхронного $\overline{R}\overline{S}$ -триггера.
- 10 **ПРОЕКТИРУЙТЕ!** Нарисуйте подробную схему (на уровне логических элементов *И*, *И-НЕ*) синхронного $\overline{R}\overline{S}$ -триггера.

5.6. Часто используемые последовательные схемы

Регистр (рис. 5.17a) — это цифровое устройство, предназначенное для временного хранения произвольного двоичного числа,

$$D = d_{n-1} \dots d_1 d_0.$$

Регистр состоит из триггеров и комбинационных схем, позволяющих запись, чтение или перенос (сдвиг) информации. Запись информации в регистр осуществляется путем подачи на вход *W* (от англ. *write* — «писать») соответствующего импульса. Так как каждый триггер может запомнить только один бит, то **емкость *n*** регистра определяется числом используемых триггеров.

В некоторых приложениях (например, при умножении и делении двоичных чисел, при записи и чтении данных на диске, при передаче данных по телефонным линиям и т. п.) появляется необходимость сдвига влево или вправо информации, содержащейся в некотором регистре. Для этого используются **сдвигающие регистры** (рис. 5.17b,c).

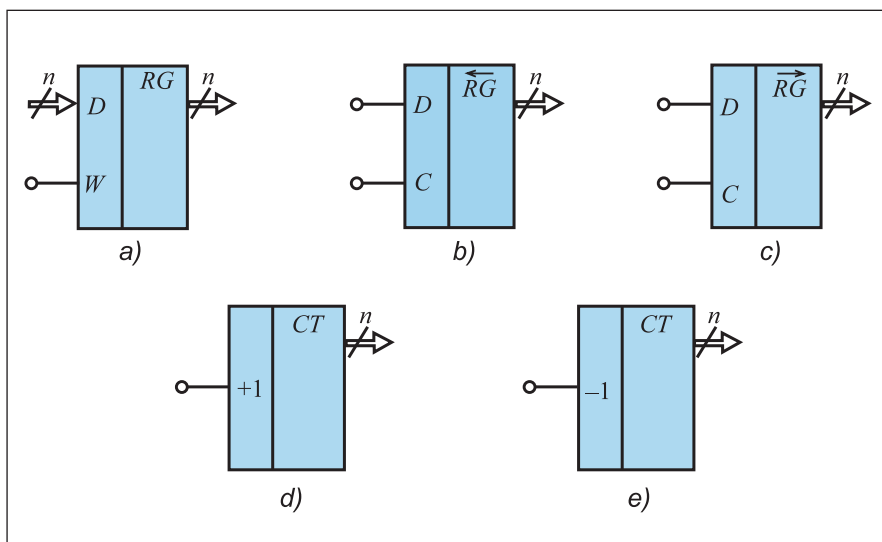


Рис. 5.17. Часто используемые последовательные схемы: а — регистр; б — регистр сдвига влево; с — регистр сдвига вправо; д — прямой счетчик; е — обратный счетчик

Последовательность состояний регистра со сдвигом влево приведена на рис. 5.18.

Моменты времени	d_3	d_2	d_1	d_0
начальный	0	1	0	1
t_1	1	0	1	0
t_2	0	1	0	0
t_3	1	0	0	0
t_4	0	0	0	0
t_5	0	0	0	0
...	...			

Рис. 5.18. Последовательность состояний регистра со сдвигом влево

Предполагается, что начальным состояние регистра является 0101, а импульсы подаются на вход C в момент времени t_1, t_2, t_3 и т. д. Очевидно, что рассматриваемый регистр вычисляет произведение $D \times 2$.

Аналогичным образом **регистр со сдвигом вправо** осуществляет деление $D : 2$.

Счетчик — это последовательная схема, которая подсчитывает количество импульсов, поступивших на ее вход. Счетчики классифицируются по следующим критериям:

– способ кодирования информации на выходе (двоичные, двоично-десятичные и т. д.);

– порядок изменения состояний счетчика (прямые, инверсные и реверсивные счетчики).

В общем случае счетчики создаются путем соединения триггерных схем с комбинационными схемами, определяющими режим изменения состояния счетчиков при поступлении на вход каждого нового импульса.

Двоичный счетчик подсчитывает в двоичной системе счисления количество импульсов, поступивших на его вход. **Емкость двоичного счетчика** зависит от количества входящих в него триггеров. Считая, что двоичным числам соответствуют различные выходные состояния счетчика, получаем в результате диапазон счета от 0 до $2^n - 1$, где n — это количество триггеров.

Моменты времени	d_2	d_1	d_0
начальный	0	0	0
t_1	0	0	1
t_2	0	1	0
t_3	0	1	1
t_4	1	0	0
t_5	1	0	1
t_6	1	1	0
t_7	1	1	1
t_8	0	0	0
t_9	0	0	1
...	...		

Рис. 5.19. Последовательность состояний прямого двоичного счетчика на 3 бита

Счетчики, изменяющие свое состояние в соответствии с таблицей на рис. 5.19, называются **прямыми**, поскольку содержимое счетчика увеличивается на единицу с поступлением на вход «+1» каждого нового импульса. Если же в счетчик предварительно записываются некоторое число и каждый новый импульс, поступивший на вход «-1», уменьшает его содержимое на единицу, то получим **инверсный счетчик** (рис. 5.17е).

Вопросы и упражнения

- ❶ В чем назначение регистра? От чего зависит емкость регистра?
- ❷ ПРОАНАЛИЗИРУЙТЕ! В регистр со сдвигом влево (рис. 5.17b) загружено двоичное число 1001. Каким станет содержимое регистра после подачи на вход С одного импульса? Двух импульсов?
- ❸ В регистр со сдвигом вправо загружено одно из следующих двоичных слов:
а) 00000; д) 00100;

- | | |
|-----------|-----------|
| b) 10000; | e) 00010; |
| c) 01000; | f) 00001; |
| g) 10001; | i) 01100; |
| h) 01010; | j) 00110. |

Каким станет содержимое регистра после поступления на вход C двух последовательных импульсов?

- 4 ПРОГРАММИРУЙТЕ! Напишите программу, которая моделирует работу регистра сдвига слева направо на n битов.
- 5 ТЕМАТИЧЕСКОЕ ИССЛЕДОВАНИЕ. Для чего предназначены счетчики? Как меняются состояния прямого двоичного счетчика? А состояния инверсного счетчика?
- 6 ПРОАНАЛИЗИРУЙТЕ! Прямой счетчик на 4 бита находится в одном из следующих исходных состояний:

- | | |
|----------|----------|
| a) 0000; | f) 1010; |
| b) 0010; | g) 1100; |
| c) 0100; | h) 1111; |
| d) 1000; | i) 0101; |
| e) 1001; | j) 0110. |

Каким станет состояние счетчика после поступления на вход 5-и импульсов? А 8-и импульсов?

- 7 ПРОАНАЛИЗИРУЙТЕ! Инверсный счетчик на 4 бита находится в исходном состоянии 1001. Каким станет состояние счетчика после поступления m входных импульсов? Число m может принимать значения 1, 4, 5, 8, 11, 17.
- 8 ПРОГРАММИРУЙТЕ! Напишите на ПАСКАЛЕ программу, которая моделирует работу прямого двоичного счетчика на n битов.

5.7. Генераторы импульсов

Импульсы используются в цифровых устройствах для того, чтобы обеспечить их последовательную работу во времени. Как правило, генераторы импульсов выполняются на основе логических элементов и элементов задержки.

Элемент задержки представляет собой электронную схему, реализующую логическую функцию повторения $y = x$, причем выходной сигнал y повторяет входной сигнал x с задержкой (опозданием) на Δ единиц времени.

Из курса физики известно, что скорость распространения сигналов конечна. Следовательно, любой проводник может рассматриваться как элемент задержки. Для того чтобы увеличить «инерционность» электронных схем и

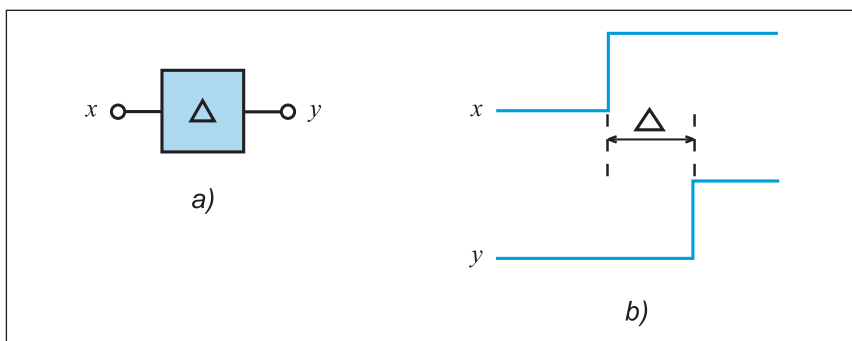


Рис. 5.20. Обозначение (a) и временные диаграммы (b) элемента задержки

достичь значительных задержек, в их состав включаются конденсаторы и резисторы. В этом случае задержка Δ определяется емкостью и сопротивлением соответствующих компонентов.

На рис. 5.20 представлены обозначение и временные диаграммы элемента задержки.

Простейшая схема **генератора периодических импульсов**, реализованного на основе элемента задержки и логического элемента **И-НЕ**, представлена на рис. 5.21. В исходном состоянии $x = 0$ и $y = 1$, а на выходе элемента задержки

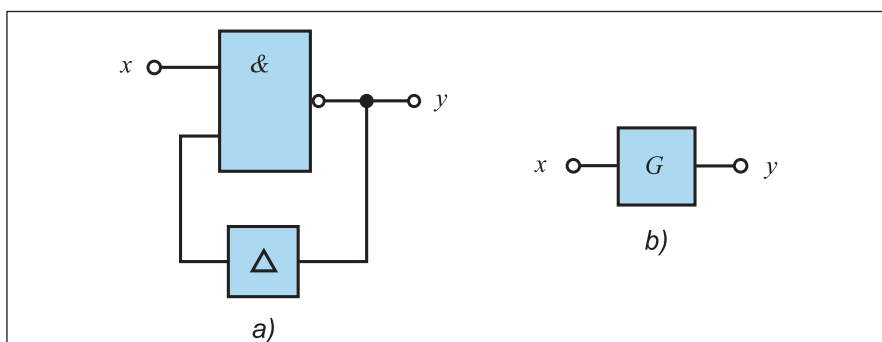


Рис. 5.21. Схема (a) и обозначение генератора периодических импульсов (b)

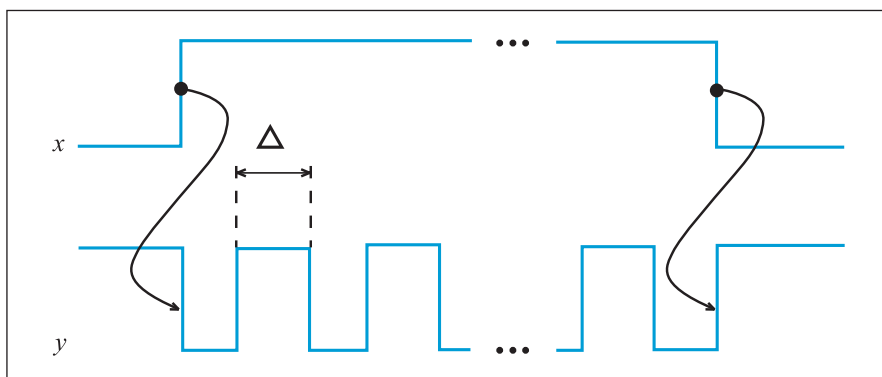


Рис. 5.22. Временные диаграммы генератора периодических импульсов

поддерживается значение логической 1. Когда на вход подается сигнал запуска $x = 1$, сигнал на выходе принимает значение 0 (рис. 5.22). Затем логическое значение 0 с задержкой Δ подается на второй вход логического элемента *И-НЕ*. Следовательно, выход принимает значение 1. Уровень логической 1 после задержки Δ вновь будет подан на вход логического элемента *И-НЕ*, заставляя тем самым появиться на выходе значение $y = 0$, и т. д.

Следовательно, на выходе у генератора вырабатывается последовательность импульсов длительности Δ . Процесс генерации может быть остановлен путем подачи на управляющий вход сигнала $x = 0$.

Вопросы и упражнения

- ❶ Для чего предназначен элемент задержки? Нарисуйте временные диаграммы рассматриваемого элемента.
- ❷ Объясните, как работает генератор периодических импульсов. От чего зависит длительность импульсов?
- ❸ ПРОЕКТИРУЙТЕ! Замените логический элемент *И-НЕ*, входящий в состав генератора периодических импульсов, представленного на рис. 5.21, логическим элементом *ИЛИ-НЕ*. Объясните, как будет работать данная схема. Нарисуйте временные диаграммы полученного генератора.
- ❹ ИССЛЕДУЙТЕ! Известно, что изменение физических параметров не может осуществляться мгновенно. Следовательно, любой логический элемент обладает задержкой δ , называемой **паразитной задержкой**, значение которой зависит от особенностей соответствующей схемы.
Исключите из схемы, представленной на рис. 5.21, элемент задержки, подавая выходной сигнал логического элемента *И-НЕ* прямо на его вход. Объясните, как будет работать полученная схема. От чего зависит длительность импульсов на выходе логического элемента? Нарисуйте соответствующие временные диаграммы.
- ❺ ПРОАНАЛИЗИРУЙТЕ! Последовательностная схема состоит из логического элемента *НЕ*, сигнал с выхода которого подается прямо на его вход. Как будет работать данная схема?

Глава 6

УСТРОЙСТВО И РАБОТА КОМПЬЮТЕРА

6.1. Функциональная схема компьютера

Функциональная схема цифрового компьютера представлена на рис. 6.1.

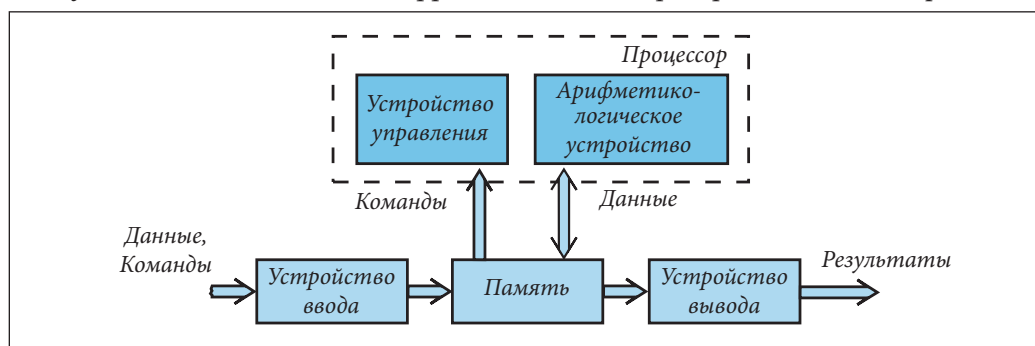


Рис. 6.1. Функциональная схема компьютера

В соответствии с данной схемой цифровой компьютер содержит следующие **функциональные блоки**:

- блок памяти, предназначенный для хранения исходных, промежуточных и конечных данных задачи вместе с командами, указывающими последовательность вычислений;
- арифметико-логическое устройство, необходимое для выполнения элементарных арифметических и логических операций;
- одно или более устройств ввода и соответственно вывода, необходимых для связи компьютера с внешней средой;
- центральное устройство управления, которое генерирует последовательности управляющих сигналов, необходимых для последовательного выполнения команд.

Арифметико-логическое устройство (АЛУ) и центральное устройство управления (ЦУУ) образуют **центральное устройство обработки информации, а проще говоря — процессор**.

Память современных компьютеров организована на двух уровнях: блок внутренней памяти с высокой скоростью работы и один или более блоков внешней памяти с низкой скоростью, но с емкостью во много раз большей, чем у внутренней памяти.

Внутренняя память (называемая иногда главной, центральной или оперативной памятью) хранит программу и используемые ею данные во время

выполнения этой программы. Ее наличие является одним из важнейших условий работы компьютера.

Внешняя память играет роль хранилища больших объемов информации и часто используемых программ с возможностью их загрузки во внутреннюю память за короткий интервал времени. В настоящее время в качестве внешней памяти используются устройства на магнитных дисках или лентах, устройства на оптических дисках, *флэш*-память и т. п.

Блоки внешней памяти и устройства ввода-вывода называются **периферийным оборудованием**.

Для обеспечения эффективного взаимодействия процессора, внутренней памяти и периферийного оборудования, в случае персональных компьютеров, их функциональная схема реализуется физически в соответствии с блок-схемой, представленной на *рис. 6.2*

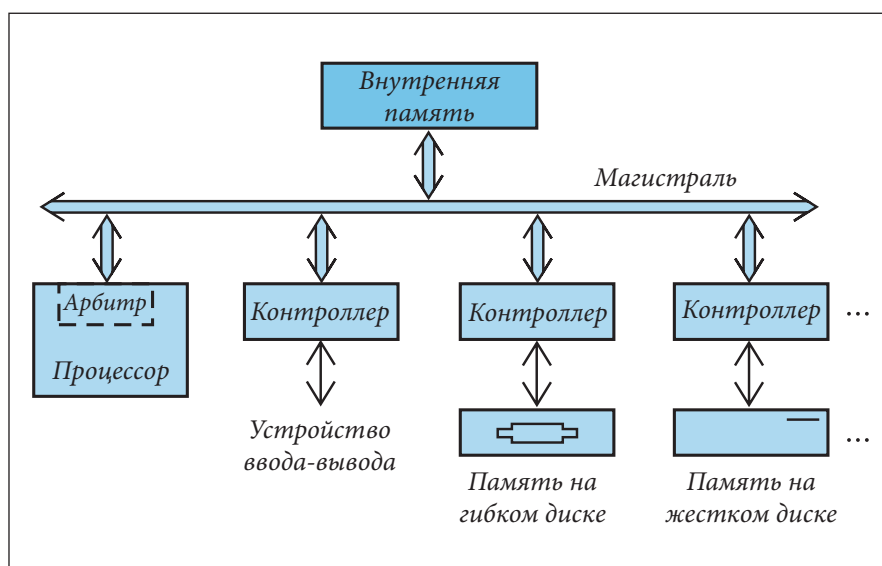


Рис. 6.2. Блок-схема персонального компьютера

Из *рис. 6.2* видно, что все современные компьютеры имеют **модульную конфигурацию**. Каждый модуль (контроллер, принтеры, устройства на магнитных дисках и т. д.) функционирует независимо и, следовательно, может включаться или исключаться из состава компьютера независимо от остальных. Таким образом, конфигурация компьютера может быть изменена в зависимости от области применения вычислительной системы.

Например, типовая издательская система содержит несколько видов принтеров: механический — для черновых текстов, лазерный или цветной — для макетов страниц, рисунков и фотографий и т. п. Система оперативного управления большим объемом данных должна содержать магнитные диски большой емкости, а компьютер, предназначенный для видеомонтажа, должен иметь в комплекте видеокамеры, мониторы с соответствующим разрешением, клавиатуры, аналогичные режиссерскому пульта, и т. п.

Вопросы и упражнения

- 1 Назовите функциональные блоки компьютера и объясните их назначение.
- 2 В чем заключается роль внутренней памяти? Как реализуется внешняя память современных компьютеров?
- 3 Назовите известные вам периферийные устройства.
- 4 Назовите компоненты персонального компьютера и объясните их назначение.
- 5 Расскажите о структуре и взаимодействии компонентов компьютера.
- 6 Как может быть изменена конфигурация вычислительной системы? Какие преимущества имеет модульная конфигурация компьютера?
- 7 **ИССЛЕДУЙТЕ!** Нарисуйте блок-схему компьютера, на котором вы работаете. Какие компоненты для функционирования компьютера являются обязательными, а какие — нет?
- 8 **ЭКСПЕРИМЕНТИРУЙТЕ!** Как можно подключить к компьютеру дополнительное устройство на магнитных дисках? Лазерный принтер? Считыватель документов (сканер)? Видеокамеру?
- 9 **ИССЛЕДУЙТЕ!** Используя поисковую систему, соберите информацию из Интернета об эволюции функциональных схем цифровых компьютеров и напишите краткое эссе на эту тему. Выделите факторы, которые привели к широкому использованию функциональных схем с магистралью.

6.2. Форматы команд

Для решения любой задачи компьютер должен знать в каждый момент времени как операцию, которая должна быть выполнена, так и данные, которые в ней участвуют. Эти операции сообщаются компьютеру с помощью команд.

Команда представляет собой последовательность двоичных цифр, с помощью которой процессору указывается операция, которую необходимо выполнить, и расположение операндов.

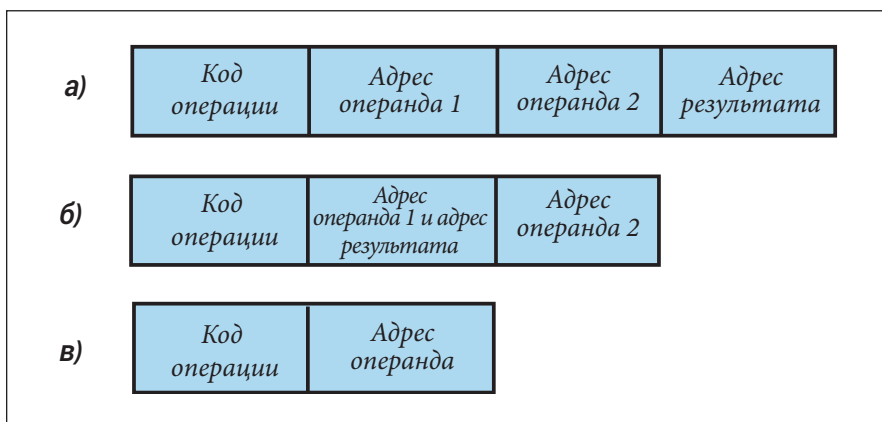


Рис. 6.3. Формат команды с тремя (а), двумя (б) и одним адресом (в)

Соответствующая двоичная последовательность, называемая иногда словом **команды**, разделена на поля, каждое из которых имеет строго определенное назначение. Количество и назначение полей называются **форматом команды**. На *рис. 6.3* представлены форматы, используемые в современных компьютерах.

В общем случае для выполнения заданной операции необходимо, чтобы ее команда содержала три адреса (*рис. 6.3а*). Первые два адреса используются для получения операндов, над которыми будет осуществлена операция, указанная в поле *Код операции*. Результат операции будет помещен по адресу, указанному в поле *Адрес результата*.

Рассмотрим пример. Предположим, что арифметические и логические операции закодированы так, как указано ниже (для простоты будем использовать десятичные эквиваленты соответствующих двоичных полей):

- 01 – сложение;
- 02 – вычитание;
- 03 – логическая операция *И*;
- 04 – логическая операция *ИЛИ*.

Команда

01 100 110 215

сообщает процессору, что необходимо сложить числа из ячеек 100 и 110 и разместить полученную сумму в ячейку с адресом 215.

Команда

02 100 110 215

сообщает процессору, что из числа, записанного в ячейку 100, необходимо вычесть число, записанное в ячейку 110. Полученный результат будет помещен в ячейку 215.

Аналогичным образом, команда

03 200 300 100

задает логическую операцию *И* над битами слов из ячеек 200 и 300. Результат будет размещен в ячейке 100.

Отметим, что в команде указываются не значения операндов, а **адреса ячеек**, в которых находятся соответствующие данные. Указанное обстоятельство делает возможным использование одних и тех же программ для обработки любых исходных данных. Тот факт, что команды работают с адресами, содержимое которых должно быть обработано, а не с самим содержимым, составляет основной принцип работы цифровых компьютеров. Рассматриваемый принцип позволяет разрабатывать и вводить в компьютер программы независимо от конкретных данных, которые будут этой программой обрабатываться.

В формате с тремя адресами (*рис. 6.3а*) **адреса задаются явно**. Для более компактного представления команд используется **неявное задание** некоторых адресов. В этом случае слово команды не содержит полей, предназначенных для косвенных адресов.

В частности, если результат, полученный после выполнения любой операции, помещается по адресу одного из операндов, то в соответствующий

формат будут входить только два адреса (рис. 6.3б). Следовательно, адрес результата задается неявно. Например, команда

```
01 100 110
```

сообщает процессору, что необходимо сложить числа из ячеек 100 и 110 и поместить полученную сумму в ячейку 100. Очевидно, что после записи суммы исходное число, находившееся в ячейке 100, будет утеряно.

Установлено, что формат с двумя адресами (самый распространенный в настоящее время) обеспечивает написание программ с количеством команд, сравнимым с количеством команд при использовании большего числа адресов.

Формат с одним адресом (рис. 6.3в) применяется в компьютерах, процессор которых содержит специальный регистр, называемый **аккумулятором**. В аккумуляторе хранится первый операнд, и в него же помещается результат выполнения соответствующей операции. Следовательно, адрес первого операнда и адрес результата задаются неявно. Например, одноадресная команда

```
01 100
```

сложенное число из аккумулятора с числом, хранящимся в ячейке 100, а полученная сумма будет помещена в аккумулятор. Естественно, исходное число из аккумулятора будет утеряно.

Команды с одним адресом эффективны с точки зрения длины слова и скорости работы компьютера. Однако программа, написанная с использованием одноадресных команд, будет длиннее, чем программа, написанная с использованием двух- или трехадресных команд.

Отметим, что современные компьютеры имеют команды различных форматов. Формат каждой команды указывается в поле *Код операции*.

Вопросы и упражнения

- 1 Перечислите форматы команд, применяемых в современных компьютерах. Объясните способ неявного задания адресов операндов.
- 2 Объясните назначение полей команд с тремя и двумя адресами.
- 3 Как определяются адреса операндов и адрес результата в случае одноадресных команд?
- 4 Объясните, как будут выполняться следующие трехадресные команды:

a) 01 200 201 202;

c) 03 100 150 250;

b) 04 202 201 200;

d) 02 250 300 310.

Коды арифметических и логических команд приведены в предыдущем параграфе.

- 5 Каким станет содержимое ячейки 100 после выполнения команды

```
01 200 300 100,
```

если в ячейки 200 и 300 записаны числа 17 и 31 соответственно?

⑥ Объясните, как будут выполнены следующие двухадресные команды:

a) 01 200 201;

c) 03 100 150;

b) 04 202 201;

d) 02 250 300.

⑦ Каким станет содержимое ячейки 200 после выполнения команды

01 200 100,

если в ячейки 100 и 200 записаны числа 18 и 32 соответственно?

⑧ Объясните, как будут выполнены следующие одноадресные команды:

a) 01 100;

c) 02 400;

b) 03 200;

d) 04 150.

⑨ Каким станет содержимое аккумулятора после выполнения команды

01 100,

если перед выполнением команды в ячейке 100 было записано число 12, а в аккумуляторе — число 26?

⑩ Назовите преимущества и недостатки форматов команд с тремя, двумя или с одним адресом.

⑪ ТЕМАТИЧЕСКОЕ ИССЛЕДОВАНИЕ. Найдите в Интернете информацию о типах компьютеров, которые используют трех-, двух- или одноадресный формат. Определите области применения, преимущества и недостатки этих типов компьютеров.

⑫ ИССЛЕДУЙТЕ! Пользуясь любой поисковой системой, соберите информацию из Интернета об эволюции форматов команд, используемых в цифровых компьютерах, и напишите краткое эссе на эту тему. Выделите факторы, которые привели к широкому применению форматов, используемых в персональных компьютерах.

6.3. Типы команд

Команды любого компьютера делятся на четыре группы:

- команды для обработки данных, которые осуществляют арифметические и логические операции над данными, определенными с помощью операндов;
- команды для передачи данных, которые пересылают информацию между регистрами и/или ячейками без ее изменения;
- команды перехода, которые в зависимости от результатов проверки некоторого условия изменяют последовательность выполнения команд программы;
- команды ввода-вывода, которые обеспечивают связь компьютера с внешней средой.

Команды для обработки данных обрабатывают данные, хранящиеся во внутренней памяти и регистрах процессора. Наиболее известными среди команд данной группы являются те, которые выполняют арифметические операции: сложение, вычитание, умножение и деление.

Логические команды типа *И*, *ИЛИ*, *НЕ* также являются командами для обработки данных и выполняются над отдельными битами двоичной информации. В класс команд для обработки данных включаются также и команды типа: *стирание* содержимого некоторой ячейки или определенного регистра, *дополнение* содержимого некоторой ячейки, *увеличение* на единицу (инкрементация) содержимого некоторого регистра и т. п. Наконец, к классу команд для обработки данных относятся команды *сдвига* двоичных слов, в которых часть адреса команды содержит целое число, определяющее количество позиций, на которое осуществляется сдвиг.

Команды для передачи данных пересылают информацию между ячейками внутренней памяти, между регистрами и между ячейками и регистрами без изменения передаваемой информации. В команде должны быть определены явным или косвенным образом адрес источника и адрес приемника передаваемой информации. Во время и после переноса информация источника остается неизменной. Наиболее часто употребляемые команды этой группы — те, с помощью которых содержимое произвольных ячеек памяти переносится в определенный регистр или аккумулятор, а также команда обратного переноса: из регистра — в ячейку внутренней памяти.

Команды перехода используют для изменения порядка выполнения команд. В обычном режиме команды любой программы анализируются и выполняются последовательно — точно в том же порядке, в котором они размещены в памяти. Этот порядок может быть изменен с помощью команд условного и безусловного перехода.

Команды условного перехода обеспечивают выбор определенной ветви, по которой будет продолжено выполнение программы в зависимости от проверяемого условия. Использование команд условного перехода дает программисту возможность принимать логические решения в ходе выполнения программы.

Команда безусловного перехода содержит адрес команды, на которую будет осуществлен переход и которая тем самым будет выполнена следующей.

Команды ввода-вывода обеспечивают связь компьютера с периферийным оборудованием. Устройства, с которыми будет осуществлена операция ввода-вывода, задаются в адресной части соответствующей команды. Как правило, команды этого вида содержат не только информацию о характере обмена (ввод или вывод), но и дополнительные параметры, необходимые для правильной работы периферийных устройств. В этих же командах определяются регистры или ячейки, в которые будут помещены или из которых будут взяты соответствующие данные.

Вопросы и упражнения

- ❶ Как классифицируются команды компьютера? Укажите назначение команд каждой группы.
- ❷ Приведите несколько примеров команд для обработки данных. Дайте оценку числа всевозможных команд для обработки данных.
- ❸ В чем назначение команд передачи данных? Оцените число всевозможных команд передачи данных.
- ❹ Когда и как применяют команды перехода? Какие условия проверки могут анализироваться этими командами?

- ⑤ Укажите назначение команд ввода-вывода. Какую информацию содержат эти команды?
- ⑥ ИССЛЕДУЙТЕ! Узнайте классификацию и состав команд компьютеров, с которыми вы работаете.

6.4. Машинный язык и язык ассемблера

Для решения любой задачи в память компьютера необходимо загрузить соответствующую программу и данные, предназначенные для обработки. Команды программы и обрабатываемые данные записываются во внутреннюю память в виде последовательностей двоичных цифр, которые центральное устройство управления может извлекать и исполнять.

Программы, представленные в виде двоичных последовательностей, напрямую исполняемых компьютером, называются программами на языке машинных кодов (машинном языке).

Для пользователя программа в машинных кодах может быть представлена в виде последовательностей двоичных цифр или, более компактно, — восьмеричных, десятичных или шестнадцатеричных чисел, записываемых в соответствующих ячейках памяти.

Разработка программ на языке машинных кодов является утомительной и неэффективной работой. Для упрощения процесса разработки программ договорились записывать команды на некотором символическом языке, называемом **языком ассемблера**. В данном языке коды операций представляются группами символов так, чтобы они как можно лучше подсказывали смысл операции. Эта группа символов длиной, как правило, три известна под названием **мнемоники команды**.

Например, коды команд учебного компьютера из предыдущего параграфа могут быть обозначены символически в соответствии с *таблицей 6.1*.

Таблица 6.1

Мнемоника команд

Код	Мнемоника	Значение команды
01	ЗАГ	Загрузить аккумулятор
02	ЗАП	Запомнить аккумулятор
03	СЛЖ	Сложение
04	ВЫЧ	Вычитание
05	БП	Безусловный переход
06	УП	Условный переход
07	СТОП	Стоп

Адреса ячеек внутренней памяти могут быть заданы с помощью символических обозначений, выбранных пользователем. Названия должны напоминать (подсказывать) назначение (смысл) содержимого соответствующих ячеек.

Например, ячейка 185, в которой хранится число x , может быть обозначена с помощью X ; ячейку 213 для числа y обозначим через Y ; ячейку 200, в которую

будет помещена сумма $x + y$, обозначим через S . На языке ассемблера фрагмент программы для сложения чисел x и y примет форму:

```
INC X
ADU Y
MEM S.
```

В общем случае существует прямое соответствие между записью команд на языке ассемблера и записью этих же команд на языке машинных кодов, которое облегчает трансляцию (перевод) программ на языке ассемблера в программы на языке машинных кодов.

Трансляция состоит в замене mnemonic команд и символических адресов на соответствующие двоичные последовательности. Такая замена осуществляется специальной программой, которую называют программой ассемблера, или просто ассемблером.

Язык машинных кодов и язык ассемблера относят к **машинно-зависимым языкам**. Такая зависимость состоит в том, что форматы, коды и мнемоника команд выражают (передают) внутреннюю структуру компьютера. Программы, написанные на этих языках, являются самыми короткими и быстрыми, но сам процесс программирования требует большого объема работы. Упрощение процесса программирования обеспечивается использованием машинно-независимых языков (*FORTRAN, BASIC, PASCAL, C* и т. д.), в которых операции обработки и типы данных не связаны с внутренним устройством конкретной модели компьютера. Однако, к сожалению, отрыв пользователя от внутренней структуры компьютера снижает эффективность соответствующих программ.

Вопросы и упражнения

- 1 В чем отличие между машинным языком и языком ассемблера?
- 2 Как представляются коды команд и адреса ячеек на языке ассемблера?
- 3 В чем назначение трансляции и как она реализуется для программ, написанных на языке ассемблера?
- 4 ПРОГРАММИРУЙТЕ! Пусть символ X обозначает ячейку 100, символ Y — ячейку 101, а символ S — ячейку 102. Представьте на языке ассемблера (см. табл. 6.1) следующие программы:

a) 01 100
04 101
02 102
02 100

b) 01 100
02 100
03 100

c) 01 101
04 100
02 102

d) 01 101
03 101
03 101
03 101
02 102

e) 01 100
02 101
03 101
02 100

f) 01 100
02 102
01 101
02 100

Каким будет содержимое ячеек 100, 101 и 102 до и после выполнения каждой программы?

- 5 Пусть символические обозначения X , Y и S определяют соответственно ячейки 100, 200 и 300. Транслируйте (вручную) следующие программы, написанные на языке ассемблера:

a)

ЗАГ	X
ВЫЧ	Y
ЗАП	S
ЗАГ	Y

b)

ЗАГ	X
ЗАГ	X
УП	S
СТОП	

c)

ЗАГ	Y
СЛЖ	X
ЗАП	S
СТОП	

d)

ЗАГ	Y
СЛЖ	Y
СЛЖ	Y
СЛЖ	Y
ЗАП	S

e)

ЗАГ	X
ЗАП	S
ЗАГ	Y
ЗАП	X
ЗАГ	X

f)

ЗАГ	X
ЗАП	S
ЗАГ	Y
ЗАП	X
ЗАГ	S

Объясните, как будет выполнена каждая из программ.

- 6 ТЕМАТИЧЕСКОЕ ИССЛЕДОВАНИЕ. Чем отличаются машинно-зависимые и машинно-независимые языки? Перечислите преимущества и недостатки каждого из названных типов языка.
- 7 ИССЛЕДУЙТЕ! Пользуясь Интернетом в качестве источника информации, сделайте краткое описание машинных кодов и языков ассемблера компьютеров, с которыми вы работаете: коды и команды, их классификация.

6.5. Аппаратные и программные ресурсы компьютера

Общее число команд любого компьютера зависит, в первую очередь, от его мощности. В случае большой вычислительной системы это число может превысить 1000, в то время как для очень маленьких компьютеров оно не больше 100. Некоторая операция может быть выполнена на одних компьютерах с помощью единственной команды, в то время как в других компьютерах, для которых нет такой команды, эта же операция выполняется с помощью некоторой последовательности имеющихся в наборе команд.

Операции, осуществляемые с помощью электронных компонентов компьютера, известны как **аппаратно-реализуемые операции**, в то время как операции, выполняемые с помощью некоторой последовательности команд, известны как **программно-реализуемые операции**. Например, операция извлечения квадратного корня в одном типе компьютеров может быть выполнена с помощью электронных компонентов, а в другом типе компьютеров — с помощью подпрограмм. Разделение на аппаратную и программную реализацию зависит от типа компьютера. На *рис. 6.4* представлено такое разделение для компьютера средней мощности.

Аппаратно-реализуемые операции выполняются с помощью одной команды, в то время как программно-реализуемые операции требуют большего числа команд. Следовательно, аппаратно-реализуемые операции выполняются быстрее, хотя соответствующий компьютер сложнее и, следовательно, дороже. Напротив, программно-реализуемые операции выполняются медленно, хотя соответствующие компьютеры проще и, очевидно, дешевле.

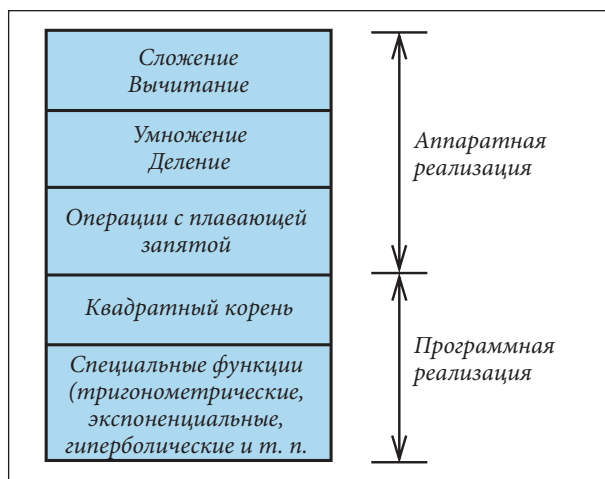


Рис. 6.4. Разделение между аппаратной и программной реализацией

Из анализа принципов работы процессора следует, что все устройства любого компьютера становятся бесполезными в отсутствие программ, которые управляют ходом выполнения операций, необходимых для решения поставленных задач. Точно так же программы становятся бесполезными при отсутствии соответствующих цифровых устройств. Следовательно, использование вычислительной техники возможно только при наличии как оборудования, называемого **техническим обеспечением**, так и соответствующих программ, называемых **программным обеспечением**.

Техническое обеспечение любой современной вычислительной системы включает процессор, внутреннюю память, устройства внешней памяти, устройства ввода-вывода и т. п. **Программное обеспечение** включает подпрограммы для выполнения программно-реализуемых операций, программы для доступа к устройствам ввода-вывода, ассемблеры, редакторы текста, компиляторы алгоритмических языков и, естественно, программы, разрабатываемые каждым пользователем.

Отметим, что в специализированной литературе техническое обеспечение иногда называют английским термином *hardware* (от англ. «металлические изделия»), а программное обеспечение — термином *software* (от англ. «мягкие изделия»). Соответственно реализация с помощью аппаратных средств называется реализацией с помощью *hardware*, а программная реализация — реализацией с помощью *software*.

Вопросы и упражнения

- ❶ От чего зависит общее количество команд произвольного компьютера?
- ❷ Как выполняются операции по обработке данных в случае их аппаратной и программной реализации?
- ❸ **ИССЛЕДУЙТЕ!** Изучите набор команд компьютера, на котором вы работаете, и определите метод реализации следующих операций:
 - умножение и деление двоичных чисел;
 - сложение и вычитание чисел с плавающей запятой;

- умножение и деление чисел с плавающей запятой;
 - извлечение квадратного корня;
 - вычисление тригонометрических функций.
- 4 ТЕМАТИЧЕСКОЕ ИССЛЕДОВАНИЕ. В чем преимущества и недостатки аппаратной реализации? А программной реализации?
- 5 ЭКСПЕРИМЕНТИРУЙТЕ! Из чего состоит техническое и программное обеспечение вычислительной системы? Какими ресурсами обладает компьютер, на котором вы работаете?

6.6. Внешняя память на магнитных лентах и дисках

Принцип работы рассматриваемых видов памяти состоит в регистрации (записи) информации на магнитном слое, находящемся в движении. Магнитный слой нанесен на нейтральный носитель, как правило, на ленту из гибкого материала или алюминиевый диск. В качестве магнитного слоя чаще всего применяется окись железа или тончайшие металлические пленки из кобальт-никелевого сплава, нанесенные (напыленные) в вакууме.

Запись и чтение информации осуществляются с помощью магнитной головки, изображенной на рис. 6.5.

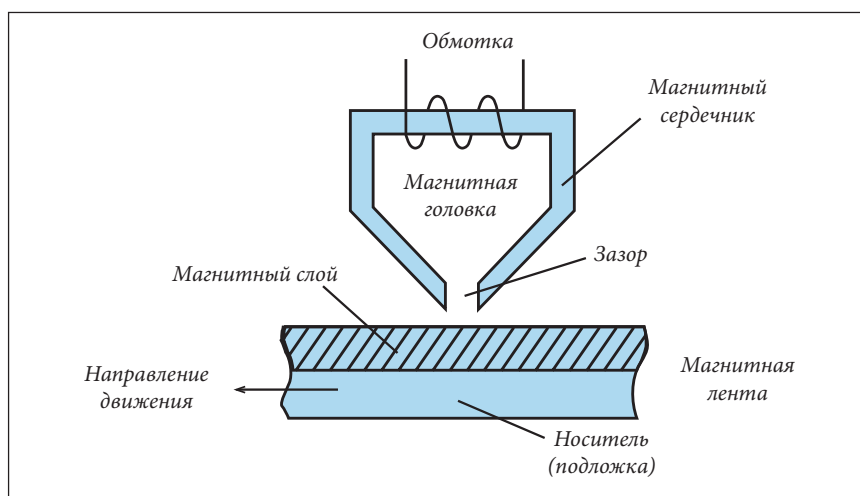


Рис. 6.5. Магнитная головка для записи и чтения информации

Головка состоит из сердечника, собранного, как правило, из очень тонких пластин (0,05 мм) из пермаллоя, и обмотки.

В немагнитном слое магнитные поля частичек окиси железа ориентированы хаотично и взаимно компенсируют друг друга. Чтобы **записать** двоичную цифру 0 или 1, через обмотку магнитной головки пропускается соответствующий импульс тока. Импульс, проходящий по обмотке, создает в зазоре сильное магнитное поле, которое намагничивает слой, находящийся в данный момент времени под головкой. Направление намагничивания, а значит, и записанная двоичная информация зависят от направления тока в

обмотке магнитной головки. На *рис. 6.6* приведен пример записи двоичной последовательности 101101 на подвижный магнитный слой.

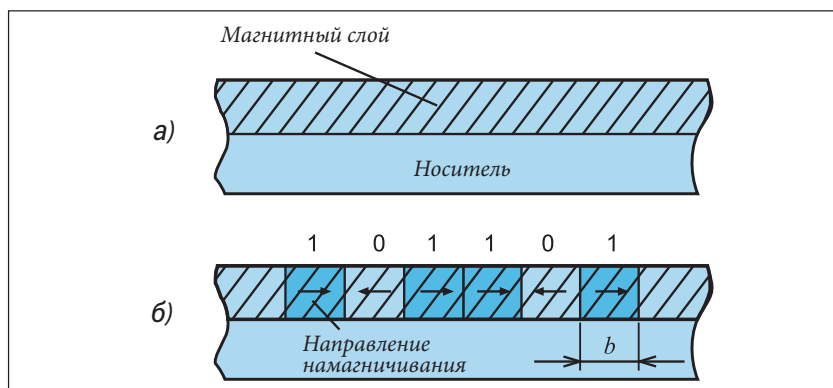


Рис. 6.6. Состояние магнитного слоя до записи (а) и после (б)

Расстояние b определяет длину участка, необходимого для записи одной двоичной цифры. Его величина зависит от скорости движения носителя, физических свойств магнитного слоя, от конструкции магнитной головки и т. д.

Количество элементов двоичной памяти на единицу длины носителя называется плотностью записи информации.

В случае магнитных носителей плотность записи задается величиной $1/b$. Конкретные значения плотности записи меняются в зависимости от устройств записи и фирмы производителя, находясь в пределах сотен и тысяч битов на миллиметр длины носителя.

Во время операции чтения магнитное поле частичек окиси железа, проходя мимо зазора магнитной головки (*рис. 6.5*), индуцирует в обмотке сигнал порядка 10^{-3} вольт. Этот сигнал усиливается и преобразуется в стандартный сигнал, который представляет соответствующую двоичную цифру 0 или 1.

В большинстве случаев **внешняя память на магнитной ленте** представляет собой автономное периферийное устройство, которое передает информацию из/во внутреннюю память компьютера после приема соответствующих команд от процессора. Устройство памяти на магнитной ленте (*рис. 6.7*) состоит из механизма протяжки ленты, устройства записи-чтения и соответствующих схем управления.

Операция чтения или записи осуществляется во время перемещения ленты. Между двумя последовательными операциями записи/чтения лента останавливается. Очевидно, что записанная информация может быть прочитана только в порядке ее физического размещения на ленте. Вследствие этого устройства на магнитных лентах называются **устройствами внешней памяти с последовательным доступом**.

Время, необходимое для выбора требуемой информации из множества данных, записанных на некотором носителе, называется временем доступа.

Время доступа устройства на магнитной ленте зависит от скорости движения ленты и места размещения читаемой информации: в начале, в конце или в середине ленты. В случае информации, записанной в конце ленты, время доступа может достигать нескольких минут.

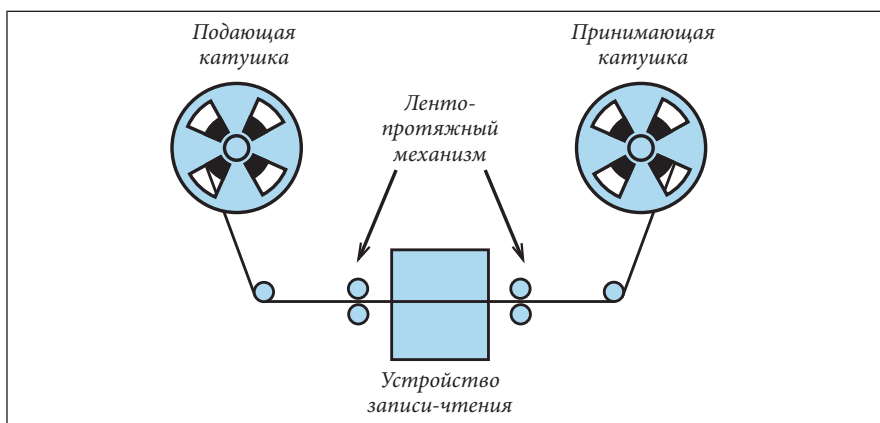


Рис. 6.7. Устройство памяти на магнитной ленте

Емкость памяти произвольной магнитной ленты зависит от плотности записи, количества дорожек, длины ленты и составляет порядка 10^8 байтов.

Из-за большого времени доступа и относительно малой емкости магнитные ленты применяют, как правило, только для архивирования информации на длительные периоды времени, до 50 лет.

Устройство на магнитных дисках в настоящее время является самым распространенным типом внешней памяти цифровых компьютеров. Носитель информации составлен из набора дисков, которые могут быть фиксированными или съемными. Диски вращаются со скоростью порядка тысячи оборотов в минуту. Каждый диск покрыт слоем ферромагнитного материала. Для записи и чтения информации над каждой поверхностью диска установлена отдельная магнитная головка (рис. 6.8).

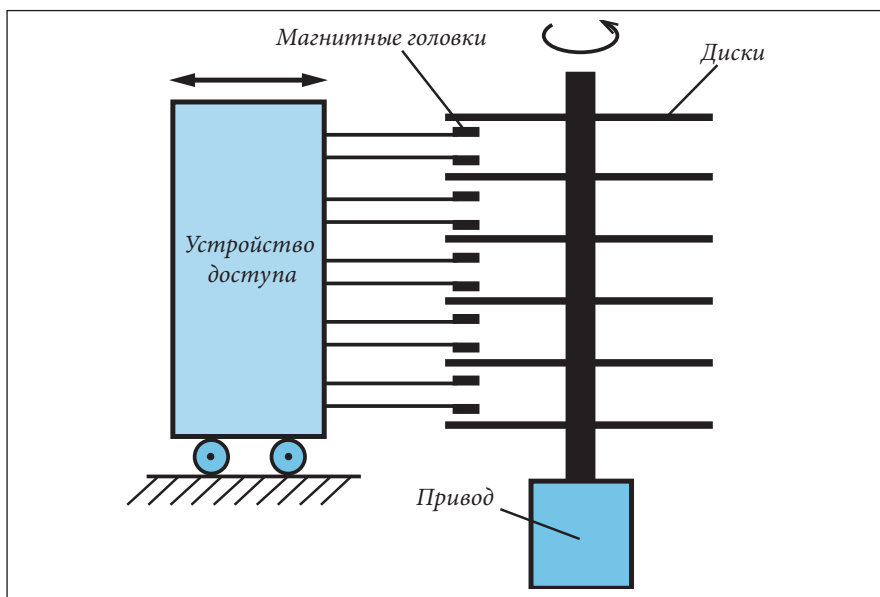


Рис. 6.8. Дисковое устройство с подвижными головками

Головки смонтированы на подвижном держателе, управляемом механизмом для точной установки на нужную дорожку. Все головки дискового блока памяти позиционируются одновременно.

Время доступа дисковых устройств складывается из времени, необходимого для перемещения набора магнитных головок от текущего к требуемому цилиндру (*рис. 6.8*), и из времени, необходимого для того, чтобы соответствующий сектор диска переместился прямо под магнитную головку. На практике пользуются **средним временем доступа**, которое для современных дисковых устройств составляет порядка 10^{-3} секунд.

Отметим, что в мощных компьютерах используются дисковые устройства с неподвижными магнитными головками — по одной головке на каждую дорожку. Указанные устройства обеспечивают время доступа порядка 10^{-4} секунды, однако они дорогостоящие.

В прошлом для обмена информацией между компьютерами использовались одиночные диски из гибкого материала, которые назывались **гибкими дисками**, или **дискетами**. Дискета помещалась в кассету из пластмассы или в особый конверт. Физическая организация данных на дискетах была такая же, как и для пакетов из дисков, однако соответствующие устройства были значительно проще, а значит, и дешевле. Для того чтобы отличать их от гибких дисков, внутренние диски персональных компьютеров стали называть **жесткими дисками**, **hard-дисками** или **винчестерами** (*winchester*).

Емкость памяти пакета из магнитных дисков зависит от количества дисков, количества цилиндров и плотности записи. В настоящее время достигнута емкость порядка 10^{12} байтов для одного диска.

Отметим, что в последние годы вместо жестких дисков стали применять устройства хранения данных, использующие полупроводниковые запоминающие устройства, называемые *чип-накопителями* или *твердотельными накопителями* (*Solid-State Drive*). Твердотельные накопители более устойчивы к механическим воздействиям и имеют меньшее время доступа. Однако на данный момент емкость этих накопителей ниже, чем у жестких дисков, а их стоимость выше.

Вопросы и упражнения

- 1 Как представляются двоичные цифры 0 и 1 при магнитной записи?
- 2 Для чего предназначена магнитная головка?
- 3 От чего зависит плотность магнитной записи информации?
- 4 Как считывается информация, записанная на магнитном слое?
- 5 Объясните, как работает устройство памяти на магнитной ленте, изображенное на *рис. 6.7*.
- 6 ИССЛЕДУЙТЕ! От чего зависит емкость памяти магнитной ленты?
- 7 Магнитная лента имеет длину 750 м. Запись информации осуществляется на 8 дорожках плюс одна дорожка для бита четности. Объем записанной информации составляет 47 МБ. Определите плотность записи информации на магнитной ленте.
- 8 Скорость магнитной ленты равна 2 м/с. На подающей катушке (*рис. 6.7*) имеется 750 м ленты. Определите время доступа к данным, которые находятся в середине ленты.
- 9 Как работает устройство на магнитных дисках с подвижными головками?

- 10 Как организована информация на пакете магнитных дисков?
- 11 В чем отличие между устройствами внешней памяти с прямым и последовательным доступом?
- 12 От чего зависит время доступа устройств на магнитных дисках?
- 13 **ТВОРИТЕ!** Используя источники информации в Интернете, напишите краткое эссе об эволюции дискет.
- 14 **ЭКСПЕРИМЕНТИРУЙТЕ!** Для жесткого диска, с которым вы работаете, определите:
 - емкость диска;
 - среднее время доступа.
- 15 **ТВОРИТЕ!** Используя источники информации в Интернете, напишите краткое эссе об эволюции жестких дисков.
- 16 **ТЕМАТИЧЕСКОЕ ИССЛЕДОВАНИЕ.** Разработайте небольшое исследование, в котором осветите преимущества и недостатки жестких дисков и полупроводниковых запоминающих устройств (SSD). Проанализируйте частоту использования этих запоминающих устройств в настольных и портативных персональных компьютерах (*ноутбуках/лэптопах*), продаваемых в нашей стране.

6.7. Внешняя память на оптических дисках

Принцип работы устройств памяти на оптических дисках состоит в хранении информации на движущемся отражающем слое. Отражающий слой из алюминия, золота или серебра нанесен на прозрачную основу из пластмассы.

В зависимости от режима записи и чтения информации различаем:

1) **Оптические диски только для чтения.** Информация на такие диски записывается производителем и не может быть изменена пользователем. Для таких дисков используется английское обозначение *CD-ROM (Compact Disc-Read Only Memory)*.

2) **Записываемые оптические диски.** Информация на такие диски записывается самим пользователем, но только один раз. В дальнейшем таким диском можно пользоваться многократно только для чтения. Английское обозначение таких дисков *CD-R (Compact Disc-Recordable)*.

3) **Перезаписываемые оптические диски.** Рассматриваемые диски допускают многократные циклы записи-чтения информации и обозначаются как *CD-RW (Compact Disc-ReWritable)*.

Для обеспечения совместимости устройств записи/чтения формат данных и размеры оптических дисков стандартизованы. На *рис. 6.9* представлена структура оптического диска *CD-ROM только для чтения*, предназначенного для широкого круга пользователей.

На таких дисках записанные двоичные цифры представляют собой последовательность углублений (на английском *pit*) на одной из поверхностей диска. Углубления размещены на поверхности с небольшими перерывами, а их последовательность образует дорожку спиральной формы.

Размеры углублений имеют порядок одного микрона ($1 \text{ мкм} = 10^{-3} \text{ мм}$), расстояние между витками спирали 1,6 мкм, длина спирали 5300 м. Диск содержит 20 000 дорожек (витков), на которых находятся около $6 \cdot 10^9$ углублений. Емкость памяти диска равна 640 Мегабайтам.

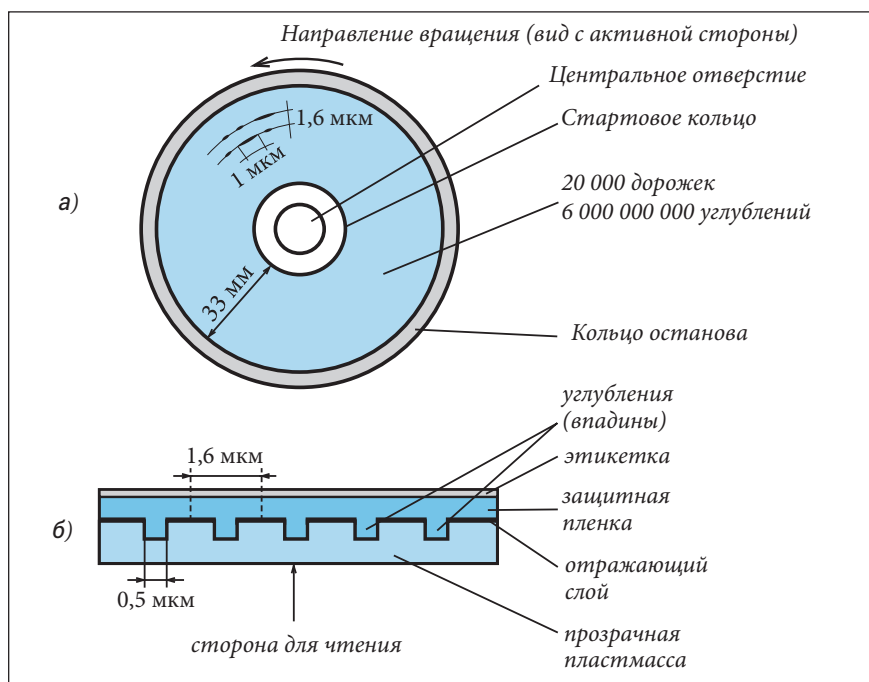


Рис. 6.9. Структура оптического диска CD-ROM:
а — расположение дорожек на диске; б — вид на диск в разрезе
(перпендикулярно дорожкам)

Чтение оптического диска осуществляется с помощью лазерного луча, который, отразившись от активной поверхности, попадает на фоточувствительную ячейку (рис. 6.10).

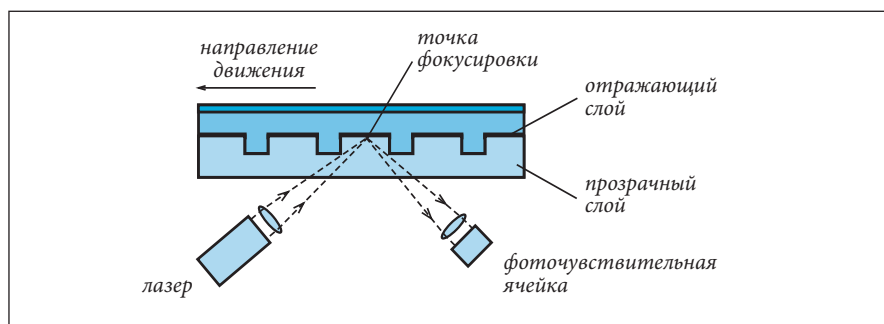


Рис. 6.10. Чтение оптических дисков

Проходя вдоль соответствующих дорожек, лазерный луч отражается, когда свет попадает в **точку фокусировки** и рассеивается в противном случае. Другими словами, углубления на рабочей поверхности оптического диска изменяют (модулируют) интенсивность отраженного пучка. Вследствие этого на выходе фотоячейки формируется сигнал, который воспроизводит последовательности двоичных цифр 0, 1, записанных на диск на этапе его создания.

В современных устройствах памяти на оптических дисках источник света — лазер и фоточувствительная ячейка — фотодиод выполняются в едином блоке, именуемом **оптической головкой считывания**. Угловая скорость диска составляет 200–600 оборотов в минуту, а линейная скорость — 1,4 м/с. Несмотря на очень большие скорости, оптический диск практически не изнашивается, так как между оптической головкой и диском отсутствует прямой механический контакт. Вследствие этого длительность эксплуатации диска *CD-ROM* определяется качеством отражающего и защитного слоев. При использовании алюминия из-за окисления отражающий слой темнеет, ограничивая срок эксплуатации такого диска 10–15 годами. При использовании отражающего слоя из золота только преднамеренные механические воздействия, разрушающие защитный слой, могут повлиять на качество оптического диска.

Как правило, оптические диски *CD-ROM* используются для тиражирования операционных систем, трансляторов, энциклопедий, электронных игр, а также другой информации, предназначенной для очень большого числа пользователей.

Структура *CD-R* и *CD-RW* оптических дисков приведена на *рис. 6.11*.

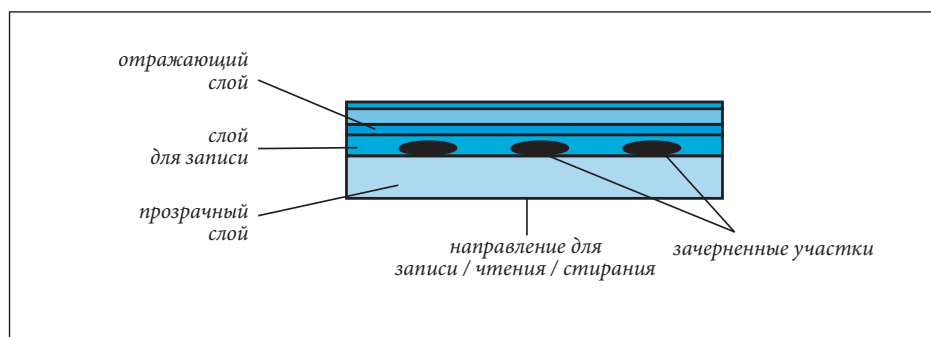


Рис. 6.11. Структура *CD-R* и *CD-RW* оптических дисков

На *рис. 6.11* видим, что рассматриваемые диски содержат особый слой, называемый **слоем для записи**. В случае ***CD-R* дисков** данный слой состоит из специального органического материала (цианина или фталоцианина), который темнеет при нагревании. Запись информации осуществляется с помощью мощного лазерного луча, вызывающего потемнение соответствующих участков слоя для записи.

При чтении затемненные области блокируют прохождение лазерного луча к отражающему слою, изменяя, таким образом, интенсивность света, попадающего на светочувствительную ячейку (*рис. 6.10*). Поскольку лазерный луч, используемый для чтения, является слабым, ранее записанный диск можно читать многократно без разрушения хранимой на нем информации.

В случае ***CD-RW* дисков** материал слоя для записи подбирается так, чтобы он вновь становился прозрачным при нагреве до особой температуры, называемой **критической температурой**. Очевидно, что после стирания на перезаписываемый диск можно записывать новые двоичные данные. Современные *CD-RW* диски выдерживают до 10 000 циклов записи/стирания.

Общим недостатком записываемых и перезаписываемых оптических дисков является их повышенная чувствительность к температуре. Продолжительность их эксплуатации меньше, чем *CD-ROM* дисков. Такие диски дороже, поскольку их отражающий слой выполнен из серебра или золота.

Обычно *CD-R* и *CD-RW* диски используются для распространения оперативной информации, предназначенной для определенного круга пользователей — баз данных, специализированных пакетов программ, отчетов особо важных симпозиумов, картин со знаменитых выставок, мультимедийных документов и т. п.

Отметим, что в результате непрерывного развития компьютерных технологий в последние годы широкое распространение получили *DVD* оптические диски (*Digital Versatile Disc* — Цифровой Многоцелевой Диск). Емкость *DVD* диска примерно в семь раз больше, чем емкость *CD* диска. Позже были разработаны еще более новые диски, такие как *HVD* (*Holographic Versatile Disc* — Многофункциональный Голографический Диск). Емкость такого диска в тысячи раз больше, чем у компакт-диска.

В настоящее время в персональных компьютерах общего назначения оптические, а иногда и жесткие диски заменяются *флэш*-памятью. В этих запоминающих устройствах информация хранится с помощью электронных схем, которые обеспечивают запись, чтение и удаление данных на гораздо более высоких скоростях.

Вопросы и упражнения

- ❶ Как представляются двоичные цифры 0, 1 при оптической записи?
- ❷ Укажите емкость памяти оптического диска.
- ❸ Как записывается информация на оптический диск *CD-ROM*?
- ❹ От чего зависит емкость памяти оптического диска?
- ❺ Известно, что длина спирали, вдоль которой записывается информация оптического диска, равна 5300 м. Емкость памяти диска составляет 640 *Мбайт*. Определите плотность записи информации на оптическом диске.
- ❻ В каких областях применяются *CD-ROM* диски?
- ❼ От чего зависит время доступа к информации на оптическом диске?
- ❽ Линейная скорость оптического диска составляет 1,4 м/с. Зная плотность записи информации $d = 128$ *Кбайтов/м*, определите **скорость передачи данных** от дискового устройства к центральному устройству. Напомним, что скорость передачи данных измеряется в *битах*, *Кбитах* или *Мбитах* в секунду.
- ❾ На 20 000 дорожках (витках) оптического диска записано около 640 *Мбайтов* информации. Сколько информации содержит одна дорожка оптического диска?
- ❿ На одном *CD-ROM* диске записано в двоичном коде около 74 минут музыки. Определите, сколько информации содержит одна песня длительностью 4 минуты 30 секунд. Сколько дорожек занимает соответствующая песня?
- ⓫ Как считывается информация с оптического диска? В чем назначение оптической головки считывания?
- ⓫ Объясните назначение слоев записываемого оптического диска. Как осуществляется запись информации на рассматриваемый диск?
- ⓫ От чего зависит длительность эксплуатации *CD-ROM* диска? Дисков *CD-R* и *CD-RW*?
- ⓫ Как записывается и стирается информация на перезаписываемом оптическом диске?
- ⓫ В каких областях применяют *CD-R* и *CD-RW* диски?

- 16 ЭКСПЕРИМЕНТИРУЙТЕ! Найдите технические параметры дисковых оптических устройств, которые, возможно, были установлены на компьютерах в кабинете информатики вашего лица.
- 17 ТВОРИТЕ! Пользуясь источниками информации из Интернета, напишите краткое эссе об эволюции оптических дисков.

6.8. Видеомонитор и клавиатура

Видеомонитор — это устройство вывода, с помощью которого информация отображается на экране.

В прошлом у первых видеомониторов информация отображалась на экране электронно-лучевой трубки. Функциональная схема такого видеомонитора представлена на *рис. 6.12*.

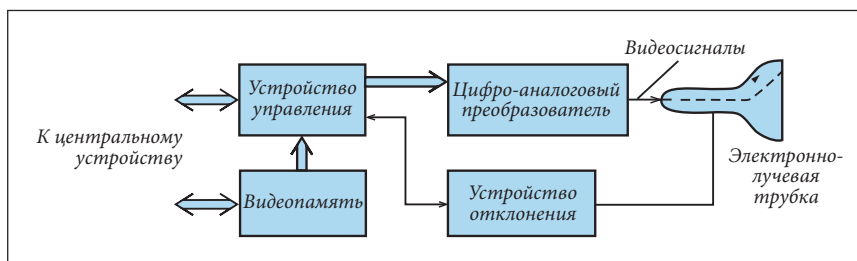


Рис. 6.12. Функциональная схема видеомонитора с электронно-лучевой трубкой

Как и в случае обычных телевизоров, изображение на экране электронно-лучевой трубки формируется из точек. Точки светятся под воздействием электронного луча. Цвет и яркость каждой точки задаются видеосигналами, поданными на соответствующие входы электронно-лучевой трубки. Обход точек в заранее установленном порядке, как правило, по строкам слева направо и сверху вниз осуществляется с помощью устройства отклонения электронного луча.

Каждой точке на экране соответствует одна ячейка в памяти видеомонитора. Ячейки видеопамяти содержат информацию относительно цвета и яркости соответствующих точек на экране электронно-лучевой трубки. Управляющее устройство читает ячейки видеопамяти в порядке обхода точек на экране. Содержимое каждой ячейки интерпретируется как команда, задающая цвет и яркость соответствующей точки. Видеосигналы, необходимые для управления электронным лучом, формируются цифро-аналоговыми преобразователями.

Данные в видеопамяти монитора записываются центральным устройством компьютера. В любой момент времени компьютер может изменить содержимое видеопамяти. Как следствие, изменяется и изображение на экране видеомонитора. Изображения можно сделать движущимися путем изменения содержимого видеопамяти с частотой, используемой в кинематографии.

В общем случае видеомонитор может функционировать в одном из двух режимов: текстовом или графическом.

В текстовом режиме экран делится на условные зоны, каждая из которых называется **знакоместом**. Как правило, эти зоны образуют 25 строк по 80 символов в каждой. На каждом знакоместе может быть отображен любой символ из

общего набора в 256 символов. Набор символов состоит из прописных и строчных букв латинского алфавита, десятичных цифр, математических символов, знаков пунктуации, букв национальных алфавитов и некоторых псевдографических символов, используемых для вывода на экран таблиц, диаграмм, рамок и т. п. Каждое знакоместо может иметь отдельные цвета для символа и фона, на котором он изображен. Это позволяет выводить на экран текст с разноцветными буквами.

В графическом режиме пользователь может управлять выводом на экран каждой точки в отдельности. Количество точек по горизонтали и вертикали определяет разрешение монитора. Например, выражение «разрешение 640×200» означает, что видеомонитор отображает 640 точек по горизонтали и 200 — по вертикали.

Существует множество международных стандартов, которые регламентируют разрешение и количество цветов видеомониторов. Эти характеристики являются общепринятыми и соблюдаются фирмами-производителями.

Например, в случае персональных компьютеров самые распространенные стандарты — это *EGA*, *VGA* и *SVGA*. Стандарт *EGA* устанавливает, что видеомонитор имеет разрешение 640×350 точек, допуская использование 64 цветов.

Стандарт *VGA*, сохраняя совместимость с *EGA*, предлагает как дополнительную характеристику разрешение 640×480 точек при 256 различных цветов.

Для улучшения качества изображения стандарт *SVGA* дополнен разрешением 1024×768 точек.

Отметим, что технические достижения последних лет сделали возможной замену электронно-лучевых трубок на плоские экраны. Плоский экран состоит из матрицы (массива) светоизлучающих ячеек, по одной ячейке для каждой из микрозон цифрового изображения, предоставляемого компьютером. Использование светоизлучающих ячеек привело к существенному уменьшению габаритов мониторов и значительному улучшению качества изображений. Такие мониторы характеризуются уменьшенным энергопотреблением и разрешением до 2560×2048 точек.

Клавиатура — это устройство ввода, которое преобразует нажатие клавиш в двоичные слова, воспринимаемые компьютером.

Электронная часть любой клавиатуры состоит из шифратора. На входы шифратора подаются логические сигналы, формируемые при нажатии клавиш. На выходе появляется слово из определенного двоичного кода, стандартного для каждого семейства компьютеров (*ISO*, *ACSI*, *UNICODE* и т. п.). Некоторые виды клавиатур снабжаются аудиогенератором, который при нажатии клавиш издает специфический звук.

У нас в Молдове, как и в англоязычных странах, самой распространенной является клавиатура типа (с раскладкой) *QWERTY*, название которой происходит от расположения символов *Q*, *W*, *E*, *R*, *T* и *Y* в верхнем ряду алфавитно-цифровых клавиш. Во франкоговорящих странах используется клавиатура *AZERTY*, в Германии — *QWERTZ* и т. п.

Несмотря на то что раскладка клавиш и их количество могут отличаться, назначение основных клавиш на всех клавиатурах одинаково.

Клавиши делятся на следующие группы: *алфавитно-цифровые*, *функциональные* и *специальные*. Группа алфавитно-цифровых клавиш включает клавиши десятичных цифр, клавиши символов английского и русского алфавитов, клавиши математических символов и знаков пунктуации. Группа функциональных клавиш включает клавиши *<F1>*, *<F2>*, ..., *<F12>*. Данные

клавиши не имеют predetermined назначения и их использование зависит от программы, выполняемой на компьютере. Специальные клавиши применяют для управления положением курсора, для ввода в компьютер двоичных слов, не имеющих отдельных клавиш, и т. п.

В мощных компьютерах видеомонитор и клавиатура могут образовывать единое устройство, именуемое **консолью**. Консоль, применяемая для управления вычислительной системой, называется **монитором**.

В случае планшетных компьютеров и компьютеров, встроенных, например в смартфоны, применяют с цифровым управлением в бытовой технике и др., в качестве устройств ввода-вывода используются **сенсорные экраны** (*touchscreen*), которые позволяют отображать визуальную информацию, а также вводить команды и данные путем перетаскивания и касания к отображаемым на них элементам управления.

Вопросы и упражнения

- 1 Объясните, как работает видеомонитор. Назовите назначение основных частей видеомонитора.
- 2 Как можно изменить изображение, выводимое на экран видеомонитора? Как «оживить» изображение на экране?
- 3 Чем отличаются режимы работы видеомонитора?
- 4 Перечислите основные показатели качества видеомонитора.
- 5 ЭКСПЕРИМЕНТИРУЙТЕ! Определите тип видеомонитора, на котором вы работаете. Определите разрешение экрана и количество доступных цветов.
- 6 Укажите составные части клавиатуры. Как определить тип клавиатуры?
- 7 Назовите группы клавиш и их назначение.
- 8 ИССЛЕДУЙТЕ! Пользуясь информацией из Интернета, проведите сравнительное исследование видеомониторов с электронно-лучевыми трубками и с плоским экраном, выделите как преимущества, так и недостатки каждого из них. Укажите области применения каждого типа видеомониторов.
- 9 ТЕМАТИЧЕСКОЕ ИССЛЕДОВАНИЕ. С помощью поисковой системы найдите в Интернете описание видеомониторов, предлагаемых торговыми предприятиями. Узнайте, как технические параметры видеомониторов влияют на их цену, каковы преимущества, недостатки и области их применения.
- 10 ТВОРИТЕ! Используя Интернет в качестве источника информации, напишите небольшое эссе об эволюции видеомониторов как промышленных, так и для персонального пользования.

6.9. Принтеры

Принтеры — это устройства вывода, которые предоставляют результаты в виде отпечатанного документа. В зависимости от используемого **принципа печати** принтеры классифицируются на:

- механические принтеры, в которых печать осуществляется с помощью молоточков или игловок;
- лазерные принтеры, в которых печать осуществляется с использованием

электростатических методов, как в копировальных аппаратах;

– струйные принтеры;

– термопринтеры, работа которых основывается на применении специальной бумаги, меняющей цвет при нагревании.

Принцип работы **матричного игольчатого принтера** представлен на рис. 6.13.

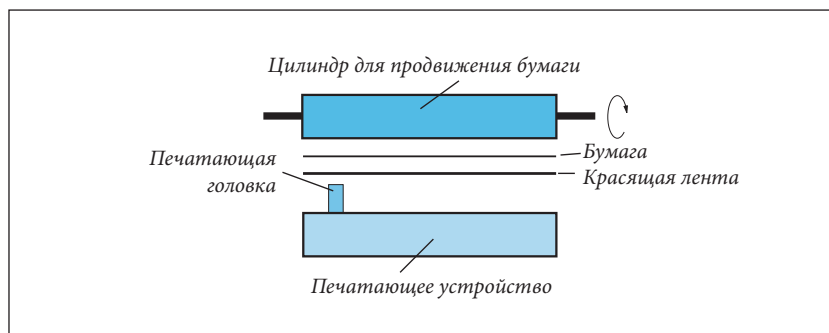


Рис. 6.13. Принцип работы матричного игольчатого принтера

Печатающая головка содержит группу тонких металлических иглол, которые в нужный момент ударяют через красящую ленту по бумаге. Конфигурация ударяющих игл в каждый из моментов времени и продвижение головки вдоль линии печати определяют печатаемые изображения.

Матричные принтеры могут работать как в графическом, так и в алфавитно-цифровом (текстовом) режиме.

В **графическом режиме** компьютер управляет печатью каждой точки в отдельности. Ясно, что из точек могут быть сформированы любые изображения: графики, эскизы, а также символы, разработанные пользователем. Печать информации в графическом режиме является очень медленной из-за большого количества перемещений печатающей головки и пошаговой подачи бумаги.

В **алфавитно-цифровом режиме** компьютер посылает печатающему устройству только коды печатаемых символов. Каждому коду соответствует изображение из точек, хранящееся в специальной памяти принтера. Соответствующие изображения печатаются за один или, самое большее, два-три прохода печатающей головки.

Печать в алфавитно-цифровом режиме выполняется быстрее, однако можно печатать только те символы, изображения которых записаны в памяти принтера. У простых матричных принтеров есть несколько стандартных наборов символов, записанных в постоянную память. Более производительные матричные принтеры обеспечивают возможность программной загрузки множества наборов символов, конфигурация которых задана пользователем.

Отметим, что чем больше количество игл в печатающей головке, тем выше качество печати. В настоящее время применяют принтеры с 9 или с 24 иглами. Качество печати можно улучшить с помощью повторной печати (2–4 раза) одного и того же символа на том же месте. Скорость печати механических игольчатых принтеров составляет 150–500 символов в минуту.

Принцип работы **лазерных принтеров** представлен на рис. 6.14.

Главным элементом такого принтера является барабан, покрытый полу-

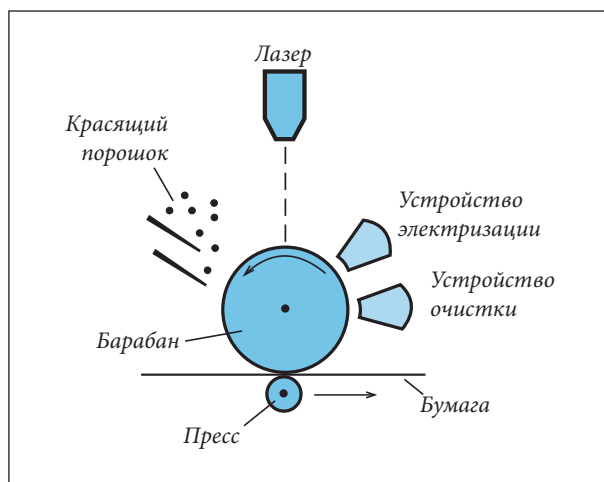


Рис. 6.14. Принцип действия лазерного принтера

проводящим слоем, который изменяет свои электрические свойства под воздействием света.

При печати текущей страницы сперва электризуется поверхность барабана. Далее с помощью лазерного луча на заряженную поверхность барабана проецируются точки печатаемого изображения. Поскольку освещенные участки изменяют свою электрическую проводимость, соответствующие заряды нейтрализуются. Следовательно, на поверхности барабана формируется невидимое электрическое изображение. Проявление изображения осуществляется с помощью очень мелких частиц красящего порошка, притягиваемых заряженными участками барабана. Изображение с барабана переносится на бумагу и закрепляется путем нагрева.

Далее все электрические заряды с поверхности барабана нейтрализуются, а остатки порошка удаляются.

Лазерные принтеры обладают самыми лучшими характеристиками среди современных принтеров. Тексты и графика, отпечатанные с помощью таких принтеров, не отличаются от типографских. Скорость печати составляет 5–15 страниц в минуту.

Струйные чернильные принтеры формируют точки печатаемого изображения из микроскопических капель, наносимых на бумагу через специальные сопла. Они обеспечивают очень хорошее качество, их применяют для цветной печати. Данные принтеры дороже, чем механические, и требуют особого технического ухода.

Термопринтеры используют матрицу из игл, селективный кратковременный нагрев которых осуществляется в зависимости от того, какой символ должен быть напечатан. Скорость печати у них относительно низка, около 300 строк в минуту, их основное преимущество состоит в малых габаритах.

Вопросы и упражнения

- ❶ Как классифицируются принтеры в зависимости от принципа печати?
- ❷ Какие узлы входят в состав любого принтера?

- ③ В чем принцип действия матричного игольчатого принтера? Как работает такой принтер?
- ④ Объясните, как работает лазерный принтер. В чем главное преимущество лазерного принтера?
- ⑤ ЭКСПЕРИМЕНТИРУЙТЕ! Определите тип принтера, которым пользуетесь вы. Найдите технические параметры принтера: набор символов, режимы функционирования, емкость буферной памяти, скорость печати.
- ⑥ ТЕМАТИЧЕСКОЕ ИССЛЕДОВАНИЕ! С помощью поисковой системы найдите в Интернете описания принтеров, предлагаемых торговыми предприятиями. Узнайте, как технические параметры принтеров влияют на их цену, каковы преимущества, недостатки и области их применения.
- ⑦ ТВОРИТЕ! Используя Интернет в качестве источника информации, напишите небольшое эссе об эволюции принтеров как промышленных, так и для персонального пользования.

6.10. Классификация компьютеров

Характеристика любого компьютера включает следующие показатели:

- скорость выполнения операций;
- емкость внутренней памяти;
- состав, емкость и время доступа устройств внешней памяти;
- состав и соответствующие технические параметры периферийного оборудования;
- масса и габариты;
- стоимость.

В зависимости от этих параметров современные компьютеры делятся на 4 категории:

- суперкомпьютеры;
- большие компьютеры;
- мини-компьютеры;
- микрокомпьютеры (персональные компьютеры).

Суперкомпьютеры могут выполнять около 10^{15} (1000 триллионов) операций в секунду, а их цена превышает 20 миллионов долларов. Исследовательские и конструкторские разработки в области суперкомпьютеров осуществляют в США и Японии фирмами *IBM*, *Gray Research*, *Fujitsu*, *ETA Systems*, *Dell*, *Sutherland* и др. Суперкомпьютеры применяют в чрезвычайно сложных системах обработки данных в авионавтике, ядерной физике, астронавтике, сейсмологии, при прогнозировании погоды и т. п.

Большие компьютеры (также **мэйнфрейм**, от английского *mainframe* — «основной шкаф») могут выполнять сотни триллионов операций в секунду, их цена составляет от 20 тысяч до нескольких миллионов долларов. Как правило, большие компьютеры включают в свой состав десятки устройств на магнитных дисках, десятки принтеров, сотни консолей, находящихся на различных расстояниях от центрального устройства. Данные компьютеры используются в больших вычислительных центрах и работают в круглосуточном режиме. Главные фирмы-производители больших компьютеров — *IBM*, *Hitachi*, *Amdahl*, *Fujitsu* и др.

Мини-компьютеры выполняли десятки и сотни миллионов операций в секунду, а их цена не превышала 200–300 тысяч долларов. Периферийное оборудование одного мини-компьютера включало в себя несколько магнитных дисков, один или два принтера, множество консолей. Мини-компьютеры были более простыми в обращении, чем большие компьютеры. Их применяли в системах автоматизированного проектирования, в промышленной автоматике, для обработки данных в научных экспериментах и т. п. Среди фирм, которые производили мини-компьютеры, отметим такие, как *IBM, Wang, Texas Instruments, Data General, DEC, Hewlett-Packard* и т. п. В настоящее время мини-компьютеры полностью вытеснены персональными компьютерами.

Микрокомпьютеры, называемые также **персональными компьютерами**, продают по относительно низким ценам — между 100 и 15 000 долларов и обеспечивают скорость вычислений порядка миллиарда операций в секунду. Блок-схема персонального компьютера представлена на *рис. 6.2* (с. 164).

Как правило, периферийное оборудование персонального компьютера включает в себя одно или несколько устройств на жестком диске или SSD (*Solid-State Drive*), устройство на оптических дисках, принтер и консоль. Модульная конструкция и группирование всего оборудования вокруг одной магистрали для передачи данных обеспечивают возможность реконфигурации микрокомпьютера в зависимости от индивидуальных потребностей каждого пользователя.

Для этого микрокомпьютеры имеют несколько портов USB (*Universal Serial Bus* — универсальная последовательная шина), которые позволяют подключать различные дополнительные периферийные устройства, включая *флэш*-память. Также персональные компьютеры, предназначенные для размещения на столе (*desktop*), портативные компьютеры (называемые *ноутбуками* или *лэптопами*) и планшетные компьютеры оснащены периферийными устройствами, обеспечивающими связь с другими цифровыми техническими системами через радиоволны (*Wi-Fi, Bluetooth*) или инфракрасные лучи.

Корпорации, производящие микрокомпьютеры, находятся в очень многих странах. Современные всемирно признанные лидеры — это фирмы *Lenovo, Apple, Hewlett-Packard, Dell, Asus, Acer, Samsung* и др.

Вопросы и упражнения

- 1 ЭКСПЕРИМЕНТИРУЙТЕ! Назовите основные параметры, характеризующие современный компьютер. Определите технические и экономические параметры компьютера, на котором вы работаете.
- 2 Как классифицируются компьютеры в зависимости от их технических и экономических параметров?
- 3 ТВОРИТЕ! На основе информации из Интернета, дайте краткую характеристику каждой категории компьютеров: суперкомпьютеров, больших компьютеров, мини-компьютеров и микрокомпьютеров. Выделите области использования компьютеров из каждой категории.
- 4 ИССЛЕДУЙТЕ! Используя один из поисковых серверов, найдите в Интернете подробную информацию об основных производителях персональных компьютеров. Составьте рейтинг этих производителей в соответствии с их долевым участием на рынке.

- ❖ **ПРОАНАЛИЗИРУЙТЕ!** Определите процент прибыли крупных компаний, производящих компьютеры, по категориям (суперкомпьютеры, большие компьютеры, макрокомпьютеры, мини-компьютеры, микрокомпьютеры), и проведите небольшое исследование, которое отразит эволюцию этого важного сектора мировой экономики.
- ❖ **ИССЛЕДУЙТЕ!** Используя данные, опубликованные Национальным статистическим бюро и Национальным агентством по регулированию в области электронных коммуникаций и информационных технологий (ANRCETI), проведите небольшое исследование о производстве и продаже компьютеров в нашей стране.
- ❖ **ИССЛЕДУЙТЕ!** Международная группа ИТ-специалистов реализует так называемый проект «Топ 500». Этот проект, запущенный в 1993 году, дважды в год публикует список 500 самых мощных суперкомпьютеров. Проведите небольшое исследование первых десяти суперкомпьютеров, выделяя технические параметры, их назначение, компании, которые их производят, а также роль международного сотрудничества в развитии информатики и информационных технологий.

6.11. Микропроцессор

Микропроцессор — это интегральная схема, которая выполняет функции центрального устройства обработки информации — выборки и исполнения команд.

Как правило, микропроцессор содержит арифметическое и управляющее устройства, группу регистров, предназначенных для временного хранения часто используемых данных, магистрали и соответствующие схемы управления (рис. 6.15).

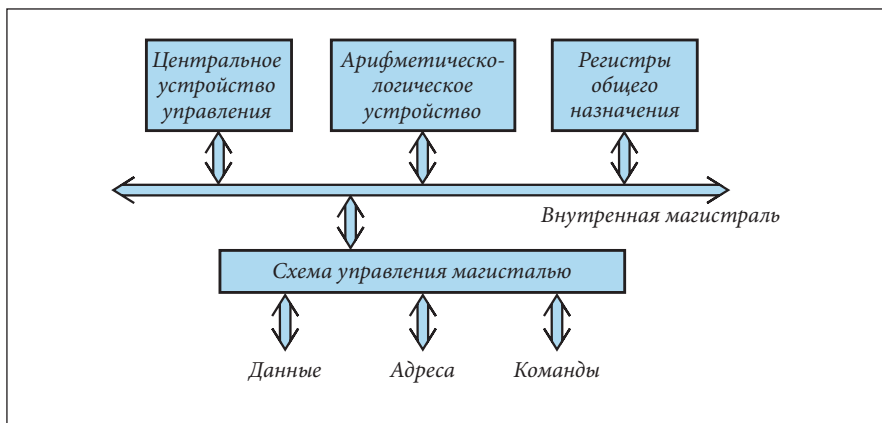


Рис. 6.15. Функциональная схема микропроцессора

Микропроцессор взаимодействует с устройствами памяти и периферийными устройствами с помощью трех магистралей: *Данные*, *Адреса* и *Команды*.

Поток информации через магистрали контролируется *Схемой управления магистралью*.

Выборка и выполнение команд происходят под управлением **центрального устройства управления**. Для этого с помощью магистрали адреса указывается адрес ячейки внутренней памяти, а с помощью магистрали команд передаются сигналы для записи или чтения. Данные для чтения или записи передаются по магистрали данных.

Микропроцессоры характеризуются следующими параметрами:

- длина слова;
- частота системных часов;
- емкость магистралей.

Длина слова представляет количество битов двоичной последовательности, которая может быть записана в регистрах и обработана арифметическим устройством микропроцессора. Большинство микропроцессоров имеют длину слова 32, 64 и более бита.

Частота системных часов представляет собой количество импульсов в секунду, вырабатываемых генератором тактовых импульсов, входящим в состав центрального устройства управления. Частота системных часов измеряется в **Мегагерцах** ($1 \text{ МГц} = 10^6 \text{ Гц}$). Поскольку тактовые импульсы синхронизируют выполнение микроопераций во всех устройствах микропроцессора, частота системных часов характеризует быстроту микропроцессора.

Емкость магистрали представляет собой количество разрядов двоичных слов, передаваемых по магистрали. Желательно, чтобы емкость магистрали данных была равна или больше длины слова микропроцессора. В противном случае для того, чтобы передать одно слово, необходимо несколько циклов магистрали данных.

Емкость магистрали адреса определяет пространство адресов, которое напрямую доступно микропроцессору. Так, микропроцессор с емкостью магистрали адреса в 16 бит имеет доступ к 216 ячейкам внутренней памяти, а микропроцессор с емкостью магистрали адресов в 32 бита может адресовать напрямую 2^{32} ячеек.

Длина слова, частота системных часов и емкость магистралей определяют **производительность микропроцессора**, которая измеряется в **Mips** — миллионов команд в секунду. Для примера в *таблице 6.2* представлены главные характеристики микропроцессоров из семейства *Intel*.

Таблица 6.2

Главные характеристики
микропроцессоров из семейства *Intel*

Микропроцессор	Длина слова, бит	Частота, МГц	Производительность, Mips
4040 (1974)	4	0,7	0,06
8085 (1976)	8	5	1,25
80286 (1982)	16	16	8
Pentium 4 (2004)	32	3800	2500
Pentium D (2005)	64	3400	4000
Intel Core i9 (2017)	64	4500	5500

Из приведенной таблицы видно, что за 1974–2017 годы длина слов, с которыми работает микропроцессор, увеличилась в 8 раз, частота системных часов примерно в 6 500 раз, а их производительность примерно в 9 200 раз. Производительность современного микропроцессора примерно в 6 000 раз больше, чем компьютера *Лунного Модуля Apollo*, с которым в 1969 году первые люди Земли высадились на Луну.

Тип микропроцессора и частоту системных часов персонального компьютера, с которым вы работаете, можно узнать с помощью контекстного меню пиктограммы „My computer”. Напоминаем, что контекстное меню выводится на экран с помощью щелчка правой кнопкой мыши по соответствующей пиктограмме.

Вопросы и упражнения

- ❶ Объясните назначение устройств, входящих в состав микропроцессора (рис. 6.15).
- ❷ Назовите главные параметры микропроцессора. Объясните смысл каждого параметра.
- ❸ ЭКСПЕРИМЕНТИРУЙТЕ! Определите тип и главные параметры микропроцессора из состава компьютера, на котором вы работаете.
- ❹ ИССЛЕДУЙТЕ! Помимо *Intel*, в современных компьютерах используют микропроцессоры других компаний. Анализируя информацию о персональных компьютерах, продаваемых в нашей стране, выявите производителей, входящих в их состав микропроцессоров. Обратите внимание на то, как цена персональных компьютеров варьирует в зависимости от используемых микропроцессоров.
- ❺ ТВОРИТЕ! Разработайте небольшое исследование микропроцессоров для настольных ПК, ноутбуков, планшетов и для смартфонов, широко используемых в нашей стране.
- ❻ ИССЛЕДУЙТЕ! Сформулированный в 1965 году «закон» Мура, основателя *Intel*, описывает долгосрочную тенденцию в истории компьютеров: количество транзисторов, которые можно разместить на интегральной схеме, удваивается примерно каждые два года. Эта тенденция продолжается более полувека. Анализируя данные о количестве транзисторов в интегральных схемах, опубликованные в Интернете, выясните, правильно ли этот «закон» описывает эволюцию микропроцессоров в последние годы.

7.1. Введение в компьютерные сети

Одновременно с расширением области использования компьютеров выросло и количество пользователей, желающих иметь доступ к средствам для эффективной обработки и хранения совместно используемой информации.

Например, при проектировании нового здания большое число специалистов — архитектор, инженер, пожарный и др. — хотят одновременно получать доступ и вносить, если необходимо, изменения в строительные чертежи, находящиеся в процессе разработки. Авиакомпании могут продавать билеты на один и тот же рейс в агентствах, находящихся в разных городах.

Простейшее решение данной задачи состоит в подключении к центральному компьютеру большой мощности множества терминалов (рис. 7.1).

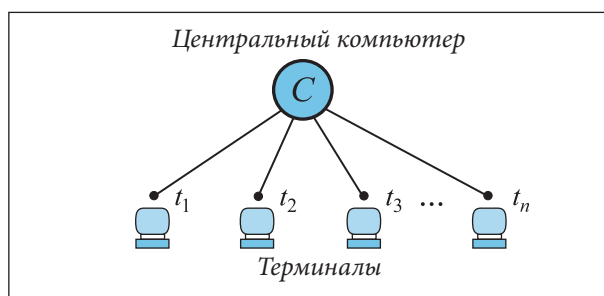


Рис. 7.1. Централизованная вычислительная система

Как правило, терминал состоит из монитора, клавиатуры и, если необходимо, принтера. Главным недостатком централизованной системы обработки данных является низкая надежность и неэффективное использование вычислительных ресурсов.

Со временем появилась тенденция к переходу от централизованных систем к размещению компьютеров у каждого пользователя и обеспечению их эффективного взаимодействия с помощью специальных соединений (рис. 7.2).

Компьютерной сетью называется множество компьютеров, которые могут обмениваться информацией с помощью системы связи.

Компьютеры любой сети подключаются к системе связи с помощью специально предназначенных блоков ввода-вывода, называемых **сетевыми адаптерами**.

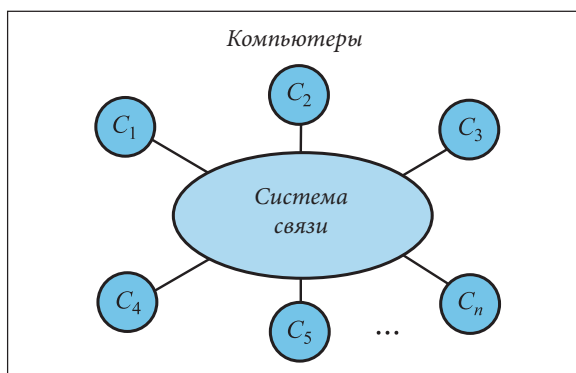


Рис. 7.2. Компьютерная сеть

Естественно, что в пределах одной определенной сети каждый компьютер, точнее каждый сетевой адаптер, имеет уникальный адрес, называемый **сетевым адресом**.

Например, компьютерная сеть может быть построена с использованием в качестве системы связи существующей телефонной сети. В таком случае сетевой адаптер включает в себя модулятор для преобразования цифровых сигналов компьютера в телефонные сигналы и демодулятор для обратного преобразования. Соответствующее устройство ввода-вывода носит название **модем (модулятор-демодулятор)**. Сетевой адрес задается номером телефона, к которому подключен модем.

Главным образом система связи состоит из **линий передачи** сигналов. Этими линиями могут быть:

- кабели на основе витых проводов («витая пара»);
- коаксиальные кабели;
- оптические кабели;
- микроволновые радиоканалы (наземные и спутниковые).

Кабели на основе витых пар аналогичны телефонным и обеспечивают скорость передачи до 1 Мбит/с . **Коаксиальные кабели** похожи на те, которые применяют в телевизионных кабельных сетях. Они обеспечивают скорость передачи до 1 Гбит/с . **Оптический кабель** состоит из стеклянного или прозрачного пластмассового волокна, покрытого защитной оболочкой. Оптический сигнал от лазера-источника распространяется по волокну и принимается фоточувствительной ячейкой. Скорость передачи информации по оптическому кабелю может достигать 1 Тбит/с .

Микроволновые радиоканалы состоят из ретрансляционных станций, которые обеспечивают прием и передачу сигналов на сантиметровых волнах. На Земле приемопередающие станции расположены в радиусе прямой видимости антенн, на расстоянии 40–50 км друг от друга. В случае космических линий связи соответствующие станции размещены на спутниках. Скорость передачи микроволновых радиоканалов составляет около 10 Гбит/с .

В зависимости от площади охвата различают следующие **типы сетей**:

- локальные сети;
- региональные сети;
- глобальные сети.

В локальных сетях компьютеры охватывают незначительную по площади зону (диаметром до 2 км) и обслуживают одну организацию. Как правило, локальные сети состоят из компьютеров, находящихся в пределах одного или нескольких зданий. Обычно линии передачи обеспечивают в данном случае кабели на основе витой пары или коаксиальные кабели.

В последние годы для маленьких и очень маленьких локальных сетей все большее распространение получили средства, обеспечивающие связь цифрового оборудования с помощью радиоволн (*Wi-Fi, Bluetooth*) или инфракрасных лучей.

Региональные сети покрывают площадь одного города или района. Линии связи реализуются на основе коаксиальных кабелей или маленьких приемопередающих станций, называемых **радиомодемами**.

Глобальные сети охватывают территории стран и континентов. В качестве линий передачи служат оптические или микроволновые радиоканалы (наземные и спутниковые).

Главное преимущество сетей состоит в возможности **обмениваться** или, другими словами, **совместном использовании** данных, программ и компьютеров каждой сети.

Например, локальные сети позволяют совместно пользоваться файлами, дисками, принтерами, сканерами, другим периферийным оборудованием. Возможность одновременного доступа нескольким пользователям к устройствам и оборудованию повышает эффективность его использования. Таким образом, сотрудники одной организации могут работать совместно над общими проектами: годовым бюджетом, планом продаж, обновлением баз данных и т. п.

Говоря о глобальных сетях, коллективы исследователей могут осуществлять сложные расчеты на уникальном суперкомпьютере или совместно анализировать результаты особо дорогостоящих научных экспериментов. На основе существующих сетей создаются различные службы: передача файлов, электронная почта, передача новостей, группы общения по интересам, электронные игры, реклама, денежные переводы, банковские операции и др.

Вопросы и упражнения

- ❶ Назовите причины, которые привели к появлению компьютерных сетей.
- ❷ Укажите недостатки централизованных вычислительных сетей.
- ❸ Назовите главные компоненты компьютерной сети.
- ❹ Объясните назначение системы связи из состава компьютерной сети.
- ❺ Какие функции выполняет сетевой адаптер? Как идентифицируются компьютеры, входящие в состав сети? Определите тип сетевого адаптера, с которым вы работаете.
- ❻ Из чего состоит система связи?
- ❼ Для чего предназначен модем? Для чего предназначен радиомодем?
- ❽ Назовите скорость передачи данных по следующим линиям связи:
 - кабель на основе витой пары;
 - коаксиальный кабель;
 - оптический кабель;
 - микроволновой радиоканал.

- 9 Оцените время передачи одного видеофильма ($\approx 800 \text{ Гбит}$) по известным вам линиям связи.
- 10 ЭКСПЕРИМЕНТИРУЙТЕ! Определите тип линий связи компьютерной сети, в которой вы работаете.
- 11 Как классифицируются компьютерные сети в зависимости от площади охвата?
- 12 ЭКСПЕРИМЕНТИРУЙТЕ! Определите тип компьютерной сети (локальная, региональная или глобальная), в которой вы работаете.
- 13 В чем преимущества компьютерных сетей? Какие услуги предлагает компьютерная сеть?
- 14 ИССЛЕДУЙТЕ! Узнайте скорость передачи для следующих коммуникационных средств компьютерных сетей:
- *Wi-Fi*;
 - *Bluetooth*;
 - с инфракрасными лучами.

7.2. Технологии взаимодействия в компьютерной сети

Ресурсами компьютерной сети являются: периферийное оборудование, линии связи, сами компьютеры, файлы, базы данных, исполняемые программы и т. п. Эффективное использование перечисленных ресурсов предполагает совместную работу или, другими словами, кооперацию компьютеров и программ, которые на них выполняют.

Технологией взаимодействия называется способ совместной работы компьютеров и программ в сети.

Наиболее часто в компьютерных сетях используется технология клиент-сервер и равный-с-равным.

В технологии **клиент-сервер** общим ресурсом (например, цветным принтером или диском большой емкости) управляет специально выделенный компьютер — **сервер**. Компьютер, цель которого получить доступ к данным ресурсам, называется **клиентом**. Для использования соответствующего ресурса клиент посылает серверу запрос. Сервер анализирует принятые запросы и, в зависимости от статуса каждого клиента, принимает или отвергает их (рис. 7.3).

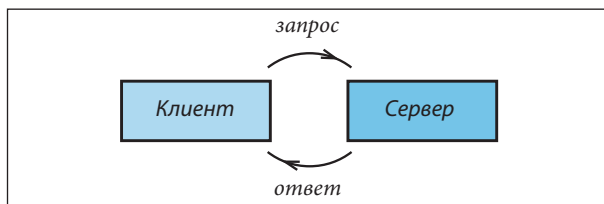


Рис. 7.3. Модель клиент-сервер

Ясно, что компьютер, который управляет общими файлами, будет называться **сервером файлов**, или **файл-сервером**, а компьютер, управляющий принтерами, — **сервером печати**.

Компьютер, который управляет линиями связи, другими общими ресурсами и, возможно, обеспечивает доступ к внешним сетям, называется **сервером**

сети. Как правило, именно на этом компьютере работает сетевая операционная система. Остальные компьютеры сети имеют более скромные параметры и называются **рабочими станциями**.

Главными преимуществами технологии клиент-сервер являются:

- эффективное использование дорогого оборудования;
- рациональное распределение работы (заданий) между компьютерами в зависимости от их мощности;
- надежная защита особо важных данных, которые хранятся только на сервере.

К сожалению, технология клиент-сервер сложна и требует мощных компьютеров. Эти факторы можно отнести к основным недостаткам данной технологии.

В технологии **равный-с-равным** функции всех компьютеров в сети идентичны.

Каждый компьютер в сети работает одновременно и как сервер, и как рабочая станция, предоставляя в общее распоряжение ресурсы, которыми располагает: часть файлов на жестком диске, устройство на оптическом диске, принтер и т. п. Как более простая, данная технология применяется в малых локальных сетях. Для больших сетей технология равный-с-равным не обеспечивает надежной защиты данных.

Аналогичным образом технология клиент-сервер применяется и для организации совместной работы двух и более программ.

Программа, предоставляющая во время своего выполнения определенные услуги, называется программой-сервером, а программы, которые обращаются к этим услугам, называются программами-клиентами.

Например, *программа-сервер*, которая управляет базой данных, выполняет следующие функции:

- обеспечивает защиту и безопасность данных;
- принимает и, если клиент получил соответствующую авторизацию, выполняет запросы на модификацию (изменение) данных;
- принимает запросы на чтение данных и в зависимости от статуса клиента разрешает или запрещает доступ к соответствующим данным;
- ведет журнал, в который заносит все операции, осуществляемые над базой данных.

Программа-клиент обеспечивает взаимодействие пользователя с базой данных и выполняет следующие функции:

- предлагает пользователю простой и удобный интерфейс;
- проверяет и редактирует данные, вводимые пользователем;
- направляет запросы программе-серверу;
- выводит информацию, извлеченную из базы данных.

Программы-серверы и программы-клиенты могут выполняться на одном и том же или на разных компьютерах. В последнем случае обработка данных является **распределенной**. Передача данных между программой-клиентом и программой-сервером осуществляется с помощью системы связи (рис. 7.2). Компьютер, на котором выполняется программа-сервер, называется **хостом** (от англ. *host* — хозяин).

Обычно в локальных сетях программа-клиент выполняется на рабочих станциях, а программа-сервер — на сетевом сервере. В случае региональных или глобальных сетей на каждом мощном компьютере выполняется несколько программ-серверов, которые предлагают разнообразные услуги программам-клиентам, выполняющимся на других компьютерах.

Вопросы и упражнения

- ❶ Объясните термин *технологии взаимодействия в сети*. Какие технологии взаимодействия в сети вы знаете?
- ❷ Как организована работа компьютеров в сети в случае технологии клиент-сервер?
- ❸ Каковы основные преимущества и недостатки технологии клиент-сервер?
- ❹ Объясните назначение сетевого сервера, файл-сервера, сервера печати и рабочей станции.
- ❺ Как организована работа компьютеров в сети в случае технологии равный-с-равным? Каковы преимущества и недостатки данной технологии?
- ❻ Определите технологию взаимодействия, реализованную в вашей сети. Есть ли в вашей сети файловый сервер и/или сервер печати?
- ❼ Существует ли в сети, с которой вы работаете, сетевой сервер? Обоснуйте ваш ответ.
- ❽ Реализована ли в вашей сети технология равный-с-равным? Обоснуйте ваш ответ.
- ❾ Как взаимодействуют программы в случае технологии клиент-сервер?
- ❿ Назовите функции программы-сервера и программы-клиента.
- ⓫ Объясните термин *хост*.
- ⓬ **ИССЛЕДУЙТЕ!** Компания открыла в разных городах страны агентства по продаже авиабилетов. Данные обо всех авиарейсах и о наличии свободных мест хранятся в компьютере, расположенном в центральном офисе компании. Какие сетевые технологии могут обеспечить эффективную работу агентств по продаже авиабилетов? Необходимы ли в данном случае программы-серверы и программы-клиенты? Какие функции будут выполнять данные программы?
- ⓭ **ПРОАНАЛИЗИРУЙТЕ!** Коллективные электронные игры возможны при совместной работе 5–10 компьютеров, объединенных в сеть. В такой игре каждый играет против всех. Соответствующие компьютеры предлагают для общего пользования раздел на жестком диске. Какая сетевая технология обеспечивает совместную работу компьютеров?
- ⓮ **ПРОЕКТИРУЙТЕ!** Разработайте технологию сетевого взаимодействия для компьютеров:
 - а) складов фирмы;
 - б) выставочных залов музея;
 - в) читальных залов библиотеки;
 - г) кассовых аппаратов магазинов, принимающих банковские карточки;
 - д) системы выдачи наличных через банкоматы;
 - е) системы оперативной проверки регистрационных номеров автомобилей (каждая бригада полиции оснащена микрокомпьютером с радиомодемом);
 - ж) лаборатории (класса) по информатике.

7.3. Топология и архитектура компьютерных сетей

Система связи любой компьютерной сети (рис. 7.2) обеспечивает передачу данных между компьютерами. Обычно передаваемые данные группируются в пакеты.

Каждый **пакет данных** содержит следующую информацию:

- адрес получателя;
- собственно данные;
- управляющую информацию;
- адрес отправителя.

Заметим, что пакет данных можно представить себе как обычный конверт, который пересылается с помощью традиционной почтовой службы. Путь, который проходит пакет, зависит от **топологии сети**.

Геометрическая конфигурация связей между компьютерами называется топологией сети.

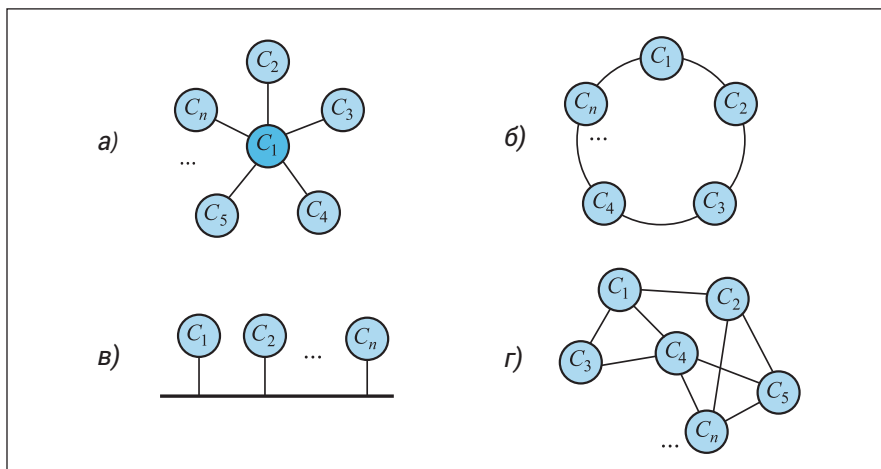


Рис. 7.4. Топологии сетей:

а — звезда; б — кольцо; в — магистраль; г — распределенная

В топологии типа **звезда** связь между двумя компьютерами C_i , C_j обеспечивается через центральный компьютер C_1 . По этой причине компьютер C_1 , называемый **главным компьютером**, играет важную роль в работе сети, осуществляя диспетчеризацию пакетов данных. Ясно, что выход из строя главного компьютера прерывает работу всей сети. Следовательно, главный компьютер должен быть очень надежным.

В топологии типа **кольцо** соединения между компьютерами образуют замкнутую цепочку. Пакет, посланный компьютером C_i , принимается компьютером C_{i+1} , который в свою очередь отправляет его компьютеру C_{i+2} и т. п., пока пакет не достигнет компьютера-приемника C_j . Поскольку выход из строя любого компьютера прерывает работу всей сети, все компьютеры C_1, C_2, \dots, C_n должны быть очень надежными.

В топологии типа **магистраль** присутствует один канал, к которому подключены все компьютеры. Каждый компьютер «подслушивает» магистраль и принимает пакеты, адресованные только ему. Любой компьютер может послать пакет только тогда, когда магистраль свободна.

Сети, основанные на топологии типа магистраль, очень надежны, поскольку связь между компьютерами C_i , C_j сохраняется даже в том случае, когда все другие компьютеры не работают.

В **распределенных топологиях** между каждой парой компьютеров существует несколько путей передачи данных. Например, один пакет данных, посланный компьютером C_1 компьютеру C_n (рис. 7.4г), может достичь адресата по маршруту $C_1 - C_2 - C_5 - C_n$, а другой пакет — по маршруту $C_1 - C_4 - C_n$. Очевидно, сеть будет функционировать даже в том случае, если один или несколько компьютеров и линий связи выйдут из строя.

Обычно топологии типа звезда, кольцо и магистраль применяют для локальных сетей. Региональные и глобальные сети имеют распределенную топологию. Объединение локальных сетей в региональные и глобальные сети осуществляется в соответствии с основными типологиями, представленными на рис. 7.4, где каждый узел C_1, C_2, \dots, C_n представляет собой подсеть. Следовательно, современные сети имеют иерархическую структуру.

Набор правил для управления процессами обмена данными в сети называется протоколом обмена или просто протоколом.

Любой **протокол** определяет режимы адресации компьютеров, длину и состав пакетов данных, алгоритмы обнаружения и исправления ошибок, режимы физического подключения сетевых адаптеров и кабелей и т. п. Одновременно с появлением первых компьютерных сетей каждый производитель вычислительной техники создавал свои собственные протоколы связи, что делало невозможным взаимное соединение компьютеров от разных производителей. Этот недостаток был устранен путем стандартизации протоколов. Напоминаем, что **стандарт** представляет собой документ, в котором регламентируется качество, характеристики, форма и т. п. определенного изделия. Международные стандарты разрабатывает Международная организация по стандартизации (*ISO — International Standards Organisation*). Другой организацией, играющей важную роль в стандартизации изделий электроники и вычислительной техники, является Институт инженеров по электротехнике и радиоэлектронике (*IEEE — Institute of Electrical and Electronics Engineers*).

Архитектурой сети называется набор ее основных характеристик: топология, протоколы связи, технология взаимодействия.

Далее рассматриваются наиболее распространенные архитектуры компьютерных сетей.

Ethernet (сеть в эфире) — локальные сети, реализованные в соответствии со стандартом *IEEE 802.3*. Используют магистраль из кабеля с витыми парами, из коаксиального или оптоволоконного кабеля. Скорость передачи данных достигает 100 Мбум/с . Данная архитектура была разработана фирмами *XEROX, Intel* и *DEC*.

Token-Ring (кольцо с жетоном) — локальные сети, реализованные в соответствии со стандартом *IEEE 802.5*. Используют кольцо из кабеля с витыми парами или из коаксиального кабеля. Скорость передачи данных достигает 16 Мбум/с . Данная архитектура разработана фирмой *IBM*.

DATAKIT — локальные, региональные или глобальные сети, разработанные фирмой *Bell Laboratories*. С точки зрения топологии эта сеть состоит из множества соединенных друг с другом звезд. Для оптоволоконного кабеля достигнута скорость передачи до $1,5 \text{ Гбум/с}$.

SNA (*System Network Architecture*) — архитектура, разработанная фирмой *IBM* для локальных, региональных и глобальных сетей. Протоколы данной

архитектуры основаны на стандартах *ISO*. Поначалу топология имела тип звезды, но в настоящее время изменилась на распределенную топологию, поддерживая и локальные сети.

ARPANET — архитектура, спроектированная несколькими университетами и корпорациями под эгидой Министерства обороны США (*Advanced Research Projects Agency*). Данная архитектура базируется на распределенной топологии и использует различные линии связи — от телефонных линий до спутниковых микроволновых каналов. Соответствующие линии связи соединяют отдельные суперкомпьютеры и разнообразные локальные и региональные сети, занимающие около половины земной поверхности. Архитектура **ARPANET** включает в себя следующие протоколы:

- протокол **IP** (*Internet Protocol*), предназначенный для соединения друг с другом локальных, региональных и глобальных сетей;
- протокол сервисов, основанных на соединениях, именуемый **TCP** (*Transmission Control Protocol*);
- протокол передачи файлов, называемый **FTP** (*File Transfer Protocol*);
- протокол электронной почты — **SMTP** (*Simple Mail Transfer Protocol*);
- протокол для удаленного подключения к компьютеру — **TELNET**.

На основе архитектуры **ARPANET** была разработана глобальная компьютерная сеть *Internet*, которая будет изучена в следующем параграфе.

Подчеркнем, что с внедрением микрокомпьютеров в различные механизмы, приборы и устройства (в частности, в бытовую технику) появилась возможность подключения их к сетям, называемым *сетями объектов* или *Интернетом вещей* (от англ. *Internet of Things*, сокращенно *IoT*).

Объектами, подключенными к *Интернету вещей*, можно управлять удаленно, что открывает новые возможности для повышения производительности труда и улучшения качества жизни людей. В то же время все более широкое распространение этих технологий сопровождается возникновением важных проблем, связанных с безопасностью сетей и подключенных к ним объектов. Возможные угрозы безопасности относятся не только к конфиденциальности и целостности обрабатываемой информации, но также к возможному материальному ущербу и угрозам безопасности людей, который может быть причинен объектами, вышедшими из-под контроля или контролируруемыми злоумышленниками.

Вопросы и упражнения

- ❶ Объясните термин *пакет данных*. Какую информацию содержит каждый пакет данных?
- ❷ Объясните термин *топология сети*.
- ❸ ТЕМАТИЧЕСКОЕ ИССЛЕДОВАНИЕ. Нарисуйте основные типы топологии сетей. Назовите преимущества и недостатки каждой из топологий.
- ❹ Объясните, как передаются пакеты данных в следующих сетях: звезда, кольцо, магистраль, распределенная топология.
- ❺ Уточните путь, по которому пойдет пакет, посланный компьютером C_2 компьютеру C_4 (рис. 7.4а).
- ❻ Найдите путь, по которому пойдут пакеты, посланные компьютером C_2 компьютеру C_4 (рис. 7.4б).

- 7 Сколько путей передачи пакетов существует между компьютерами C_1 , C_n на рис. 7.4г? Какой путь самый короткий?
- 8 Что произойдет с сетями на рис. 7.4, если выйдет из строя компьютер C_1 ? А если выйдет из строя компьютер C_2 ?
- 9 ЭКСПЕРИМЕНТИРУЙТЕ! Определите топологию локальной сети, с которой вы работаете. Перечислите преимущества и недостатки данной топологии.
- 10 Для чего предназначен протокол? Какие нормы содержит протокол?
- 11 Аргументируйте необходимость стандартизации протоколов. Кто разрабатывает соответствующие стандарты?
- 12 ЭКСПЕРИМЕНТИРУЙТЕ! Назовите протоколы, используемые в локальной сети, с которой вы работаете.
- 13 Объясните, как объединяются локальные сети в региональные и глобальные сети. Нарисуйте топологию региональных или глобальных сетей, к которым вы имеете доступ.
- 14 Объясните термин *архитектура сети*.
- 15 Приведите примеры архитектур локальных, региональных и глобальных сетей.
- 16 Охарактеризуйте архитектуру сети, с которой вы работаете.
- 17 Перечислите протоколы, используемые в архитектуре *ARPANET*. Применяются ли эти протоколы в сети, с которой вы работаете?
- 18 Объясните термин «Интернет вещей».
- 19 ИССЛЕДУЙТЕ! На основе информации, опубликованной в Интернете, узнайте, как и в какой степени Интернет вещей влияет на технологические процессы и повседневную жизнь людей.
- 20 ТВОРИТЕ! Напишите короткое эссе, в котором осветите возможности, предлагаемые Интернетом вещей, и способы преодоления препятствий, с которыми он сталкивается.

7.4. Глобальная сеть Интернет

Глобальная сеть *Интернет* основывается на распределенной топологии и состоит из отдельных компьютеров, локальных, региональных и глобальных подсетей (рис. 7.5).

Соединение сетей друг с другом осуществляется с помощью специальных сетевых устройств, называемых *шлюзами* и *маршрутизаторами*. **Шлюз** (*gateway*) — это специальный компьютер, предназначенный для соединения двух сетей с различными протоколами. **Маршрутизатор** (*router*) — это выделенный компьютер, с помощью которого соединяются сети, использующие одинаковые протоколы, и который используется для определения наилучшего пути передачи пакетов. Компьютеры, подключенные к *Интернету*, называются **хостами** (*host*). Работу сети регламентируют около 100 протоколов.

Идентификация компьютеров в пределах сети производится с помощью **интернет-адресов**. Они могут быть двух типов: цифровые и символические.

Цифровой адрес состоит из 32 двоичных цифр (4 *байтов*) и имеет структуру, представленную на рис. 7.6.

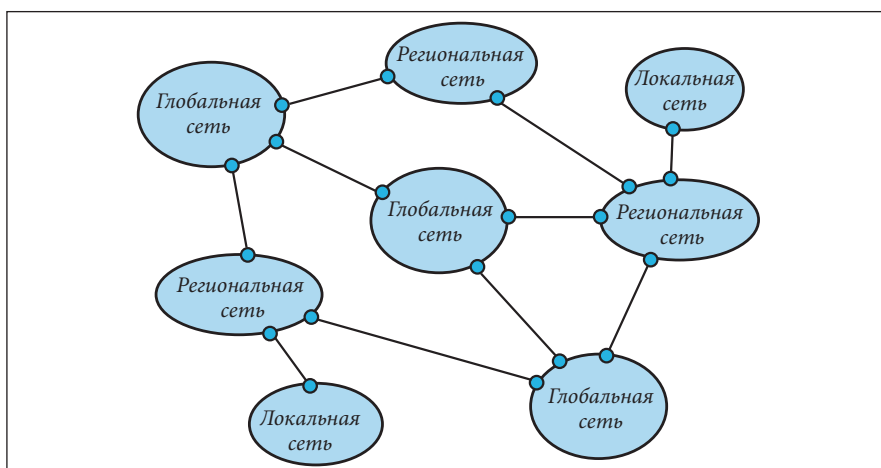


Рис. 7.5. Топология Интернета

Поскольку *Интернет* — это «сеть сетей», цифровой адрес содержит адрес подсети (поле *Адрес сети*) и адрес компьютера внутри самой подсети (поле *Адрес компьютера*). В зависимости от максимального количества компьютеров, которые можно идентифицировать внутри подсети, адреса делятся на классы *A*, *B* и *C*. Адреса, принадлежащие классам *D* и *E*, имеют специальное назначение. Характеристика *интернет-адресов* представлена в *таблице 7.1*.

Общая структура	Класс адреса	Адрес сети	Адрес компьютера
	0 1 ... 7 8 ... 31		
Класс A	0	Адрес сети	Адрес компьютера
	01 2 ... 15 16 ... 31		
Класс B	10	Адрес сети	Адрес компьютера
	012 3 ... 23 24 ... 31		
Класс C	110	Адрес сети	Адрес компьютера

Рис. 7.6. Структура цифровых адресов

Таблица 7.1

Характеристика цифровых адресов

Класс	Количество доступных адресов	
	сетей	компьютеров
A	$2^7=128$	$2^{24} = 16\,777\,216$
B	$2^{14}=16\,384$	$2^{16} = 65\,536$
C	$2^{21}=2\,097\,152$	$2^8 = 256$

Адреса класса *A* выделены для больших сетей, как правило, для глобальных. Сеть такого рода может содержать около 16 миллионов компьютеров. Адреса класса *B* предназначены для средних сетей, как правило, для региональных. Сеть такого рода может содержать около 65 тысяч компьютеров. Адреса класса *C* зарезервированы за относительно малыми сетями, включающими до 256 компьютеров. Каждый *интернет-адрес* уникален. Адреса компьютерам присваивает **Информационный центр сети** (*Network Information Center*).

Например, двоичное число

10010010 00110011 00001001 11110111

является адресом класса *B* и идентифицирует компьютер под номером

00001001 11110111,

который входит в состав сети с номером

010010 00110011.

Для удобства двоичные адреса представлены в десятичной форме (каждый байт в отдельности). Десятичные числа, соответствующие каждому байту, отделены друг от друга точками.

В случае рассмотренного выше примера получаем:

$(10010010)_2 = (146)_{10};$

$(00110011)_2 = (51)_{10};$

$(00001001)_2 = (9)_{10};$

$(11110111)_2 = (247)_{10}.$

Значит, в десятичной форме это адрес вида: 146.51.9.247.

Адреса в десятичной и тем более в двоичной форме неудобны для большинства пользователей. Поэтому чаще используются символические адреса.

Символический адрес состоит из имени компьютера-хоста и имен доменов, отделенных друг от друга точками. **Домен** представляет собой группу компьютеров, образованную по тематическим или географическим признакам. Любой домен может быть разделен на субдомены, приобретая таким образом иерархическую структуру. Доменные имена указаны в порядке увеличения зоны покрытия.

Например, символические адреса

c1.lme.ch.md

c5.lme.ch.md

идентифицируют компьютеры c1 и c5 из домена lme (Лицей “Михай Еминеску”).

Символические адреса

c1.lic.ch.md

c9.lic.ch.md

идентифицируют компьютеры `c1` и `c9` из домена `lic` (Лицей “Ион Крянгэ”). Домены `lme` и `lic` — это поддомены домена `ch` (Кишинэу). В свою очередь, `ch` — это поддомен домена `md` (Республика Молдова).

Аналогичным образом, символические адреса

`rector.ase.men.ro`

`decan.ase.men.ro`

определяют компьютеры `rector` и `decan` домена `ase` (Академия экономических знаний). Домен `ase` — это поддомен домена `men` (Министерство народного образования), а `men` — это поддомен домена `ro` (Румыния).

Обычно домен самого верхнего уровня является страной (`md`, `ro`, `us` и т.п.) или видом организации (`com` — коммерческая, `mil` — военная, `edu` — образовательная и т. п.).

Отношения включения между доменами можно отобразить с помощью диаграмм Эйлера, часто используемых в теории множеств. Для примера на *рис. 7.7* представлена такая диаграмма для некоторых символических адресов домена `md`.

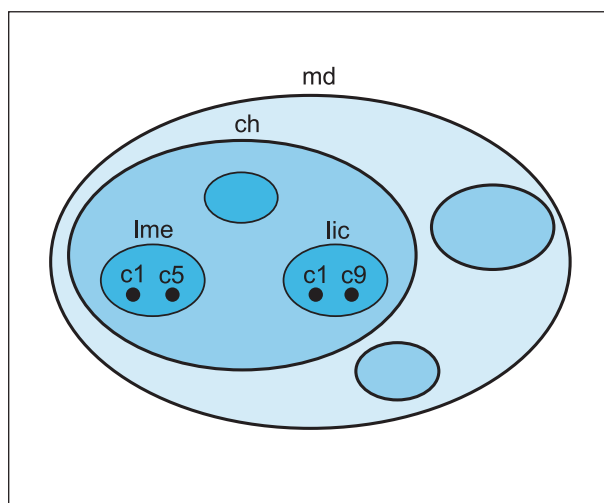


Рис. 7.7. Диаграмма Эйлера для символических адресов

Для **преобразования** символических адресов в цифровые и обратно в каждом домене имеется **сервер имен** (*name server*). Эта программа управляет соответствующим доменом без вмешательства серверов верхнего уровня иерархии. Следовательно, в *Интернете* не существует центрального компьютера, отвечающего за приблизительно 10^{10} адресов сети.

В случае примера, приведенного ранее (*рис. 7.7*), адреса компьютеров `c1` и `c5` обрабатываются на сервере `lme`, а адреса компьютеров `c1` и `c9` — на сервере `lic`. Сервер `ch` будет работать только с адресами серверов `lic` и `lme`, не вмешиваясь в обработку адресов из доменов `lic` и `lme`. Аналогично сервер `men` не обрабатывает адреса компьютеров `rector` и `decan`, предоставляя эту работу серверу `ase`.

Вопросы и упражнения

- ❶ Нарисуйте топологию *Интернета*. Как соединяются друг с другом подсети внутри *Интернета*?
- ❷ Для чего предназначены шлюзы и маршрутизаторы?
- ❸ Как идентифицируются компьютеры в *Интернете*? В чем преимущества и недостатки цифровых адресов? А символических адресов?
- ❹ На какие классы подразделяются цифровые адреса? Для чего предназначена такая классификация?
- ❺ Объясните назначение полей *Адрес сети* и *Адрес компьютера*, входящие в состав цифрового адреса.
- ❻ Кто выделяет адреса для компьютеров из *Интернета*?
- ❼ Определите классы следующих адресов. Уточните адрес подсети и адрес компьютера в подсети:

a) 45.201.19.63; d) 192.109.58.170;

b) 201.165.213.91; e) 15.21.207.250;

c) 154.36.79.200; f) 217.15.69.113.

- ❽ По каким критериям компьютеры объединены в домены? Приведите примеры нескольких доменов верхнего уровня.
- ❾ Даны следующие символические адреса:

a) c1.lme.ch.md; f) c4.lme.ch.md;

b) c3.lme.ch.md; g) c5.lme.ch.md;

c) c1.lic.ch.md; h) c9.lic.ch.md;

d) director.lic.ch.md; i) prof.lic.ch.md;

e) elev1.lic.ch.md; j) elev4.lic.ch.md.

Уточните домены каждого компьютера и отношения включения между доменами. Нарисуйте диаграммы *Эйлера* для рассматриваемых адресов.

- ❿ Нарисуйте диаграммы *Эйлера* для перечисленных символических адресов. Уточните домены компьютеров и соответствующие отношения включения.

a) rector.ase.men.ro; d) rector.ase.met.md;

b) decan.ase.men.ro; e) decan.ase.met.md;

c) student.info.ase.men.ro; f) student.cib.met.md.

- ⓫ В чем назначение сервера имен? Какие адреса обрабатывают эти серверы?
- ⓬ Уточните серверы имен для адресов из упражнений 9 и 10. Укажите адреса, обрабатываемые каждым сервером.
- ⓭ ЭКСПЕРИМЕНТИРУЙТЕ! Определите цифровой и символический адреса компьютера, на котором вы работаете. Уточните класс адресов, адрес подсети и адрес компьютера в рамках подсети. Нарисуйте диаграмму *Эйлера*, представляющую отношения включения между доменами, к которым принадлежит ваш компьютер.

7.5. Сервисы Интернета

В спектр услуг *Интернета* входит:

- удаленный доступ к компьютерам;
- передача файлов;
- электронная почта;
- новости и конференции (дискуссии);
- распространение и поиск информации и т. п.

Взаимодействие компьютеров и программ, которые представляют данные услуги, основывается на модели клиент-сервер. Обычно на компьютере потребителя выполняется программа-клиент, а на компьютере, предоставляющем сервис (оказывающем услугу), выполняется программа-сервер.

Сервис Telnet позволяет пользователю получить доступ к другому компьютеру, находящемуся на произвольном расстоянии (так называемый удаленный доступ). После установления соединения компьютер пользователя используется как простой терминал, находящийся на большом расстоянии от центрального компьютера. Пользователь может запускать на удаленном компьютере различные программы, просматривать файлы, изменять текущую директорию и т. п. Защита компьютеров и соответствующих данных обеспечивается применением паролей. Сервис Telnet применяется для совместного использования дорогостоящих ресурсов, например суперкомпьютеров.

Сервис передачи файлов, или коротко — **сервис FTP** (*File Transfer Protocol*) позволяет пользователю копировать файлы, находящиеся на компьютерах, размещенных в различных географических точках. Данный сервис предлагает два режима передачи файлов:

- двоичный режим, при котором сохраняется последовательность битов файла, так что оригинал и копия идентичны с точностью до бита;
- текстовый режим, в котором передаются наборы символов в коде ASCII.

Как правило, для получения доступа к серверу FTP клиент должен ввести пароль. Однако существуют публичные серверы (*FTP anonymous*), которые разрешают доступ к файлам без необходимости ввода специального пароля.

Сервис электронной почты (*electronic mail*, или сокращенно — *e-mail*) имитирует режимы работы обычной почты.

В электронное письмо, называемое **сообщением** (*message*), входят:

- адрес получателя;
- тема сообщения, выраженная в нескольких словах;
- адрес отправителя;
- текст письма;
- файлы, которые могут быть присоединены к сообщению.

Присоединенные файлы могут быть любого типа: текстовые, звуковые, графические, программы и т. п.

Письма хранятся в специальных файлах, называемых **почтовыми ящиками** (*mailbox*). Адрес любого почтового ящика имеет вид:

`<Имя ящика>@<Адрес компьютера>` ,

где

`<Имя ящика>` — это название почтового ящика (обычно фамилия пользователя или аббревиатура);

@ — символ коммерческого „at”;

<Адрес компьютера> — символический адрес компьютера — клиента, на котором создан почтовый ящик.

Примеры:

- 1) petrescu@c1.lme.ch.md
- 2) florea@director.lic.ch.md
- 3) ionescu@c1.lme.ch.md
- 4) barbu@director.lic.ch.md

Читатели могут отправлять письма авторам данного учебника по адресу:

Anatol_Gremalschi@yahoo.com

Сообщения пересылаются через сеть почтовых серверов, которые выполняют роль обычных почтовых отделений.

Сервис электронной почты очень популярен благодаря его неоспоримым преимуществам — скорости, возможности присоединять к письмам файлы любого типа и развитым возможностям редактирования писем.

Самым современным сервисом распространения и поиска информации в *Интернете* является **сервис WWW** (*World Wide Web* — Всемирная паутина). В данном сервисе информация представлена в форме Web-страниц.

Web-страница — это файл, написанный на языке *HTML* (*Hypertext Markup Language* — Язык разметки гипертекста), который содержит, кроме самой информации, ссылки на другие страницы *Web*. Адресуемые ссылками страницы могут быть размещены на этом же компьютере или на компьютерах, расположенных в различных географических точках (рис. 7.8).

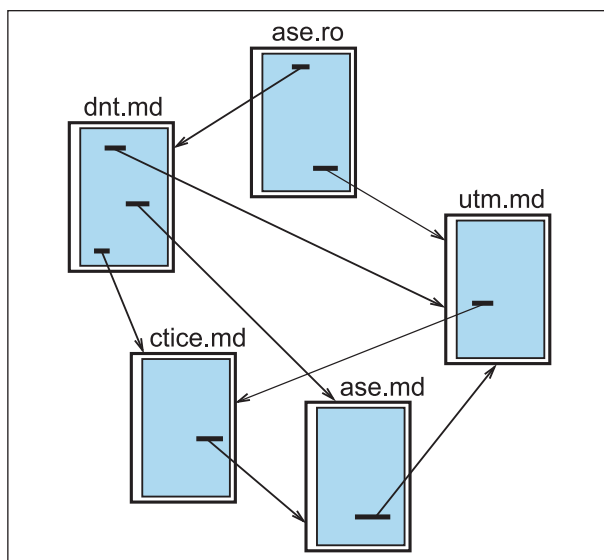


Рис. 7.8. Web-страницы

Взаимодействие программ в рамках сервиса WWW выполняется по типу клиент-сервер. Пользователь, который хочет предложить широкой публике определенную информацию, устанавливает на своем компьютере программу WWW-сервер и разрабатывает одну или несколько Web-страниц. Сервер принимает запросы, поступающие от других компьютеров, и предоставляет доступ к соответствующим страницам. Компьютер, на котором размещены Web-страницы и WWW-сервер, носит название **сайт** (англ. *site* — «местонахождение»).

Программа-клиент обеспечивает прием и отображение на экране Web-страниц, которые считываются с различных компьютеров (серверов) в *Интернете*. Сразу после запуска эта программа отображает установленную по умолчанию домашнюю страницу (*home page*) и ожидает указаний пользователя. Когда пользователь активизирует ссылку, программа-клиент устанавливает соединение с Web-сервером и копирует с него страницу, определенную ссылкой. Скопированная страница выводится на экран.

Далее, когда пользователь активизирует другую ссылку, программа-клиент вновь устанавливает соединение с одним из компьютеров сети, вновь читает Web-страницу и т. п. Другими словами, пользователь «перелистывает» Web-страницы, находящиеся на различных компьютерах, независимо от их географического расположения. По этой причине программы-клиенты называются **программами перелистывания** или **программами навигации** (с англ. *browser* или *explorer*).

В рамках сервиса WWW ресурсы сети определяются с помощью специальных адресов, называемых адресами URL (*Uniform Resource Locator* — Универсальный указатель ресурсов). Данные адреса имеют вид:

`<протокол>: // <Символический адрес>[:<Порт>]/<Путь>/<Файл>`

где

<Протокол> — определяет название протокола для передачи данных по сети;

<Символический адрес> — адрес компьютера, содержащего соответствующий файл;

<Порт> — порт доступа (необязателен);

<Путь>/<Файл> — спецификация файла.

Вот пример нескольких URL-адресов, которые содержат полезную информацию:

<http://www.mecc.gov.md> — сайт Министерства образования, культуры и исследований, Республика Молдова;

<http://www.ctice.gov.md> — сайт Центра информационных и коммуникационных технологий в образовании, Республика Молдова;

<https://www.sciencemuseum.org.uk> — сайт Лондонского Музея науки;

<http://www.nasa.gov> — сайт агентства NASA, США.

Напоминаем, что обозначение *http* определяет протокол передачи гипертекста (*Hypertext Transfer Protocol*).

В настоящее время количество файлов в *Интернете* исчисляется миллиардами. Естественно, не может быть и речи о поиске необходимой информации путем чтения каждого файла в отдельности. Для упрощения поиска информации в *Интернете* были созданы специальные поисковые серверы (*search engine*).

Поисковый сервер — это компьютер, который непрерывно исследует сеть и читает Web-страницы или другие ресурсы.

Их классификация зависит от содержания, а их адреса запоминаются в базе данных на поисковом сервере.

Программа-клиент посылает поисковому серверу запрос, в котором указывает признаки необходимой информации. Сервер опрашивает базу данных и передает клиенту список адресов, по которым может быть найдена запрошенная информация.

Для примера приведем адреса наиболее популярных серверов поиска:

<http://www.yahoo.com> — сервер YAHOO (*Yet Another Hierarchically Organized Oracle* — Еще Один Иерархически Организованный Оракул) компании *Yahoo! Inc.*;

<http://www.google.com> — сервер GOOGLE корпорации *Google Inc.*;

<http://www.bing.com> — сервер BING корпорации *Microsoft Inc.*;

<http://www.yandex.ru> — сервер YANDEX компании «Яндекс»;

Доступ к данным серверам свободный, т. е. бесплатный.

Вопросы и упражнения

- ❶ Назовите гамму услуг, предлагаемых в *Интернете*. Как взаимодействуют компьютеры сети в процессе оказания некоторой услуги?
- ❷ Для чего предназначен сервис *FTP*? Какие файлы можно передавать с помощью данного вида сервиса?
- ❸ Изучите программу *FTP*, установленную на вашем компьютере. Скопируйте несколько файлов с публичных серверов *FTP* или с других серверов, к которым вы имеете доступ.
- ❹ От чего зависит скорость передачи файлов? Определите скорость передачи файлов с нескольких серверов *FTP*, расположенных в Республике Молдова, Румынии, США и России.
- ❺ Как задаются адреса в сервисе электронной почты?
- ❻ Объясните, как взаимодействуют компьютеры клиентов и сервер почты в рамках соответствующей службы.
- ❼ Назовите составляющие электронного письма? Какие из них необязательны?
- ❽ Объясните, как формируется электронный адрес. Узнайте *email*-адреса ваших друзей.
- ❾ Изучите программу электронной почты, установленную на компьютере, с которым вы работаете. Проверьте, имеет ли данная программа следующие возможности:
 - использование нескольких почтовых ящиков;
 - сортировку и хранение писем в папках;
 - написание писем по шаблонам;
 - шифрование и дешифрование корреспонденции;
 - оповещение в момент поступления новой корреспонденции;
 - проверку получения адресатом посланного письма;
 - использование электронных подписей.
- ❿ В чем преимущества электронной почты? Может ли она заменить традиционную почту?

- ❶ РАБОТА В ГРУППАХ. Создайте группы по четыре пользователя электронной почты и экспериментально определите скорость передачи корреспонденции.
- ❷ Какую информацию содержит *Web*-страница? Как эти страницы образуют всемирную паутину?
- ❸ Для чего предназначен *Web*-сервер? А *Web*-клиент? Как взаимодействуют эти программы?
- ❹ Объясните, как работает *Web*-клиент. Как данная программа находит *Web*-страницы, размещенные на разных компьютерах?
- ❺ Объясните способ указания ресурсов в *Интернете* с помощью адресов *URL*. Для чего предназначены различные поля таких адресов?
- ❻ ЭКСПЕРИМЕНТИРУЙТЕ! Определите тип программы просмотра *Интернета*, установленной на вашем компьютере. Каковы возможности данной программы? Прочитайте *Web*-страницы, адреса которых приведены в данном параграфе.
- ❼ Для чего предназначен поисковый сервер? Какие услуги предлагает такой сервер?
- ❽ ИССЛЕДУЙТЕ! Найдите с помощью любого поискового сервера поставщиков услуг (провайдеров) *Интернет* в Республике Молдова.
- ❾ ИССЛЕДУЙТЕ! Кроме сервисов, изученных в данном параграфе, сеть *Интернет* предлагает и другие сервисы, такие как *Archie*, *Gopher*, *WAIS*, подписки новостей, конференции и т. п. Используя поисковый сервер, найдите информацию об этих службах.
- ❿ ПРОЕКТЫ. Создайте *Web*-страницы вашего класса и лица.

Глава 8

МОДУЛИ ПО ВЫБОРУ

Модули в этой главе не являются обязательными. Что это значит? Вы можете выбрать один из них и изучить его с помощью компьютерного обучения в удобном для вас темпе самостоятельно или с коллегами, выполняя практические упражнения и разрабатывая проекты. После того, как вы выбрали модуль, вместе с учителем информатики необходимо решить, какие прикладные программы вы будете использовать. Также необходимо убедиться, что эти программы лицензированы или распространяются бесплатно, что они установлены на компьютерах в компьютерных классах и, в зависимости от обстоятельств, на ваших персональных компьютерах.

Масштабные проекты (например, в рамках города, села, школы) будут разрабатывать команды учеников, и каждый член команды будет специализироваться на определенном аспекте (например, историческом, демографическом, экономическом, культурном, этнографическом и др.).

Также команды будут разрабатывать проекты, связанные с богатым мультимедийным контентом, специализация членов команды будет осуществляться по типу и специфике материалов (например, изображения, аудио, видео, дизайн графических интерфейсов, программирование алгоритмов обработки данных и т. п.).

8.1. Техники аудио-видео обработки

Данные техники будут изучаться поэтапно следующим образом.

1. Оцифровка аудиоинформации. Вспомните из физики характеристики звука как физического явления и параметры, которые влияют на восприятие звуковых волн человеком. Из информатики вспомните цифровое кодирование аудиоинформации. Используя богатые ресурсы Интернета, узнайте, как осуществляются сжатие и распаковка аудиоданных, каковы форматы файлов, содержащих аудиоданные.

Особое внимание уделите оборудованию для цифровой записи и воспроизведения звука: технические параметры, взаимодействие с персональными компьютерами, соотношение цены и качества, области применения. В зависимости от оборудования компьютерного класса и имеющегося у вас персонального цифрового оборудования, например: цифровых записывающих устройств, звуковых карт, микрофонов, динамиков, усилителей, цифровых мультимедийных студий и т. д., решите, как создавать, хранить и воспроизводить звуковые записи.

Упражнения, рекомендуемые для данного этапа:

- объяснение принципов кодирования и декодирования аудиоданных;
- определение объема несжатых аудиоданных, зная параметры кодирования;
- определение аудиоформата по расширению имени файла;

– распознавание и объяснение основных параметров и характеристик оборудования, наиболее часто используемого для сбора, записи, обработки и воспроизведения звуковых фрагментов.

Для более глубокого понимания физических и информационных процессов, связанных с цифровой записью и воспроизведением звука, рекомендуются следующие *тематические исследования*:

- аналоговая и цифровая запись звука;
- сравнительный анализ устройств, предназначенных для цифровой обработки звука;
- сравнительный анализ форматов аудиофайлов.

Закрепление и углубление знаний из области оцифровки звуковой информации возможны, работая над следующими *исследовательскими проектами*:

- влияние частоты дискретизации звуковых сигналов на качество их воспроизведения;
- влияние величины шага квантования звуковых сигналов на качество их воспроизведения;
- влияние степени сжатия звуковых сигналов на качество их воспроизведения.

2. Обработка цифровых аудиозаписей. Внимательно изучите предлагаемые на рынке программных продуктов лицензионные и свободно распространяемые программные приложения, предназначенные для цифровой обработки аудиоданных. Решите, какую программу вы будете использовать, и установите ее на том компьютере, на котором будете работать.

Изучите рабочую среду и преимущества, предоставляемые звуковым редактором, с которым вам предстоит работать. Особое внимание обратите на поддерживаемые форматы аудиофайлов, способы их хранения и организации, совместимость редактора с внешними устройствами записи и воспроизведения звука. Убедитесь, что графический интерфейс редактора достаточно интуитивен и прост для использования.

Экспериментируя с различными цифровыми аудиозаписями, постарайтесь развить следующие навыки по их обработке:

- разбивка звуковой записи на фрагменты и их склеивание;
- преобразование амплитуды (увеличение, микширование, нормализация);
- изменение тона и длительности звука;
- фильтрование звуковых сигналов;
- применение эффектов.

В процессе обработки звуковых записей используйте средства для анализа звука, предлагаемые редактором: анализ спектра, изменение объема.

С целью рациональной организации аудиофайлов на носителях информации и с учетом повышенных требований к качеству воспроизведения оцифрованного звука произведите конверсию звуковых форматов.

Упражнения, которые помогут вам закрепить теоретические знания и развить практические навыки применения цифровых устройств и программных продуктов, предназначенных для обработки звука, включают:

- применение средств, предоставляемых звуковым редактором;
- сравнительный анализ одной и той же записи, представленной в различных аудиоформатах;
- анализ качества воспроизведения звука в зависимости от параметров записи и хранения аудиоданных;

- конверсию аудиоформатов звуковых файлов;
- передачу аудиофайлов через различные средства цифровых коммуникаций.

Синергия информационных и специализированных аудиовизуальных средств может быть обеспечена путем разработки следующих *проектов*:

- разработка звукового фона для важных событий в личной жизни, семье, классе, школе, обществе;
- запись и микширование аудиоинформации из нескольких источников во время праздничных, спортивных, художественных, развлекательных мероприятий;
- разработка саундтреков к видеофильмам о различных событиях школьной и общественной жизни.

3. Оцифровка видеоинформации. Для начала вам нужно вспомнить из физики о волновой и корпускулярной природе света и иметь в виду, что в информатике свет рассматривается как электромагнитная волна. Затем узнайте в Интернете, как человеческий глаз воспринимает свет и как цвета, яркость и контраст изображения зависят от величины физических параметров электромагнитных волн. Для более глубокого понимания процессов оцифровки видеоинформации вам необходимо освежить свои знания о методах квантования динамических изображений и расчета количества информации, которую они содержат. Обратите особое внимание на частоту дискретизации во времени, частоту дискретизации в пространстве, шаг квантования видеосигнала, мощность разрешения, количество основных цветов.

Исходя из того, что очень часто видеоизображения содержат информацию, избыточную для человеческого восприятия, ознакомьтесь с методами сжатия и распаковки видеоданных с соответствующими этим методам форматами файлов. Определите связь между скоростью передачи данных и качеством воспроизведения видео, записанного в цифровой форме.

Закрепите теоретические знания, изучив оборудование для цифровой записи видеоинформации и ее воспроизведения: технические параметры, взаимодействие с персональными компьютерами, соотношение цены и качества, области применения. В зависимости от оснащения вашей компьютерной лаборатории и имеющегося у вас личного цифрового оборудования, например: цифровых видеокамер, видеокарт, объективов, средств просмотра, мультимедийных проекторов, цифровых мультимедийных студий и т. д., решите, как вы будете создавать, хранить и воспроизводить цифровые видеозаписи.

Упражнения, рекомендуемые для данного этапа:

- описание сфер человеческой деятельности, в которых используется видеоинформация;
- описание и сравнение параметров электромагнитных волн с точки зрения их восприятия человеком;
- описание и объяснение видеофакторов, потенциально опасных для здоровья человека;
- объяснение принципов кодирования и декодирования видеоданных;
- определение объема несжатых видеоданных, зная параметры кодирования;
- определение формата видео, зная расширения имени файла;
- определение и объяснение основных параметров и характеристик оборудования, часто используемого для сбора, записи, обработки и воспроизведения видеофрагментов.

Рекомендуется провести следующие *тематические исследования*:

- аналоговая запись и цифровая запись видеoinформации;
- восприятие человеком различных электромагнитных волн;
- влияние изменения параметров электромагнитных волн на их восприятие человеком.

Перечисленные исследовательские проекты предназначены для более глубокого понимания того, как теоретические достижения информатики влияют на технологии сбора, записи, хранения, обработки и воспроизведения видеoinформации:

- влияние изменения частоты дискретизации видеосигналов на качество их воспроизведения;
- влияние изменения шага квантования видеосигналов на качество их воспроизведения;
- влияние степени сжатия видеосигналов на качество их воспроизведения.

4. Цифровая обработка видео. На рынке программных продуктов и в свободном распространении можно найти продвигаемые самые разнообразные компьютерные программы, предназначенные для обработки видеозаписей. Пользователи могут выбирать как профессиональные, так и любительские программные приложения. Обычно практически во всех операционных системах есть приложения для воспроизведения видео, а в некоторых из них даже есть видеоредакторы.

Поэтому перед прохождением данного этапа каждый ученик под руководством учителя информатики должен решить, какие программные продукты использовать с учетом их совместимости с цифровым оборудованием, используемым для записи и воспроизведения видеoinформации. Необходимо также учитывать тот факт, что редакторы цифровых видео требуют значительных вычислительных ресурсов, таких как память (порядка гигабайт) и быстродействие (порядка гига операций в секунду).

Изучение возможностей предлагаемых цифровых видеоредакторов будет направлено на формирование и развитие следующих навыков:

- разбиение видео на фрагменты и их склеивание;
- редактирование видеофрагментов;
- временные преобразования;
- ассоциация звукового сопровождения;
- изменение времени воспроизведения;
- фильтрация видеосигналов;
- применение эффектов;
- субтитры в видеофрагментах;
- преобразование формата.

Упражнения, рекомендуемые для данного этапа:

- использование основных возможностей цифрового видеоредактора;
- сравнительный анализ записей одних и тех же видеофрагментов в разных видеоформатах;
- анализ качества воспроизведения видеофрагментов в зависимости от параметров записи и хранения;
- конвертация форматов видеофайлов.

Тематика рекомендуемых проектов будет аналогична той же, что и в случае цифровой обработки аудиофрагментов:

- разработка видеодомфона для важных событий в личной жизни, семье, классе, школе, обществе;
- запись и микширование видеоинформации, поступающей из нескольких источников во время праздничных, спортивных, художественных, развлекательных мероприятий;
- монтаж видеофильмов о различных событиях школьной и общественной жизни.

5. Распространение мультимедийной информации. На данном этапе, самостоятельно или под руководством учителя информатики, изучите онлайн-сервисы, которые предлагают вам возможность:

- осуществлять поиск нужных мультимедийных ресурсов;
- распространять собственные мультимедийные разработки;
- редактировать онлайн мультимедийные фрагменты.

Рекомендуется разработать каталог предпочитаемых услуг и подробно изучить каждую из наиболее часто используемых пользователями услуг.

На данном этапе предлагаем следующие *упражнения*:

- регистрация и создание собственных профилей в онлайн-сервисах;
- публикация собственных мультимедийных разработок;
- передача мультимедийных файлов различными цифровыми средствами связи;
- поиск мультимедийной информации в Интернете;
- включение ссылок в личные, классные, школьные, ассоциативные веб-страницы на собственные мультимедийные ресурсы в Интернете.

В качестве исследования рекомендуем провести сравнительный анализ онлайн-сервисов распространения мультимедийной информации, выявив их преимущества и недостатки, области применения с учетом специфики мультимедийной информации, предназначенной для распространения в виртуальном пространстве.

6. Цифровая этика и соблюдение авторских прав. Если в древние времена тексты, высеченные на каменных плитах или написанные на папирусе, были доступны очень небольшому кругу людей, то с появлением книгопечатания количество людей, получивших доступ к книгам, а затем и к газетам, увеличилось в геометрической прогрессии. Хотя радио и телевидение еще больше упростили процесс распространения информации, такие структуры, как типографии, радио- и телестанции, принадлежали и по-прежнему принадлежат государству или крупным компаниям. В результате у простых граждан не было возможности самостоятельно публиковать для максимально возможного количества людей созданные ими книги, фотографии, фонограммы, фильмы. Ситуация радикально изменилась с появлением Интернета и услуг онлайн-мультимедийного вещания, которые позволяют каждому из нас публиковать ту информацию, которую мы считаем важной, интересной, оригинальной или ценной с научной или художественной точки зрения.

Несомненно, эта свобода выражения мнений способствует консолидации и развитию демократических обществ, но требует от каждого из нас соблюдения в виртуальном пространстве определенных правил. Вы изучали эти правила, обычно называемые *правилами цифровой этики*, начиная с гимназических классов. Однако мы настоятельно рекомендуем вам освежить свои знания в этой области, строго соблюдать правила и помнить, что в многокультурном и

многоконфессиональном мире слова, изображения, звуковые и видеоматериалы могут быть интерпретированы и поняты по-разному. Поэтому прежде чем размещать в Интернете разработанный вами мультимедийный продукт, убедитесь, что он несет позитивную информацию, продвигает общечеловеческие ценности, поощряет участие и инклюзию.

Другой очень важный аспект, связанный с распространением мультимедийных продуктов, это необходимость соблюдения авторских и смежных прав. Обратите внимание, что авторское право распространяется на литературные, художественные и научные произведения, выраженные в следующих формах:

- письменная (рукопись, машинописный текст, партитура и т. д.);
- устная (публичное исполнение и др.);
- аудио или видео печать (механическая, магнитная, цифровая, оптическая и др.);
- изображение (рисунок, эскиз, картина, план, фотография и др.);
- трехмерная (скульптура, модель, макет, конструкция и др.).

Поэтому прежде чем включать в свои разработки фрагменты из каких бы то ни было произведений, уточните, допускают ли это обладатели авторских прав.

Также не представляйте в качестве личных творений идеи, фрагменты текста, изображения, фонограммы, видеogramмы, взятые из произведений и мультимедийных продуктов, созданных другими, так как это плагиат, а проще говоря — воровство и противоправное деяние, наказуемое соответствующими статьями закона.

Если авторы произведений и мультимедийных продуктов разрешают использование материалов (об этом можно осведомиться, внимательно прочитав лицензию), обязательно укажите источник: имя автора, название работы, издательство, название студии, URL-адрес и т. д.

Рекомендуемые *упражнения* для данного этапа:

- идентификация знаков, декларирующих авторские права;
- разъяснение правил соблюдения авторских прав;
- определение типов лицензий на распространение;
- использование лицензий на распространение.

В качестве *тематических исследований* рекомендуется проанализировать соблюдение авторских прав в собственных мультимедийных продуктах, в мультимедийных продуктах, разработанных и опубликованных в Сети однокурсниками и учениками вашей школы, учащимися других учебных заведений. Используйте функции, предоставляемые поисковыми системами, для поиска текстовой, графической, аудио- и видеоинформации. Пригодятся вам и программы по борьбе с плагиатом, доступные в Интернете, в режиме онлайн.

8.2. Визуальное программирование

Исторически первые программы были написаны на языке машинного кода. Впоследствии программисты перешли на языки ассемблера и языки программирования высокого уровня. В случае языков программирования высокого уровня программы представляют собой тексты, которые для «понимания» компьютером «переводятся» в двоичные коды.

В течение длительного периода времени программные продукты разрабатывались с использованием языков программирования высокого уровня, перво-

начально пользуясь для написания кода простыми текстовыми редакторами, а затем интегрированными средами разработки программ. Эти среды, помимо редактирования текстов и быстрого внесения изменений, позволяют запускать и отлаживать разрабатываемые программы. Вы уже знаете такие среды программирования и даже разрабатывали и отлаживали в них программы.

Независимо от среды разработки программ, используемых в учебных или производственных целях, в них работают с текстами. Однако с ростом сложности программ тексты стали объемными и трудными для восприятия даже их авторами, что стимулировало развитие так называемого визуального программирования — способа разработки программ не путем написания текстов, а путем манипулирования графическими объектами.

Вам уже знакома среда визуального программирования *Scratch*, специально разработанная для изучения информатики, начиная с начальных классов. В этой среде программирования учащиеся реализуют алгоритмы управления так называемыми спрайтами, манипулируя графическими объектами, которые представляют самих спрайтов, команды, которые они должны выполнять, и блоки управления потоком выполнения: ветвление в зависимости от соблюдения определенных условий, организацию циклов и подпрограмм.

Классическими средствами визуального программирования, разработанными для производственных целей, являются приложения для работы с электронными таблицами, изучаемые в гимназических классах. В этих приложениях пользователь реализует алгоритмы обработки числовых данных не путем написания программ на языке высокого уровня, например на PASCAL или C++, а путем работы с конкретными графическими объектами, такими как рабочие листы, ячейки, строки, столбцы, диаграммы, графики, формы и т. д.

В настоящее время известные компании, работающие в области информационных и коммуникационных технологий, предлагают очень широкий спектр средств визуального программирования, что значительно облегчает реализацию даже самых сложных алгоритмов. Всё более широкое использование визуального программирования открывает новые возможности не только для людей, которые хотят сделать профессиональную карьеру в области информатики, но и для тех, кто увлечен цифровым искусством, компьютерным обучением, разработкой интерактивных компьютерных игр.

Таким образом, знание визуального программирования позволит ученикам принимать обоснованные решения относительно продолжения учебы после окончания лицея, а также поможет им применять компьютер в тех сферах профессиональной деятельности, в которых они намерены специализироваться. А чтобы облегчить процесс изучения визуального программирования, мы рекомендуем ученикам пройти следующие этапы:

1. Выбор среды визуального программирования. Очевидно, что любая программа, разработанная в средах визуального программирования, имеет свой эквивалент в текстовой форме, написанный на языке программирования высокого уровня. Более того, чтобы эффективно реализовать определенные алгоритмы обработки данных, иногда даже в визуальном программировании, требуется писать подпрограммы на языке высокого уровня. Поэтому при выборе среды визуального программирования будет приниматься во внимание не только простота ее использования, но и ориентация на определенный язык программирования высокого уровня.

Например, ученики, изучавшие язык *PASCAL*, выберут среды визуального программирования *Delphi* или *Lazarus*, а те, кто изучал языки семейства *C* — среды визуального программирования *Microsoft Visual Studio* или *Code :: Blocks*.

Ученики, которые хотят изучать только основы визуального программирования, не прибегая к профессиональному языку программирования высокого уровня, могут сделать это с помощью *MIT App Inventor*, который использует визуальный язык и графические интерфейсы, аналогичные тем, которые применяются в среде программирования *Scratch*.

Рекомендуем ученикам, прежде чем выбирать определенную среду визуального программирования, ознакомиться с их классификацией, с их техническими и эргономическими параметрами, с требованиями, которые эти продукты предъявляют к компьютерам, которые будут использоваться в процессе обучения, а также к требованиям относительно возможности передачи через Интернет. Ученики также должны учитывать свои цели: углубить навыки разработки графических интерфейсов для консольных и веб-приложений или развить новые навыки в областях, где пересекаются информационные технологии и искусство: цифровая анимация, компьютерные игры, дополненная реальность (англ. *augmented reality*, *AR*) и др.

2. Изучение понятий и основных принципов визуального программирования. На этом этапе вам нужно будет ознакомиться с понятиями графического объекта, связей и действий, с типами первичных графических объектов и метаобъектов, со свойствами объектов. Особое внимание необходимо уделить пониманию специфики событийно-ориентированного программирования в визуальном контексте: ситуации, действия и результаты.

Основные *упражнения*, рекомендуемые для данного этапа:

- объяснение принципов визуального программирования;
- объяснение основных понятий визуального программирования;
- классификация объектов / графических команд;
- контроль событий, действий, состояний.

Следующие *тематические исследования* будут способствовать более глубокому пониманию специфики визуального программирования:

- графические команды в сравнении с письменными командами;
- гибкость визуального программирования;
- средства отладки программ: визуальные и процедурные;
- средства отладки программ: визуальные и объектно-ориентированные.

3. Формирование и развитие навыков использования сред визуального программирования. Используя в качестве обучающего инструмента систему поддержки среды визуального программирования, выбранную на первом этапе и установленную, при необходимости, на компьютере, определите основные компоненты используемого программного продукта и особенности графического пользовательского интерфейса. Подробно изучите структуру проекта, типологию файлов в составе проектов и способы их организации.

Работая над дидактическими проектами, предложенными преподавателем информатики, сформируйте необходимые навыки для ввода и редактирования программ, их запуска и отладки. Основные *упражнения*, рекомендуемые для этой цели:

- определение главных компонентов среды визуального программирования;

- объяснение структуры проектов в средах визуального программирования;
- идентификация файлов проекта;
- включение графических команд в программы;
- отладка визуальных программ;
- запуск визуальных программ.

Работа над следующими *тематическими исследованиями* поможет формированию углубленного видения богатого разнообразия средств, предназначенных для визуального программирования:

- преимущества и недостатки сред визуального программирования;
- сравнительный анализ визуальной и традиционной сред программирования;
- представление некоторых широко используемых сред и языков визуального программирования (*Alice, Kodu, Scratch, ToonTalk, Cameleon, Filter Forge* и др.).

4. Формирование и развитие компетенции визуального программирования. Этот этап начнется с изучения графических команд и критериев их классификации. Далее вам необходимо ознакомиться и изучить на небольших примерах средства, обеспечивающие:

- выполнение движения: движение, вращение, контроль наступления определенного события, контроль наступления определенной ситуации;
- настройку внешнего вида: изменение размера, отображение сообщений, отображение значений, изменение объектов, настройка видимости, настройка слоев;
- управление звуком: настройка, запуск, остановка, продолжительность, применение эффектов;
- обработка событий: запуск / остановка программ, действия клавиш, действия кнопок мыши, действия над графическими объектами, управление потоком сообщений;
- управление потоком выполнения: команды выбора, циклические команды, паузы, остановки, клонирование;
- контроль ситуаций: настройка свойств объекта, ввод данных, настройка свойств периферийных устройств, последовательное распределение во времени;
- создание и вызов подпрограмм.

Упражнения, рекомендуемые для данного этапа:

- перемещение и контроль графических объектов;
- отображение сообщений;
- настройка геометрических / цветовых / визуальных / графических свойств объекта;
- интеграция звуковых компонентов (мультимедиа) в визуальные программы;
- управление динамическими графическими объектами с помощью клавиатуры и мыши;
- создание и вызов подпрограмм.

Также на этом этапе начинается разработка проектов со следующими рекомендуемыми темами:

- взаимодействие между двумя и более визуальными персонажами (объектами);
- проигрывание диалогов между персонажами;
- элементарные игры (прототип тенниса, ловля падающих предметов, погони);

- анимированные рассказы.

Ученикам предлагается самостоятельно или под руководством учителя информатики конкретизировать тему проекта, над которым они хотели бы работать, решить, как организовать, индивидуально или в группах, работу над предлагаемым ими проектом и разработать план работы над ним.

5. Обработка внешних данных. За любыми графическими объектами скрываются определенные данные: информация об их форме и внешнем виде, их расположении в пространстве, траектории движения в рабочем пространстве, сообщениях, полученных от других объектов, и т. д. Результаты обработки соответствующих данных определяют поведение объектов во времени и пространстве, взаимодействие между собой и с внешней средой.

Чтобы изучить способы реализации алгоритмов обработки данных, вспомните о простых и составных типах данных, а также об операциях, которые могут быть выполнены с соответствующими данными. Особое внимание уделите специфике преобразования типов и арифметическим, реляционным и логическим операциям, которые можно выполнять с конкретными данными в средах визуального программирования.

Далее мы рекомендуем изучить визуальные объекты для ввода и отображения данных, события и действия, предназначенные для модификации данных: ситуации, возникшие в процессе выполнения программы, команды с клавиатуры, мыши, другого периферийного оборудования.

Упражнения, способствующие развитию навыков обработки данных в визуальных программах:

- объявление и использование переменных, относящихся к простым и составным типам данных;
- использование арифметических, реляционных и логических операций в визуальных программах;
- изменение значений переменных посредством действий и реакций на события (нажатие клавиш и кнопок мыши, получение сообщений от других графических объектов или извне и т. д.);
- отображение результатов путем связывания их с графическим объектом;
- объявление и использование переменных, принадлежащих к составным типам данных;
- поиск информации в структурах данных;
- изменение структуры данных.

Более глубокое понимание специфики обработки данных в случае визуального программирования может быть обеспечено путем проведения *тематического исследования*, в котором будут проанализированы преимущества и недостатки того, как реализовать вычислительные алгоритмы посредством текстового и визуального программирования. Такое *тематическое исследование* также поможет выявить возможности привязки текстовых программ к определенным объектам в составе визуальных программ.

Проекты, рекомендуемые для данного этапа:

- вычисление максимального числа из множества чисел, указанных персонажами (объектами);
- подсчет суммы множества чисел, указанных персонажами;
- нахождение простых чисел из множества чисел, указанных персонажами;
- вычисление наибольшего общего делителя;

– сортировка массивов.

В целом интеграцию знаний и развитие навыков визуального программирования можно обеспечить путем разработки сложных проектов, таких как графические интерфейсы для консольных и веб-приложений, обучающие игры, развлекательные игры, компьютерные обучающие программы, анимационные фильмы (мультфильмы), продукты с дополненной реальностью (рекламные материалы, динамические модели, динамические реконструкции исторических объектов и др.). Разработка таких проектов возможна путем создания команд учеников, в которых, помимо тесного сотрудничества, должно быть четкое разграничение ролей и обязанностей, специализация в соответствии с призванием и интересами каждого ученика.

8.3. Языки разметки гипертекста

Возможно, в предыдущем классе вы изучали дополнительный модуль веб-дизайна. Изучив этот модуль, вы узнали, как создавать веб-документы, используя как приложения общего назначения, такие как офисные приложения, так и специализированные приложения, называемые веб-редакторами. В обоих случаях ученики, возможно будущие веб-дизайнеры, работали с объектами создаваемых веб-страниц, то есть с текстом, изображениями, элементами навигации, вводом информации, воспроизведением аудио- и видеофрагментов и т. д.

И универсальные, и специализированные приложения, предназначенные для разработки веб-документов, которые вы изучали в предыдущих курсах, работают в WYSIWYG (*What You See Is What You Get* — англ., то, что вы видите, это именно то, что вы получаете). В этом режиме веб-разработчик работает напрямую только с объектами веб-страниц, не прибегая к возможностям, предлагаемым языком HTML.

Напомним, что язык HTML (*Hyper Text Markup Language*) был специально разработан для форматирования текстов веб-страниц, вставки в них элементов управления и ссылок на различные объекты (другие веб-страницы, изображения, аудио-, видеофрагменты и т. д.), находящиеся как на локальном компьютере, так и в любом месте в Интернете.

Хотя работа в режиме WYSIWYG намного удобнее для разработчиков веб-страниц, чем разметка текста с помощью HTML, во многих случаях прямое «написание» веб-страниц на языке HTML обеспечивает значительную экономию памяти и дает дизайнеру больше возможностей размещения объектов на странице и их форматирования.

Рекомендуется проводить изучение языка HTML поэтапно, а именно:

1. Разработка простых веб-документов. На этом этапе особое внимание следует уделить пониманию концепции языка HTML, роли маркеров (тегов), способов их вставки в простые неформатированные тексты, отображают программы навигации как веб-страницы. Необходимо просмотреть список маркеров и уточнить их назначение.

Само создание веб-страниц начинается с разработки их общей структуры,

установления объектов, которые будут в них вставлены. На этом этапе будут подробно изучены только инструменты HTML для форматирования текста:

- заголовки;
- абзацы;
- комментарии;
- физические стили;
- логические стили;
- разделители;
- упорядоченные списки;
- неупорядоченные списки;
- списки определений.

Изучение инструментов форматирования текста в составе документов HTML будет проводиться в тесной связи с элементами обработки текста, изученными в гимназических классах. При этом особое внимание необходимо уделить ключевым словам в составе меток, способам организации меток попарно, в начале и в конце. Поскольку репертуар атрибутов тегов очень широк, усилия по обучению будут сосредоточены не на их запоминании, а на формировании навыков использования онлайн- и автономных систем поддержки, которые могут предоставить подробную информацию о каждом теге и о каждом из его атрибутов.

Рекомендуемые *практические упражнения* для этого этапа включают создание веб-страниц, которые для начала содержат только текст, отформатированный так, чтобы выделить суть информации:

- Мой дом/Моя школа/Мой город/Родное село.
- Мой класс/Мой школьный кружок/Моя спортивная секция.
- Студенческий совет / Молодежный совет.
- Волонтерство в моей школе, моем селе/городе.
- Книжный магазин/Библиотека/Дом культуры/Музыкальный салон.
- Музей села/Городской музей.
- История моего села/города.
- Знаменитости из моего села/города.
- Искусство в моей жизни (в сферах: музыка, пластика, декоративное искусство).
- Спорт в моей жизни.
- Здорово быть здоровым.
- Тренажерные залы / фитнес-залы.
- Салоны красоты / Салоны моды.
- Швейные / ремонтные мастерские (автомобили, компьютеры, бытовая техника, аудио, видео).
- Крестьянское хозяйство.
- Магазины (детские товары, школьные товары, хозяйственные товары, одежда).

Если вы ранее уже разрабатывали такие страницы, рекомендуется отображать HTML-файлы в режиме *View Source* (Просмотр исходного кода) и анализировать, как использовать маркеры (теги) для форматирования входящих в них текстов.

2. Создание ссылок. Самый распространенный сервис Интернета — WWW (*World Wide Web* — Всемирная сеть [паука]) основан на документах HTML, связанных между собой, и внешних по отношению к ним файлах. Переход от

одного документа HTML к другому и отображение внешних файлов в программе навигации осуществляются с помощью специальных ссылок, называемых в последнее время линками (от англ. *link*).

В документе HTML ссылки создаются путем связывания URL-адресов с активными областями страницы. Активные области могут быть фрагментами текста или изображений, а щелчок по такой области требует, чтобы программа навигации нашла связанный ресурс на локальном компьютере или в виртуальном пространстве, в зависимости от случая, и отобразила его на локальном компьютере.

Рекомендуется, чтобы процесс изучения того, как создавать ссылки, выполнялся следующим образом:

- ознакомление с используемой терминологией: привязка, URL, внутренняя ссылка, внешняя ссылка, путь, комментарий к ссылке;
- развитие возможностей создания / выбора активных областей HTML-документов, состоящих как из текстовых фрагментов, так и из изображений;
- доработка с использованием форматов маркеров, предназначенных для создания ссылок;
- создание ссылок на внешний документ, на документ, находящийся в том же или другом каталоге на локальном компьютере;
- создание ссылок на сайт;
- создание ссылок на фрагмент того же или другого HTML-документа;
- создание средств для запуска приложения для обмена сообщениями;
- создание средств копирования файлов.

На начальном этапе в качестве HTML-документов, в которых будут создаваться ссылки, используются файлы, предложенные преподавателем, а после развития необходимых навыков, путем создания ссылок в документах HTML, разработанных учениками в проектах, предложенных для практической деятельности на предыдущем этапе (Мой дом, Мой класс, Студенческий совет, Волонтерство в моей школе и т. д.).

3. Вставка мультимедийных объектов. Первоначально HTML-страницы, отображаемые на экране, содержали только текст. Позже, с развитием информационных и коммуникационных технологий, в них начали появляться ссылки на различные мультимедийные объекты: статические рисунки, анимированные изображения, фотографии, аудио-, видеофрагменты и т. д.

Процесс изучения того, как вставлять ссылки на мультимедийные объекты в HTML-документы, будет состоять из следующих шагов:

- знакомство с форматами графических, аудио- и видеофайлов;
- организация файлов, содержащих мультимедийные объекты, в каталогах, связанных с разрабатываемыми HTML-документами;
- ознакомление со свойствами мультимедийных объектов, актуальными для их вставки в HTML-документы: размеры, границы, комментарии, элементы управления, ссылки;
- изучение специфики интегрируемых мультимедийных объектов: управляющих элементов, интеграционных атрибутов;
- вставка в HTML-документы изображений, звуковых фрагментов, видеофрагментов;
- настройка свойств мультимедийных объектов, вставляемых в HTML-документы.

На начальном этапе отработка практических навыков вставки мультимедийных объектов в HTML-документы будет осуществляться с использованием HTML-файлов и мультимедийных объектов, предложенных преподавателем. В дальнейшем развитие этих навыков будет достигаться в рамках проектов, разработанных учениками, ориентировочная тема которых указана ранее, в соответствующем списке первого этапа. С этой целью ученики самостоятельно, индивидуально или в группах/командах подготовят те мультимедийные объекты (изображения, фотографии, звуковые фрагменты, видеофрагменты), которые значительно помогут развить предложенную тему.

В процессе создания и/или выбора мультимедийных объектов, используемых для включения в состав HTML-документов, будут строго соблюдаться правила цифровой этики и авторские права. Плагиат исключён!

4. Организация контента. Современные версии языка HTML предлагают несколько возможностей для организации контента. В рамках лицейской программы будем изучать только организацию контента с помощью таблиц. С этой целью мы рекомендуем ученикам:

- освежить свои знания о таблицах: заголовок, строка, столбец, ячейка, заголовок строки, заголовок столбца, свойства ячейки, свойства содержимого ячейки;
- ознакомиться с маркерами, используемыми для создания и форматирования таблиц и их атрибутами;
- организовать содержимое разрабатываемых веб-страниц, используя макет страницы с помощью таблиц;
- придать веб-страницам дизайн, который будет выделять информацию, необходимую передать путем редактирования соответствующих таблиц: объединение и разделение ячеек, применение цветовых и градиентных эффектов, установка свойств границ.

Как и на предыдущих этапах, первоначально маркеры таблиц и их атрибуты будут изучаться с использованием очень простых документов HTML, предложенных учителем или созданных учениками самостоятельно, а затем — в проектах, разработанных учениками индивидуально или в группах/командах.

5. Тестирование документов HTML. Для начала документ HTML будет протестирован локально с помощью программы навигации. В процессе тестирования будет проверена доступность всех объектов в HTML-документах, правильность ссылок, работа инструментов навигации, правильность отображения рисунков, правильность воспроизведения аудио- и видеофрагментов.

В случае ошибок необходимо вернуться к этапу создания веб-страниц и объектов, из которых они состоят.

В конце изучения данного модуля ученикам предлагается провести коллективный анализ разработанных ими HTML-документов, используя для этой цели как технические критерии (используемые шрифты и их размер, макет, качество изображения, точность воспроизведения аудио и видео, удобство навигации), так и художественные (оригинальность, актуальность содержания по отношению к передаваемой информации, способ выделения сообщений, соответствие цветов и стилей отображения).