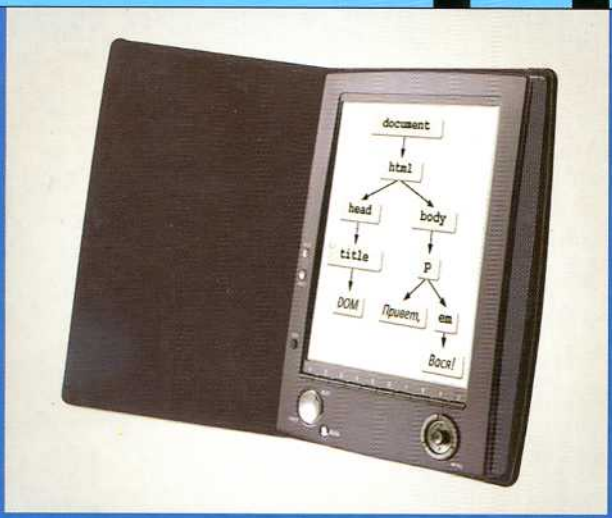
ФГОС

11



К. Ю. Поляков
Е. А. Еремин

ИНФОРМАТИКА

1

УГЛУБЛЕННЫЙ УРОВЕНЬ

ИЗДАТЕЛЬСТВО БИНОМ

ФГОС

К. Ю. Поляков, Е. А. Еремин

ИНФОРМАТИКА

УГЛУБЛЕННЫЙ УРОВЕНЬ

Учебник для 11 класса

в 2-х частях

Часть 1

Рекомендовано
Министерством образования и науки
Российской Федерации
к использованию в образовательном процессе
в имеющих государственную аккредитацию
и реализующих образовательные программы
общего образования образовательных учреждениях

Москва
БИНОМ. Лаборатория знаний
2013

УДК 004.9
ББК 32.97
П54

Поляков К. Ю.

П54 Информатика. Углубленный уровень : учебник для 11 класса : в 2 ч. Ч. 1 / К. Ю. Поляков, Е. А. Еремин. — М. : БИНОМ. Лаборатория знаний, 2013. — 240 с. : ил.

ISBN 978-5-9963-1418-8 (Ч. 1)
ISBN 978-5-9963-1153-8

Учебник предназначен для изучения курса информатики на углубленном уровне в 11 классах общеобразовательных учреждений. Содержание учебника является продолжением курса 10 класса и опирается на изученный в 7–9 классах курс информатики для основной школы.

Рассматриваются вопросы передачи информации, информационные системы и базы данных, разработка вебсайтов, компьютерное моделирование, методы объектно-ориентированного программирования, компьютерная графика и анимация.

Учебник входит в учебно-методический комплект (УМК), включающий в себя также учебник для 10 класса и компьютерный практикум.

Предполагается широкое использование ресурсов портала Федерального центра электронных образовательных ресурсов (http://fcior.edu.ru/).

Соответствует Федеральному государственному образовательному стандарту среднего (полного) общего образования (2012 г.).

УДК 004.9
ББК 32.97

Учебное издание

Поляков Константин Юрьевич
Еремин Евгений Александрович

ИНФОРМАТИКА.
УГЛУБЛЕННЫЙ УРОВЕНЬ
Учебник для 11 класса

В двух частях
Часть первая

Ведущий редактор *О. Полежаева*
Ведущие методисты *И. Сретенская, И. Хлобыстова*
Обложка: *И. Марев*. Художественный редактор *Н. Новак*
Иллюстрации: *Я. Соловцова, Ю. Белаш*
Технический редактор *Е. Денюкова*. Корректор *Е. Клитина*
Компьютерная верстка: *В. Носенко*

Подписано в печать 28.03.13. Формат 70×100/16.
Усл. печ. л. 19,50. Тираж 10 000 экз. Заказ № 34084.

Издательство «БИНОМ. Лаборатория знаний»
125167, Москва, проезд Аэропорта, д. 3
Телефон: (499) 157-5272, e-mail: binom@Lbz.ru
http://www.Lbz.ru, http://e-umk.Lbz.ru, http://metodist.Lbz.ru

При участии ООО Агентство печати «Столица»
www.apstolica.ru; e-mail: apstolica@bk.ru

Отпечатано в соответствии с качеством предоставленных издательством электронных носителей в ОАО «Саратовский полиграфкомбинат».
410004, г. Саратов, ул. Чернышевского, 59. www.sarpk.ru

ISBN 978-5-9963-1418-8 (Ч. 1)
ISBN 978-5-9963-1153-8

Оглавление

От авторов

Перед вами учебник по информатике углублённого уровня для 11 класса. Так же как и в учебнике для 10 класса, в некоторых главах мы рассмотрим вроде бы знакомые вопросы, но несколько с другой точки зрения, более глубоко. Например, в первой главе (она называется «Информация и информационные процессы») вы узнаете, как связано количество информации с теорией вероятностей, как работают упаковщики (программы для сжатия данных) и как можно построить код, позволяющий исправлять ошибки при передаче данных.

Большое внимание в учебнике уделено вопросам алгоритмизации и программирования. Этому посвящены три главы: «Элементы теории алгоритмов», «Алгоритмизация и программирование» и «Объектно-ориентированное программирование». В первой из них вы узнаете, как и зачем ввели строгое математическое понятие «алгоритм», как сравнивать качество алгоритмов и как доказывать правильность программ. Вторая глава знакомит с программированием структур данных, с некоторыми из них (например, со списками, графами и деревьями) вы уже работали в 10 классе. В последней главе объясняются основы объектного подхода к разработке программ, который применяется в крупных промышленных проектах. В практической части используется объектная версия языка Паскаль, реализованная в свободно распространяемом компиляторе *FreePascal*, и среда быстрой разработки *Lazarus*.

В конце каждого параграфа есть вопросы и задания, которые помогут понять, хорошо ли вы усвоили материал. В тексте нет прямых ответов на некоторые вопросы, но есть вся необходимая информация для ответа на них.

Задачи в конце параграфов помогут закрепить материал на практических работах. Самые сложные задачи (на взгляд авторов) отмечены звёздочкой (*).

Мы старались сделать так, чтобы содержание учебника как можно меньше зависело от программного обеспечения, установленного на ваших компьютерах. Весь курс можно успешно изучать, используя только свободное программное обеспечение (СПО) — операционную систему *Linux*, офисный пакет *OpenOffice.org* или его модификации (например, *LibreOffice*), компилятор *FreePascal*, графический редактор *Gimp*, программу трёхмерного моделирования *Blender* и др.

В заключение нам хочется поблагодарить наших коллег, которые взяли на себя труд прочитать предварительные версии отдельных глав учебника и высказать множество полезных замечаний, позволивших сделать учебник более точным, ясным и понятным:

- *А. П. Шестакова*, кандидата педагогических наук, заведующего кафедрой информатики и вычислительной техники Пермского государственного педагогического университета, который вдохновил авторов на написание этого учебника;
- *М. А. Ройтберга*, доктора физико-математических наук, заведующего лабораторией прикладной математики Института математических проблем биологии РАН, г. Пущино;
- *С. С. Михалковича*, кандидата физико-математических наук, доцента кафедры алгебры и дискретной математики Южного федерального университета, г. Ростов-на-Дону;
- *Е. В. Андрееву*, кандидата физико-математических наук, заведующую кафедрой информатики СУНЦ МГУ, г. Москва;
- *О. А. Тузову*, учителя информатики школы № 550, г. Санкт-Петербург;
- *А. Г. Тамаревскую*, учителя информатики лицея № 533, г. Санкт-Петербург;
- *Н. Д. Шумилину*, кандидата педагогических наук, учителя информатики МОУ «Тверская гимназия № 6», г. Тверь;
- *Л. Б. Кулагину*, учителя информатики ФМЛ № 239, г. Санкт-Петербург;
- *Ю. М. Розенфарба*, учителя информатики и ИКТ МОУ «Межозёрная средняя общеобразовательная школа», Челябинская область;

- *Т. А. Мисаренкова*, учителя информатики школы № 163, г. Санкт-Петербург;
- *К. А. Малеванова*, технического директора компании «ПиН Телеком»;
- *А. Г. Архангельского*, генерального директора ЗАО «АЗет Дизайн»;
- *И. А. Васильевского*, дизайнера, специалиста по 3D-моделированию.

Уважаемые ученики!

В работе с книгой вам помогут навигационные значки:

 — важное утверждение или определение.

 — вопросы и задания к параграфу.

 — дополнительное разъяснение.

 — задания для подготовки к итоговой аттестации.

 — К каждой главе учебника рекомендуются:
1) электронный образовательный ресурс (ЭОР) с сайта Федерального центра образовательных ресурсов (ФЦИОР): http://fcior.edu.ru
Доступ к ЭОР из каталога ФЦИОР:
http://fcior.edu.ru/catalog/meta/4/mc/discipline%20OO/mi/4.06/p/page.html.
Ресурсы размещены в алфавитном порядке, согласно названиям учебных тем;
2) практические работы на методическом сайте издательства lbz.metodist.ru в авторской мастерской К. Ю. Полякова и Е. А. Еремина.

 — Проектное или исследовательское задание.

В ходе выполнения проекта (исследования) вы можете:

- подготовить набор полезных ссылок с использованием веб-ресурсов;
- подготовить небольшое выступление с использованием презентации (5–7 мин.);
- оформить доклад и поместить его на сайт школьной конференции;
- подтвердить полученные результаты расчётами и графиками (диаграммами);
- подготовить видеоролик;
- разместить материалы проекта (исследования) в коллекции обучающих модулей по предмету на сайте школы.

Глава 1
Информация и информационные процессы

§ 1
Количество информации

Формула Хартли

Вы знаете, что при выборе из двух возможных вариантов количество полученной информации равно 1 биту. Если количество вариантов N равно 2^I, то количество информации при выборе одного из них равно I битов. А как вычислить количество информации, если количество вариантов не равно степени числа 2?

Ответить на этот вопрос стало возможно только после того, как вы изучили логарифмы в курсе математики. Из формулы

$$N = 2^I$$

сразу следует, что I — это степень, в которую нужно возвести 2, чтобы получить N, т. е. *логарифм*:

$$I = \log_2 N.$$

Эта формула называется **формулой Хартли** в честь американского инженера Ральфа Хартли, который предложил её в 1928 г.

Ральф Хартли (1888–1979)

Пусть, например, на лётном поле стоят 10 самолётов (с номерами от 1 до 10) и известно, что один из них летит в Санкт-Петербург. Сколько информации в сообщении «Самолёт № 2 летит в Санкт-Петербург»? У нас есть 10 вариантов, из которых выбирается один, поэтому по формуле Хартли количество информации равно

$$I = \log_2 10 \approx 3{,}322 \text{ бита}.$$

Обратите внимание, что для значений N, которые не равны целой степени числа 2, количество информации в битах — дробное число.

С помощью формулы Хартли можно вычислить теоретическое количество информации в сообщении. Предположим, что алфавит (полный набор допустимых символов) включает 50 символов (в этом случае говорят, что **мощность алфавита** равна 50). Тогда информация при получении каждого символа составляет

$$I = \log_2 50 \approx 5{,}644 \text{ бита}.$$

Если сообщение содержит 100 символов, его общий информационный объём примерно равен

$$5{,}644 \cdot 100 = 564{,}4 \text{ бита}.$$

В общем случае объём сообщения, использующего алфавит из N символов и состоящего из M символов, равен

$$I = M \cdot \log_2 N.$$

Такой **подход** к определению количества информации называют **алфавитным**. Конечно, на практике невозможно использовать для кодирования символа нецелое число битов, поэтому используют первое целое число, которое больше теоретически рассчитанного значения. Например, при использовании алфавита из 50 символов каждый символ будет закодирован с помощью 6 битов ($50 \le 2^6 = 64$).

Сколько разных сообщений можно передать, если известен алфавит и длина сообщения? Предположим, что для кодирования сообщения используются 4 буквы, например «А», «Б», «В» и «Г», и сообщение состоит из двух символов. Поскольку каждый символ может быть выбран 4 разными способами, на каждый вариант выбора первого символа есть 4 варианта выбора второго. Поэтому общее число разных двухбуквенных сообщений вычисляется как $4 \cdot 4 = 4^2 = 16$. Если в сообщение добавить ещё один символ, то для каждой из 16 комбинаций первых двух символов третий можно выбрать четырьмя способами, так что число разных трёхсимвольных сообщений равно $4 \cdot 4 \cdot 4 = 4^3 = 64$.

В общем случае, если используется алфавит из N символов и сообщение состоит из M символов, то количество разных возможных сообщений равно

$$K = N^M.$$

Задачи

1. В русском лото 99 бочонков. Какое количество информации содержится в сообщении «Первым вытащили бочонок с номером 16»?

2. В классе 25 учеников. Какое количество информации содержится в сообщении «Сегодня дежурит Василий Иванов»?
3. В чём состоит алфавитный подход к оценке количества информации?
4. Оцените теоретическое количество информации в сообщении «Приеду в четверг». Используемый алфавит состоит из заглавных и строчных русских букв и пробела.
5. Каков будет фактический объём сообщения «Приеду в четверг», если при передаче каждый символ кодируется минимально возможным целым числом битов?
6. В некоторой стране автомобильный номер длиной 7 символов составляется из заглавных букв (всего используется 26 букв) и десятичных цифр в любом порядке. Каждый символ кодируется одинаковым и минимально возможным количеством битов, а каждый номер — одинаковым и минимально возможным количеством байтов. Определите объём памяти, необходимый для хранения 20 автомобильных номеров.
7. Объём сообщения, содержащего 4096 символов, равен 1/512 части мегабайта. Какова мощность алфавита, с помощью которого записано это сообщение?
8. Какое наименьшее число символов должно быть в алфавите, чтобы с помощью всевозможных трёхбуквенных слов, состоящих из символов данного алфавита, можно было передать не менее 9 различных сообщений?

Информация и вероятность

Всё, что написано в предыдущем пункте, верно только при одном уточнении: все события (символы алфавита) одинаково ожидаемы, т. е. нельзя заранее сказать, что какой-то символ встречается чаще, а какой-то — реже. В реальности это предположение не всегда верно. Например, в тексте на русском языке некоторые символы встречаются часто, а некоторые — очень редко. Числа во втором столбце табл. 1.1 означают относительную долю символа в больших текстах. Например, доля 0,175 для пробела означает, что примерно 17,5% всех символов в текстах на русском языке — пробелы.

Таблица 1.1

Пробел	0,175
О	0,090
Е, Ё	0,072
А	0,062
И	0,062
Т	0,053
Н	0,053
С	0,045
Р	0,040
В	0,038
Л	0,035
К	0,028
М	0,026
Д	0,025
П	0,023
У	0,021
Я	0,018
Ы	0,016
З	0,016
Ь, Ъ	0,014
Б	0,014
Г	0,013
Ч	0,012
Й	0,010
Х	0,009
Ж	0,007
Ю	0,006
Ш	0,005
Ц	0,004
Щ	0,003
Э	0,003
Ф	0,002

Иногда среди всех возможных событий есть ожидаемые и неожиданные. Например, на вопрос «Идёт ли сейчас снег?» летом мы ожидаем услышать ответ «Нет». При этом ответ «Да» будет очень неожиданным, и после его получения наши дальнейшие планы могут сильно измениться. Это значит, что при таком ответе мы получаем гораздо больше информации. Как её измерить точно?

Сначала нужно разобраться с тем, что значит «менее ожидаемое» событие и «более ожидаемое». Математики в этом случае используют понятие **«вероятность»**: если событие более ожидаемое, то его вероятность (точнее, вероятность того, что оно произойдёт) больше.

Вероятность — это число в интервале от 0 до 1. В математике вероятность принято обозначать буквой p (от латинского *probabilis* — вероятный, возможный).

Сначала рассмотрим предельные случаи, когда вероятность равна 0 или 1. Пусть, например, x — некоторое неизвестное вещественное число, которое задумал ведущий. Вы знаете, что для любого вещественного x *всегда* $x^2 \geq 0$. В этом случае считают, что вероятность события $x^2 \geq 0$ равна 1 (событие $x^2 \geq 0$ обязательно произойдёт). Часто вероятность измеряют в процентах от 1, тогда вероятность события $x^2 \geq 0$ равна 100%. В то же время вероятность события $x^2 < 0$ равна нулю, это значит, что событие $x^2 < 0$ никогда не произойдёт.

Теперь предположим, что мы бросаем монету и смотрим, какой стороной она упала, «орлом» или «решкой». Если повторять этот опыт много раз, мы заметим, что количество «орлов» и «решек» примерно равно (конечно, если монета не имеет дефектов). При этом вероятность каждого из двух событий равна 0,5, или 50%. Скорее всего, вы слышали выражение «50 на 50», которое означает, что ни одному из двух вариантов нельзя отдать предпочтение — их вероятности равны.

Вероятность события можно определить с помощью большого количества испытаний. Если из N испытаний нужное нам событие случилось m раз, то вероятность такого события можно оценить как $\frac{m}{N}$.

Например, классический игральный кубик имеет 6 граней; если кубик качественный, вероятность выпадения каждой грани равна 1/6. Вероятность выпадения чётного числа можно подсчитать так: среди чисел от 1 до 6 всего

3 чётных числа, поэтому при большом числе испытаний в половине случаев (в 3 из 6) будут выпадать чётные числа, т. е. вероятность равна 0,5. А вероятность выпадения числа, меньшего 3, равна $2/6 = 1/3 \approx 0{,}33$, потому что только 2 из 6 чисел (1 и 2) удовлетворяют условию.

Теперь можно переходить к главному вопросу: как вычислить количество информации, если в сообщении получен символ, вероятность появления которого равна p. Попробуем сначала определить, какими свойствами должна обладать эта величина, исходя из «здравого смысла».

Во-первых, чем меньше вероятность, тем более неожидан символ и тем больше информации мы получили. Если вероятность события близка к нулю, количество информации должно стремиться к бесконечности (получение такого символа очень неожиданно).

Во-вторых, представим себе, что мы получаем символы только одного вида, например только буквы «А». Тогда вероятность появления символа «А» равна 1 и никакой новой информации в этом символе для нас нет — мы всё заранее знали. Следовательно, при $p = 1$ информация должна быть равна нулю.

В-третьих, «здравый смысл» подсказывает, что когда мы бросаем игральный кубик два (три, 102, 1002) раза, мы получаем информации в два (три, 102, 1002) раза больше, чем при однократном бросании кубика. Это свойство называют принципом *аддитивности* (сложения).

Математики доказали, что этими свойствами обладает только логарифмическая функция вида $f(p) = -K \cdot \log_2 p$, где коэффициент K можно выбирать произвольно, как удобно в конкретной задаче. Если взять $K = 1$, мы получим количество информации в битах.

Если событие имеет вероятность p, то количество информации в битах, полученное в сообщении об этом событии, равно

$$I = -\log_2 p = \log_2 \frac{1}{p}. \qquad (1)$$

Поскольку вероятность p не больше 1, количество информации не может быть меньше нуля. Легко проверить, что если вероятность равна 1, то количество полученной информации равно нулю. Если вероятность стремится к 0, величина $\log_2 p$ стремится к $-\infty$, а количество информации — к ∞.

Третье свойство (аддитивность, сложение вероятностей) проверим на примере. Пусть есть два мешка, в каждом из которых лежат 8 шариков разного цвета. Вычислим, какое количество информации мы получили из сообщения «Из первого мешка вытащили (наугад) красный шарик, а из второго — зелёный». Из каждого мешка можно вытащить один из восьми шариков, т. е. вероятность вытащить какой-то определённый шарик равна 1/8. Следовательно, количество информации в сообщении «Из первого мешка вытащили красный шарик» равно

$$I = \log_2 \frac{1}{\frac{1}{8}} = \log_2 8 = 3 \text{ бита.}$$

Точно такую же информацию, 3 бита, мы получаем из сообщения «Из второго мешка вытащили зелёный шарик».

Теперь посмотрим, какое количество информации несёт исходное полное сообщение «Из первого мешка вытащили (наугад) красный шарик, а из второго — зелёный». Какова вероятность именно этого исхода? Есть 8 способов вытащить шарик из каждого мешка, всего получаем 8·8 = 64 варианта, поэтому вероятность каждого из них равна 1/64. Тогда количество информации в сообщении равно

$$I = \log_2 \frac{1}{\frac{1}{64}} = \log_2 64 = 6 \text{ битов,}$$

т. е. равно сумме количеств информации в двух отдельных сообщениях. Мы показали, что свойство аддитивности выполняется.

Возможно, вы уже заметили, что последний пример фактически свёлся к использованию формулы Хартли. Если все N вариантов имеют равные вероятности, то вероятность каждого варианта равна $p = \frac{1}{N}$, следовательно, количество информации о любом из возможных событий вычисляется как

$$I = \log_2 \frac{1}{\frac{1}{N}} = \log_2 N.$$

Этот результат совпадает с формулой Хартли.

Однако формулу Хартли нельзя использовать, если вероятности событий разные. Пусть, например, детям раздают воздушные шарики разного цвета, причём 7 из каждых 10 шариков — зелёные. Какое количество информации содержится в сообщении

«Маша получила зелёный шарик»? Здесь формула Хартли неприменима, однако можно использовать формулу (1). Вероятность того, что Маше достался зелёный шарик, равна 7/10, а количество информации вычисляется как

$$I = \log_2 \frac{1}{\frac{7}{10}} = \log_2 \frac{10}{7} \text{ битов.}$$

Величина под знаком двоичного логарифма не является степенью двойки. Как её вычислить? Для этого используется свойство логарифма, позволяющее переходить к другому основанию, например, к десятичным или натуральным логарифмам, которые умеют вычислять калькуляторы:

$$I = \log_2 x = \frac{\lg x}{\lg 2} = \frac{\ln x}{\ln 2} \text{ битов.}$$

В данном случае получаем

$$I = \frac{\lg \frac{10}{7}}{\lg 2} = \frac{\ln \frac{10}{7}}{\ln 2} \approx 0{,}515 \text{ бита.}$$

Вопросы и задания

1. Как вы понимаете термин «вероятность события»?
2. В каких случаях вероятность равна 1? Когда она равна 0?
3. Что неправильно в сообщении: «"Спартак" выиграет с вероятностью 200%»?

Задачи

1. Вероятность появления символа @ в некотором тексте равна 0,125. Сколько битов информации несёт сообщение о том, что очередной символ текста — @?
2. В садке у рыбака сидят 2 окуня, 4 плотвы и 10 уклеек. Не смотря в садок, рыбак вытаскивает наугад одну рыбу. Какова вероятность того, что это будет плотва?
3. В пруду плавают 10 линей, 20 окуней и 70 карасей. Считаем, что они одинаково голодны и равномерно распределены по водоёму. Какова вероятность того, что первая рыба, пойманная рыбаком, будет линем? Окунем? Карасём?
4. Ученик задумал целое число от 1 до 100. Какова вероятность того, что это будет число в интервале от 21 до 30? От 31 до 55? Больше 25? Равно 25?

5. Ученик задумал целое число от –10 до 10. Какова вероятность того, что квадрат этого числа больше 25? Меньше 10? Равен 49?
6. В корзине лежат 8 чёрных шаров и 24 белых. Какова вероятность вытащить чёрный шар? Сколько битов информации несёт сообщение о том, что достали чёрный шар?
7. В корзине лежат 32 клубка шерсти, из них 4 красных. Сколько битов информации несёт сообщение о том, что достали клубок красной шерсти?
8. В коробке лежат 64 цветных карандаша. Сообщение о том, что достали белый карандаш, несёт 4 бита информации. Сколько белых карандашей было в коробке?
9. В ящике лежат чёрные и белые перчатки. Среди них 2 пары чёрных. Сообщение о том, что достали чёрные перчатки, несет 4 бита информации. Сколько всего пар перчаток было в ящике?
10. За контрольную работу в классе из 30 человек выставлено 6 пятёрок, 15 четвёрок, 8 троек и 1 двойка. Сколько битов информации несёт сообщение о том, что Иван Петров получил четвёрку?
11. В ящике лежат 20 шаров, из них 10 чёрных, 5 белых, 4 жёлтых и 1 красный. Сколько битов информации несёт сообщение о том, что достали белый шар?
12. За четверть ученик получил 20 оценок. Сообщение о том, что он вчера получил четвёрку, несёт 2 бита информации. Сколько четвёрок получил ученик за четверть?
13. В корзине лежат чёрные и белые шары. Среди них 18 чёрных шаров. Сообщение о том, что достали белый шар, несёт 2 бита информации. Сколько всего шаров было в корзине?
14. В алфавите языка племени тумба-юмба 4 буквы: гласные О и А, согласные Ш и Щ. Вероятности их появления в тексте: А — 0,35; О — 0,4; Ш — 0,1; Щ — 0,15. Сколько битов информации несёт сообщение о том, что очередной символ текста — согласная?

*15. Автобус № 25 ходит в 2 раза чаще, чем автобус № 13. Сообщение о том, что к остановке подошел автобус № 25, несет 4 бита информации. Сколько битов информации в сообщении «К остановке подошел автобус № 13»?

*16. В зоопарке 32 обезьяны живут в двух вольерах, А и Б. Одна из обезьян заболела. Сообщение «Заболела обезьяна из вольера А» содержит 4 бита информации. Сколько обезьян живут в вольере Б?

Формула Шеннона

Информация играет для нас важную роль потому, что наше знание всегда неполно, в нём есть *неопределённость*. Эта неопределённость мешает нам решать свои задачи, принимать правиль-

ные решения. Полученная информация уменьшает («снимает») неопределённость, полностью или частично. Поэтому количество полученной информации можно оценить по величине уменьшения неопределённости:

$$\Delta H = H_{\text{нач}} - H_{\text{кон}},$$

где $H_{\text{нач}}$ — начальная неопределённость, а $H_{\text{кон}}$ — конечная (после получения сообщения). Если неопределённость полностью снимается, то $H_{\text{кон}} = 0$.

Клод Шеннон (1916–2001)

Чтобы оценить информацию с этой точки зрения, нужно как-то вычислить неопределённость, выразить её числом. Эту задачу решил в 1948 г. американский математик Клод Шеннон.

Пусть неопределённость состоит в том, что мы можем получить одно из N возможных сообщений, причём известно, что вероятность получения сообщения с номером i равна p_i.

Неопределённость знания об источнике данных вычисляется по формуле Шеннона

$$H = -\sum_{i=1}^{N} p_i \cdot \log_2 p_i = \sum_{i=1}^{N} p_i \cdot \log_2 \frac{1}{p_i}.$$

Величина H часто называется *информационной энтропией*. С точки зрения математики это *среднее количество информации*, которую мы получаем при полном снятии неопределённости (когда выбран один из возможных вариантов).

Когда неопределённость наибольшая? Зададим вопрос «Идёт ли сейчас снег?» зимой и летом. Летом неопределённость очень маленькая, так как, скорее всего, снега нет, ситуация ясна. Зимой же неопределённость велика, потому что снег может идти или не идти примерно с равной вероятностью.

Перейдём к числам. Будем считать, что вероятность снега зимой равна $p_1 = 0{,}5$. Чему равна вероятность p_2 того, что снега нет? «Здравый смысл» подсказывает, что $p_2 = 0{,}5$ (остальные 50%). Математики говорят, что два события, «Снег идёт» и «Снега нет», составляют **полную систему**. Это значит, что обязательно случится какое-нибудь одно из этих событий, и при этом другое точно не произойдёт. Слово «обязательно» означает, что вероятность этих двух событий в сумме равна 1.

Сумма вероятностей всех событий, составляющих полную систему, равна 1.

Для «зимнего» случая количество информации при получении сообщений «Снег идёт» и «Снега нет» одинаковое, потому что их вероятности одинаковые:

$$I_1 = I_2 = \log_2 \frac{1}{0{,}5} = 1 \text{ бит}.$$

Неопределённость, вычисленная по формуле Шеннона, также равна 1 биту:

$$H = p_1 \cdot \log_2 \frac{1}{p_1} + p_2 \cdot \log_2 \frac{1}{p_2} = 0{,}5 \cdot \log_2 \frac{1}{0{,}5} + 0{,}5 \cdot \log_2 \frac{1}{0{,}5} = \log_2 2 = 1 \text{ бит}.$$

Для лета будем считать вероятность выпадения снега равной $p_1 = 0{,}0001$. Тогда вероятность того, что снега нет, равна $p_2 = 1 - 0{,}0001 = 0{,}9999$. Сообщения «Снег идёт» и «Снега нет» несут разное количество информации:

$$I_1 = \log_2 \frac{1}{0{,}0001} = 13{,}29 \text{ бита}, \quad I_2 = \log_2 \frac{1}{0{,}9999} = 0{,}00014 \text{ бита},$$

а неопределённость (среднее количество информации) равна

$$H = 0{,}0001 \cdot \log_2 \frac{1}{0{,}0001} + 0{,}9999 \cdot \log_2 \frac{1}{0{,}9999} \approx 0{,}0015 \text{ бита}.$$

Мы получили то, что ожидали: зимой неопределённость в ответе на вопрос «Идёт ли сейчас снег?» значительно больше, чем летом. Можно предположить (и это действительно так), что неопределённость наибольшая в том случае, когда вероятности всех событий равны.

Летом эта неопределённость очень близка к нулю, поэтому можно предположить (и это также верно), что она стремится к нулю, если вероятность одного из двух событий стремится к нулю.

На рисунке 1.1 построена зависимость величины неопределённости H от вероятности первого события p_1. При этом вероятность второго события определяется как $p_2 = 1 - p_1$, так как они составляют полную систему. Главный вывод этого примера таков:

Неопределённость наибольшая для случая, когда все события равновероятны.

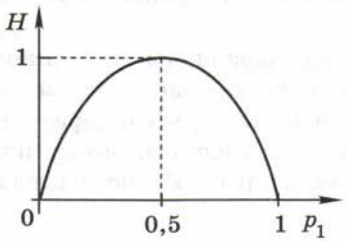


Рис. 1.1

При этом вероятность каждого из N событий равна $p = \frac{1}{N}$, поэтому по формуле Шеннона

$$H = \sum_{i=1}^{N} p \cdot \log_2 \frac{1}{p} = \sum_{i=1}^{N} \frac{1}{N} \cdot \log_2 N = \log_2 N.$$

Отсюда следует, что:

При равновероятных событиях неопределённость совпадает с количеством информации, вычисленной по формуле Хартли.

Вопросы и задания

1. В чём заключается неопределённость?
2. Как связана неопределённость с информацией, которую мы получаем при сообщении об отдельных событиях?
3. Что такое полная система событий?
4. Что можно сказать о вероятностях событий, входящих в полную систему?
5. В каком случае неопределённость наибольшая?
6. В каком случае неопределённость стремится к нулю?
7. В каком случае неопределённость совпадает с количеством информации, вычисленным по формуле Хартли?

Задачи

1. Из аэропорта Куково можно улететь на самолетах Ту-154, АН-148, Боинг-737. Вероятность полёта на самолёте Ту-154 равна 0,6; вероятность полёта на Боинге-737 равна 0,1. Чему равна вероятность полёта на АН-148?
2. В коробке 3 красных карандаша и 7 синих. Чему равна неопределённость при выборе наугад одного карандаша из коробки?
3. На улице Строителей из 20 домов 6 деревянных, 8 сделаны из кирпича, а оставшиеся — из железобетонных плит. Чему равна неопределённость ответа на вопрос «Из чего сделан дом № 16 на улице Строителей»?

§ 2
Передача данных

Скорость передачи данных

Скорость передачи данных — это количество битов (байтов, Кбайт и т. д.), которое передаётся по каналу связи за единицу времени (например, за 1 с).

Пропускная способность любого реального канала связи ограничена. Это значит, что есть некоторая наибольшая возможная скорость передачи данных, которую принципиально невозможно превысить.

Основная единица измерения скорости — биты в секунду (бит/с, англ. **bps** — *bits per second*). Для характеристики быстродействующих каналов применяют килобиты в секунду (Кбит/с) и мегабиты в секунду (Мбит/с), иногда используют байты в секунду (байт/с) и килобайты в секунду (Кбайт/с).

Информационный объём I данных, переданных по каналу за время t, вычисляется по формуле $I = v \cdot t$, где v — скорость передачи данных. Например, если скорость передачи данных равна 512 000 бит/с, за 1 минуту можно передать файл объёмом

$$512\,000 \text{ бит/с} \cdot 60 \text{ с} = 30\,720\,000 \text{ битов} = 3\,840\,000 \text{ байтов} = 3075 \text{ Кбайт}.$$

Обнаружение ошибок

В реальных каналах связи всегда присутствуют помехи, искажающие сигнал. В некоторых случаях ошибки допустимы, например, при прослушивании радиопередачи через Интернет небольшое искажение звука не мешает понимать речь. Однако чаще всего требуется обеспечить точную передачу данных. Для этого в первую очередь нужно определить факт возникновения ошибки и, если это произошло, передать блок данных ещё раз.

Представьте себе, что получена цепочка нулей и единиц 1010101110, причём все биты независимы. В этом случае нет абсолютно никакой возможности определить, верно ли передана последовательность. Поэтому необходимо вводить **избыточность** в передаваемое сообщение (включать в него «лишние» биты) только для того, чтобы обнаружить ошибку.

Простейший вариант — добавить 1 бит в конце блока данных, который будет равен 1, если в основном сообщении нечётное число единиц, и равен 0 для сообщения с чётным числом единиц. Этот дополнительный бит называется **битом чётности**. Бит чётности используется при передаче данных в сетях, проверка чётности часто реализуется аппаратно (с помощью электроники).

Например, пусть требуется передать два бита данных. Возможны всего 4 разных сообщения: 00, 01, 10 и 11. Первое и четвёртое из них содержат чётное число единиц (0 и 2), значит, бит чётности для них равен 0. Во втором и третьем сообщениях нечётное число единиц (1), поэтому бит чётности будет равен 1. Таким образом, сообщения с добавленным битом чётности будут выглядеть так:

$$00\underline{0},\ 01\underline{1},\ 10\underline{1},\ 11\underline{0}.$$

Первые два бита несут полезную информацию, а третий (подчёркнутый) — вспомогательный, он служит только для обнаружения ошибки. Обратим внимание, что каждое из этих трёхбитных сообщений содержит чётное число единиц.

Подумаем, сколько ошибок может обнаружить такой метод. Если при передаче неверно передан только один из битов, количество единиц в сообщении стало нечётным, это и служит признаком ошибки при передаче. Однако исправить ошибку нельзя, потому что непонятно, в каком именно разряде она случилась.

Если же изменилось два бита, чётность не меняется, и такая ошибка не обнаруживается. В длинной цепочке применение бита

чётности позволяет обнаруживать нечётное число ошибок (1, 3, 5, ...), а ошибки в чётном количестве разрядов остаются незамеченными.

Контроль с помощью бита чётности применяется для небольших блоков данных (чаще всего — для каждого отдельного байта) и хорошо работает тогда, когда отдельные ошибки при передаче независимы одна от другой и встречаются редко.

Для обнаружения искажений в передаче файлов, когда может сразу возникнуть множество ошибок, используют другой метод — вычисляют контрольную сумму с помощью какой-нибудь хэш-функции (вспомните материал учебника для 10 класса). Чаще всего для этой цели применяют алгоритмы **CRC** (англ. *Cyclic Redundancy Code* — циклический избыточный код), а также криптографические хэш-функции MD5, SHA-1 и другие. Если контрольная сумма блока данных, вычисленная приёмником, не совпадает с контрольной суммой, записанной передающей стороной, то произошла ошибка.

Помехоустойчивые коды

Значительно сложнее исправить ошибку сразу (без повторной передачи), однако в некоторых случаях и эту задачу удаётся решить. Для этого требуется настолько увеличить избыточность кода (добавить «лишние» биты), что небольшое число ошибок всё равно позволяет достаточно уверенно распознать переданное сообщение. Например, несмотря на помехи в телефонной линии, обычно мы легко понимаем собеседника. Это значит, что речь обладает достаточно большой избыточностью, и это позволяет исправлять ошибки «на ходу».

Пусть, например, нужно передать один бит, 0 или 1. Утроим его, добавив ещё два бита, совпадающих с первым. Таким образом, получаются два «правильных» сообщения:

000 и 111.

Теперь посмотрим, что получится, если при передаче одного из битов сообщения 000 произодёт ошибка и приёмник получит искажённое сообщение 001. Заметим, что оно отличается одним битом от 000 и двумя битами от второго возможного варианта — 111. Значит, скорее всего, произошла ошибка в последнем бите и сообщение нужно исправить на 000. Если приёмник получил 101, можно точно сказать, что произошла ошибка, однако попытка исправить её приведёт к неверному варианту, так как ближайшая

«правильная» последовательность — это 111. Таким образом, такой код *обнаруживает* одну или две ошибки. Кроме того, он позволяет *исправить* (!) одну ошибку, т. е. является помехоустойчивым.

Помехоустойчивый код — это код, который позволяет исправлять ошибки, если их количество не превышает некоторого уровня.

Выше мы фактически применили понятие «расстояния» между двумя кодами. В теории передачи информации эта величина называется расстоянием Хэмминга в честь американского математика Р. Хэмминга.

Расстояние Хэмминга — это количество позиций, в которых различаются два закодированных сообщения одинаковой длины.

Например, расстояние d между кодами 001 и 100 равно

$$d(\underline{0}0\underline{1}, \underline{1}0\underline{0}) = 2,$$

потому что они различаются в двух битах (эти биты подчёркнуты). В приведённом выше примере расстояние между «правильными» последовательностями (*словами*) равно $d(000, 111) = 3$. Такой код позволяет обнаружить одну или две ошибки и исправить одну ошибку.

В общем случае, если минимальное расстояние между «правильными» словами равно d, можно обнаружить от 1 до $d - 1$ ошибок, потому что при этом полученный код будет отличаться от всех допустимых вариантов. Для исправления r ошибок необходимо, чтобы выполнялось условие

$$d \geq 2r + 1.$$

Это значит, что слово, в котором сделано r ошибок, должно быть ближе к исходному слову (из которого оно получено искажением), чем к любому другому.

Рассмотрим более сложный пример. Пусть нужно передавать три произвольных бита, обеспечив обнаружение двух любых ошибок и исправление одной ошибки. В этом случае можно использо-

вать, например, такой код с тремя контрольными битами (они подчёркнуты):

000 000	100 101
001 111	101 010
010 011	110 110
011 100	111 001

Расстояние Хэмминга между любыми двумя словами в таблице не менее 3, поэтому код обнаруживает две ошибки и позволяет исправить одну. Как же вычислить ошибочный бит?

Предположим, что было получено кодовое слово 011011. Определив расстояние Хэмминга до каждого из «правильных» слов, находим единственное слово 010011, расстояние до которого равно 1 (расстояния до остальных слов больше). Значит, скорее всего, это слово и было передано, но исказилось из-за помех.

На практике используют несколько более сложные коды, которые называются **кодами Хэмминга**. В них информационные и контрольные биты перемешаны, и за счёт этого можно сразу, без перебора, определить номер бита, в котором произошла ошибка. Наиболее известен семибитный код, в котором 4 бита — это данные, а 3 бита — контрольные. В нём минимальное расстояние между словами равно 3, поэтому он позволяет обнаружить две ошибки и исправить одну.

Вопросы и задания

1. В каких единицах измеряют скорость передачи данных?
2. Почему для любого канала связи скорость передачи данных ограничена?
3. Как вычисляется информационный объём данных, который можно передать за некоторое время?
4. В каких случаях при передаче данных допустимы незначительные ошибки?
5. Что такое избыточность сообщения? Для чего её можно использовать? Приведите примеры.
6. Как помехи влияют на передачу данных?
7. Что такое бит чётности? В каких случаях с помощью бита чётности можно обнаружить ошибку, а в каких — нельзя?
8. Можно ли исправить ошибку, обнаружив неверное значение бита чётности?
9. Для чего используется метод вычисления контрольной суммы?

10. Какой код называют помехоустойчивым?
11. Каково должно быть расстояние Хэмминга между двумя кодами, чтобы можно было исправить 2 ошибки?
12. Как исправляется ошибка при использовании помехоустойчивого кода?
13. Сколько ошибок обнаруживает 7-битный код Хэмминга, описанный в конце параграфа, и сколько ошибок он позволяет исправить?

Подготовьте сообщение

а) «Алгоритмы CRC»

б) «Коды Хемминга»

Задачи

1. Через соединение со скоростью 128 000 бит/с передают файл размером 625 Кбайт. Определите время передачи файла в секундах.
2. Передача файла через соединение со скоростью 512 000 бит/с заняла 1 минуту. Определите размер файла в килобайтах.
3. Скорость передачи данных равна 64 000 бит/с. Сколько времени займёт передача файла объёмом 375 Кбайт по этому каналу?
4. У Васи есть доступ к Интернету по высокоскоростному одностороннему радиоканалу, обеспечивающему скорость получения им информации 256 000 бит/с. У Пети нет скоростного доступа к Интернету, но есть возможность получать информацию от Васи по низкоскоростному телефонному каналу со средней скоростью 32 768 бит/с. Петя договорился с Васей, что тот будет скачивать для него данные объёмом 5 Мбайт по высокоскоростному каналу и ретранслировать их Пете по низкоскоростному каналу. Компьютер Васи может начать ретрансляцию данных не раньше, чем им будут получены первые 375 Кбайт этих данных. Каков минимально возможный промежуток времени (в секундах) с момента начала скачивания Васей данных до полного их получения Петей?
5. Определите максимальный размер файла в килобайтах, который может быть передан за 8 минут со скоростью 32 000 бит/с.
6. Сколько секунд потребуется, чтобы передать 400 страниц текста, состоящего из 30 строк по 60 символов каждая по линии со скоростью 128 000 бит/с, при условии что каждый символ кодируется 1 байтом?
7. Передача текстового файла по каналу связи со скоростью 128 000 бит/с заняла 1,5 мин. Определите, сколько страниц содержал переданный текст, если известно, что он был представлен в 16-битной кодировке UNICODE, а на одной странице — 400 символов.

8. Передача текстового файла через соединение со скоростью 64 000 бит/с заняла 10 с. Определите, сколько страниц содержал переданный текст, если известно, что он был представлен в 16-битной кодировке UNICODE и на каждой странице — 400 символов.
9. Сколько секунд потребуется, чтобы передать цветное растровое изображение размером 1280 × 800 пикселей по линии со скоростью 256 000 бит/с при условии, что цвет каждого пикселя кодируется 24 битами?
10. Сколько секунд потребуется, чтобы передать цветное растровое изображение размером 1000 × 800 пикселей по линии со скоростью 128 000 бит/с при условии, что цвет каждого пикселя кодируется 24 битами?
11. Через некоторый канал связи за 2 минуты был передан файл, размер которого — 3750 Кбайт. Определите минимальную скорость, при которой такая передача возможна.
12*. Модем, передающий информацию со скоростью 256 000 бит/с, передал файл с несжатой стереофонической музыкой за 2 минуты 45 секунд. Найдите разрядность кодирования этой музыки, если известно, что её продолжительность составила 1 минуту и оцифровка производилась с частотой 22 000 Гц.
13. Найдите расстояние между кодами 11101 и 10110, YUIX и YAIY.
14. Найдите все пятизначные двоичные коды, расстояние от которых до кода 11101 равно 1. Сколько всего может быть таких слов для n-битного кода?
15. Для передачи восьмеричных чисел используется помехоустойчивый шестибитный код, приведённый в тексте параграфа. Раскодируйте сообщение, исправив ошибки:

001 101 011 011 011 101 110 101 101 011

§ 3 Сжатие данных

Основные понятия

Для того чтобы сэкономить место на внешних носителях (жёстких дисках, флэш-дисках) и ускорить передачу данных по компьютерным сетям, нужно сжать данные — уменьшить информационный объём, сократить длину двоичного кода. Это можно сделать, устранив **избыточность** использованного кода.

Например, пусть текстовый файл объёмом 10 Кбайт содержит всего четырех различных символа: латинские буквы «A», «B», «C» и пробел. Вы знаете, что для кодирования одного из четырёх возможных вариантов достаточно 2 битов, поэтому использовать для его передачи обычное 8-битное кодирование символов невыгодно. Можно присвоить каждому из четырёх символов двухбитные коды, например, так:

A — 00, B — 01, C — 10, пробел — 11.

Тогда последовательность «ABA CABABA», занимающая 10 байтов в однобайтной кодировке, может быть представлена как цепочка из 20 битов:

00010011100001000100

Таким образом, нам удалось уменьшить информационный объём текста в 4 раза, и передаваться он будет в 4 раза быстрее. Однако непонятно, как раскодировать это сообщение, ведь получатель не знает, какой код был использован. Выход состоит в том, чтобы включить в сообщение служебную информацию — заголовок, в котором каждому коду будет сопоставлен ASCII-код символа. Условимся, что первый байт заголовка — это количество используемых символов N, а следующие N байтов — это ASCII-коды этих символов. В данном случае заголовок занимает 5 байтов и выглядит так:

00000100_2	01000001_2	01000010_2	01000011_2	00100000_2
4 символа	A (код 65)	B (код 66)	C (код 67)	пробел (код 32)

Файл, занимающий 10 Кбайт в 8-битной кодировке, содержит 10 240 символов. В сжатом виде каждый символ кодируется двумя битами, кроме того, есть 5-байтный заголовок. Поэтому сжатый файл будет иметь объём

$$5 + 10240 \cdot 2/8 \text{ байтов} = 2565 \text{ байтов}.$$

Коэффициент сжатия — это отношение размеров исходного и сжатого файлов.

В данном случае удалось сжать файл почти в 4 раза, коэффициент сжатия равен

$$k = 10\ 240/2565 \approx 4.$$

Если принимающая сторона «знает» формат файла (заголовок + закодированные данные), она сможет восстановить в точности его исходный вид. Такое сжатие называют *сжатием без потерь*. Оно используется для упаковки текстов, программ, данных, которые ни в коем случае нельзя искажать.

Сжатие без потерь — это такое уменьшение объёма закодированных данных, при котором можно восстановить их исходный вид из кода без искажений.

За счёт чего удалось сжать файл? Только за счёт того, что в файле была некоторая закономерность, избыточность — использовались только 4 символа вместо полного набора.

Алгоритм RLE

При сжатии данных, в которых есть цепочки одинаковых кодов, можно применять ещё один простой алгоритм, который называется **кодированием цепочек одинаковых** символов (англ. **RLE** — *Run Length Encoding*). Представим себе файл, в котором записаны сначала 100 русских букв «А», а потом — 100 букв «Б»:

1 ... 100	101 ... 200
АА……АА	ББ……ББ

При использовании алгоритма RLE сначала записывается количество повторений первого символа, затем — сам первый символ, затем — количество повторений второго символа, затем — второй символ и т. д. В данном случае весь закодированный файл занимает 4 байта:

01100100_2	11000000_2	01100100_2	11000001_2
100	А (код 192)	100	Б (код 193)

Таким образом, мы сжали файл в 50 раз за счёт того, что в нём снова была *избыточность* — цепочки одинаковых символов. Это

сжатие без потерь, потому что, зная алгоритм упаковки, исходные данные можно точно восстановить из кода.

Очевидно, что такой подход будет приводить к *увеличению* (в 2 раза) объема данных в том случае, когда в файле нет соседних одинаковых символов. Чтобы улучшить результаты RLE-кодирования даже в этом наихудшем случае, алгоритм модифицировали следующим образом. Упакованная последовательность содержит управляющие байты, за каждым управляющим байтом следует один или несколько байтов данных. Если старший бит управляющего байта равен 1, то следующий за управляющим байт данных при распаковке нужно повторить столько раз, сколько записано в оставшихся 7 битах управляющего байта. Если же старший бит управляющего байта равен 0, то надо взять несколько следующих байтов данных без изменения. Сколько именно — записано в оставшихся 7 битах управляющего байта. Например, управляющий байт 10000111_2 говорит о том, что следующий за ним байт надо повторить 7 раз, а управляющий байт 00000100_2 — о том, что следующие за ним 4 байта надо взять без изменений. Например, последовательность

10001111_2	11000000_2	00000010_2	11000001_2	11000010_2
повтор 15	А (код 192)	2	Б (код 193)	В (код 194)

распаковывается в 17 символов: АААААААААААААААББ.

Алгоритм RLE успешно использовался для сжатия рисунков, в которых большие области закрашены одним цветом, и некоторых звуковых данных. Сейчас вместо него применяют более совершенные, но более сложные методы. Один из них (алгоритм Хаффмана) мы рассмотрим далее. Алгоритм RLE используется, например, на одном из этапов кодирования рисунков в формате JPEG. Возможность использования RLE-сжатия есть также в формате BMP (для рисунков с палитрой 16 или 256 цветов).

Префиксные коды

Вспомните азбуку Морзе, в которой для уменьшения длины сообщения используется неравномерный код — часто встречающиеся буквы (А, Е, М, Н, Т) кодируются короткими последовательностями, а редко встречающиеся — более длинными. Такой код можно представить в виде структуры, которая называется **деревом** (вспомните дерево каталогов).

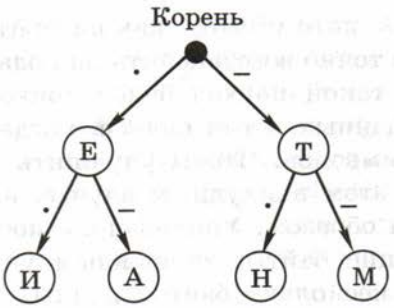


Рис. 1.2

На рисунке 1.2 показано неполное дерево кода Морзе, построенное только для символов, коды которых состоят из одного и двух знаков (точек и тире). Дерево состоит из узлов (большая чёрная точка и кружки с символами алфавита) и соединяющих их направленных рёбер, стрелки указывают направление движения. Верхний узел (в который не входит ни одна стрелка) называется **корнем дерева**. Из корня и из всех промежуточных узлов (кроме конечных узлов — **листьев**) выходят две стрелки, левая помечена точкой, а правая — знаком «тире». Чтобы найти код символа, нужно пройти по стрелкам от корня дерева к нужному узлу, выписывая метки стрелок, по которым мы переходим. В дереве нет циклов (замкнутых путей), поэтому кодовое слово для каждого символа определяется единственным образом. По этому дереву можно построить такие кодовые слова:

Е • И •• А •– Т – Н –• М ––

Это неравномерный код, в нём символы имеют коды разной длины. При этом всегда возникает проблема разделения последовательности на отдельные кодовые слова. В коде Морзе она решена с помощью символа-разделителя — паузы. Однако можно не вводить дополнительный символ, если выполняется **условие Фано**: ни одно из кодовых слов не является началом другого кодового слова. Это позволяет однозначно раскодировать сообщение в реальном времени, по мере получения очередных символов.

Префиксный код — это код, в котором ни одно кодовое слово не является началом другого кодового слова (условие Фано).

Для использования этой идеи в компьютерной обработке данных нужно было разработать алгоритм построения префиксного

кода. Впервые эту задачу решили, независимо друг от друга, американские математики и инженеры Клод Шеннон и Роберт Фано (**код Шеннона–Фано**). Они использовали *избыточность* сообщений, состоящую в том, что символы в тексте имеют разные частоты встречаемости. В этом случае для построения кода нужно читать данные исходного файла два раза: на первом проходе определяется частота встречаемости каждого символа, затем строится код с учётом этих данных, и на втором проходе символы текста заменяются на их коды.

Пусть, например, текст состоит только из букв «О», «Е», «Н», «Т» и пробела. Известно, сколько раз они встретились в тексте: пробел — 179, О — 89, Е — 72, Н — 53 и Т — 50 раз. Делим символы на 2 группы так, чтобы общее количество найденных в тексте символов первой группы было примерно равно общему количеству символов второй группы. В нашем случае лучший вариант — это объединить пробел и букву Т в первую группу (сумма 179 + 50 = 229), а остальные символы — во вторую (сумма 89 + 72 + 53 = 214).

Символы первой группы будут иметь коды, начинающиеся с 0, а остальные — с 1. В первой группе всего два символа, у одного из них, например у пробела, вторая цифра кода будет 0 (и полный код 00), а у второго — 1 (код буквы Т — 01) .

Во второй группе три символа, поэтому продолжаем деление на две группы, примерно равные по количеству символов в тексте. В первую выделяем одну букву, которая чаще всего встречается — это буква О (её код будет 10), а во вторую — буквы Е и Н (они получают коды 110 и 111). Код Шеннона–Фано, построенный для этого случая, можно нарисовать в виде дерева (рис. 1.3).

Символ ␣ на этой схеме обозначает пробел.

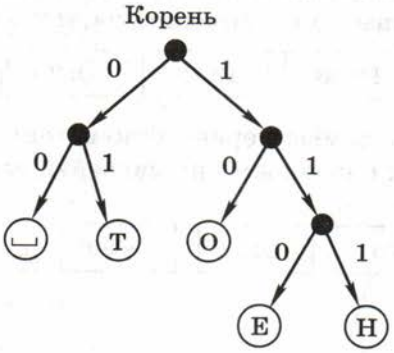


Рис. 1.3

Легко проверить, что для этого кода выполняется условие Фано. Это можно сразу определить по построенному дереву. В нём все символы располагаются в листьях, а не в промежуточных узлах. Это значит, что «по пути» от корня дерева до любого символа никаких других символов в промежуточных узлах не встречается (сравните с деревом кода Морзе).

Для раскодирования очередного символа последовательности мы просто спускаемся от корня дерева, выбирая левую ветку, если очередной бит — 0, и правую, если этот бит равен 1. Дойдя до листа дерева, мы определяем символ, а затем снова начинаем с корня дерева, чтобы раскодировать следующий символ, и т. д. Например, пусть получена последовательность

01100110001101111001.

Результат ее раскодирования — «ТОТО ЕНОТ».

Алгоритм Хаффмана

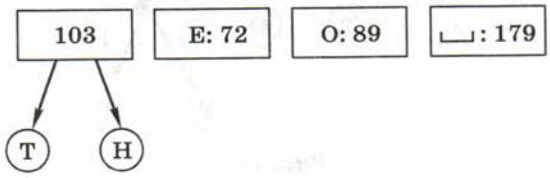
Дэвид Хаффман (1925–1999)

Было доказано, что в некоторых случаях кодирование Шеннона–Фано дает неоптимальное решение, и можно построить код, который ещё больше уменьшит длину кодовой последовательности. Например, в предыдущем примере (код Шеннона–Фано) пробел и буква «Т» имеют коды одинаковой длины, хотя буква «Т» встречается в 3,5 раза реже пробела. Через несколько лет Дэвид Хаффман, ученик Фано, разработал новый алгоритм кодирования и доказал его оптимальность.

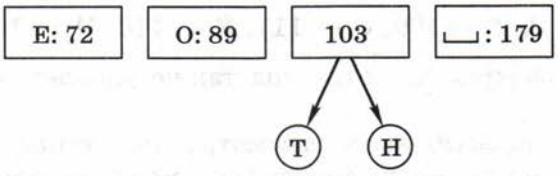
Построим код Хаффмана для того же самого примера. Сначала отсортируем буквы по увеличению частоты встречаемости:

Т: 50	Н: 53	Е: 72	О: 89	␣: 179

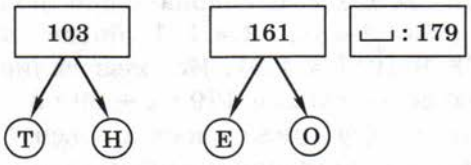
Затем берём две самые первые буквы, они становятся листьями дерева, а в узел, с которым они связаны, записываем сумму их частот:



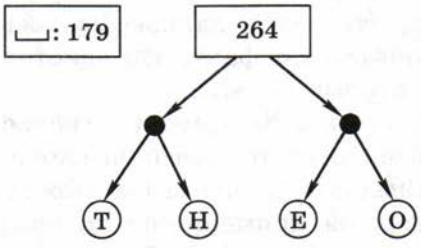
Таким образом, буквы, которые встречаются реже всего, получили самый длинный код. Снова сортируем буквы по возрастанию частоты, но для букв «Т» и «Н» используется их суммарная частота:



Повторяем ту же процедуру для букв «Е» и «О», частоты которых оказались минимальными, и сортируем по возрастанию частоты:



Теперь объединяем уже не отдельные буквы, а пары, и снова сортируем:



На последнем шаге остаётся объединить символ «пробел» с деревом, которое построено для остальных символов. У каждой стрелки, идущей влево от какого-то узла, ставим код 0, а у каждой стрелки, идущей вправо — код 1 (рис. 1.4).

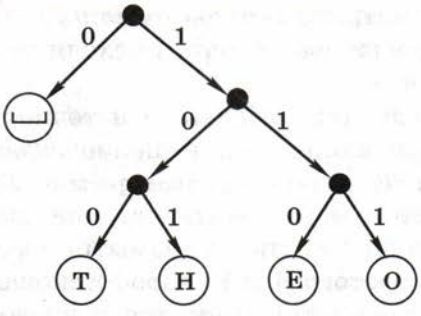


Рис. 1.4

По этому дереву, спускаясь от корня к листьям-символам, получаем коды Хаффмана, обеспечивающие оптимальное сжатие с учётом частоты встречаемости символов:

␣ — 0, Т — 100, О — 111, Е — 110, Н — 101.

Легко проверить, что этот код также удовлетворяет условию Фано.

Сравним эффективность рассмотренных выше методов. Для алфавита из 5 символов при равномерном кодировании нужно использовать 3 бита на каждый символ, так что общее число битов в сообщении равно $(179 + 89 + 72 + 53 + 50) \cdot 3 = 1329$ битов. При кодировании методом Шеннона–Фано получаем $179 \cdot 2 + 89 \cdot 2 + 72 \cdot 3 + 53 \cdot 3 + 50 \cdot 2 = 1011$ битов, коэффициент сжатия составляет $1329/1011 \approx 1{,}31$. Использование кода Хаффмана даёт последовательность длиной $179 \cdot 1 + 89 \cdot 3 + 72 \cdot 3 + 53 \cdot 3 + 50 \cdot 3 = 971$ бит, коэффициент сжатия равен 1,37. В сравнении со случаем, когда для передачи используется однобайтный код (8 битов на символ), выигрыш получается ещё более весомым: кодирование Хаффмана сжимает данные примерно в 3,65 раза. Обратим внимание, что сжать данные удалось за счёт *избыточности*: мы использовали тот факт, что некоторые символы встречаются чаще, а некоторые — реже.

Алгоритм кодирования Хаффмана и сейчас широко применяется благодаря своей простоте, высокой скорости кодирования и отсутствию патентных ограничений (он может быть использован свободно). Например, он применяется на некоторых этапах при сжатии рисунков (в формате JPEG) и звуковой информации (в формате MP3).

Сжатие с потерями

Выше мы рассмотрели методы *сжатия без потерь*, при которых можно в точности восстановить исходную информацию, зная алгоритм упаковки.

В некоторых случаях небольшие неточности в передаче данных несущественно влияют на решение задачи. Например, небольшие помехи в телефонном разговоре или в радиопередаче, которая транслируется через Интернет, не мешают пониманию речи. Некоторое дополнительное размытие уже и без того размытого изображения (фотографии) непрофессионал даже не заметит. Ещё в большей степени это относится к видеофильмам. Поэтому

иногда можно немного пожертвовать качеством изображения или звука ради того, чтобы значительно сократить объём данных. В этих случаях используется *сжатие с потерями*.

Сжатие с потерями — это такое уменьшение объёма закодированных данных, при котором распакованный файл может отличаться от оригинала.

Самые известные примеры сжатия с потерями — это алгоритмы JPEG (для изображений), MP3 (для упаковки звука) и все алгоритмы упаковки видеофильмов (MJPEG, MPEG4, DivX, XviD).

Простейший метод сжатия рисунков — снижение глубины цвета. На рисунке 1.5 показаны три изображения с различной глубиной цвета. Третье из них занимает при кодировании в 4 раза меньше места, чем первое, хотя воспринимается уже с трудом.

8 битов на пиксель (256 цветов) | 4 бита на пиксель (16 цветов) | 2 бита на пиксель (4 цвета)

Рис. 1.5

Алгоритм JPEG (англ. *Joint Photographic Experts Group* — объединенная группа экспертов по фотографии) — один из наиболее эффективных методов сжатия с потерями для изображений. Он довольно сложен, поэтому мы опишем только основные идеи этого алгоритма.

Из классической RGB-модели (красный, зелёный, синий) цвет переводится в **модель YCbCr**, где Y — яркость, Cb — «синева», Cr — «краснота»:

$$Y = 0{,}299 \cdot R + 0{,}587 \cdot G + 0{,}114 \cdot B,$$
$$Cb = 128 - 0{,}1687 \cdot R - 0{,}3313 \cdot G + 0{,}5 \cdot B,$$
$$Cr = 128 + 0{,}5 \cdot R - 0{,}4187 \cdot G - 0{,}0813 \cdot B,$$

В этих формулах R, G и B обозначают красную, зелёную и синюю составляющие цвета в RGB-модели. Чёрно-белое изображение (точнее, оттенки серого), содержит только Y-составляющую, компоненты цвета Cb и Cr равны 128 и не несут полезной информации.

Основная идея сжатия основана на том, что человеческий глаз более чувствителен к изменению яркости, т. е. составляющей Y. Поэтому на синюю и красную составляющие, Cb и Cr, приходится меньше полезной информации, и на этом можно сэкономить. Рассмотрим квадрат размером 2 × 2 пикселя. Можно, например, сделать так: запомнить Y-компоненту для всех пикселей изображения (4 числа) и по одной компоненте Cb и Cr на все 4 пикселя (рис. 1.6).

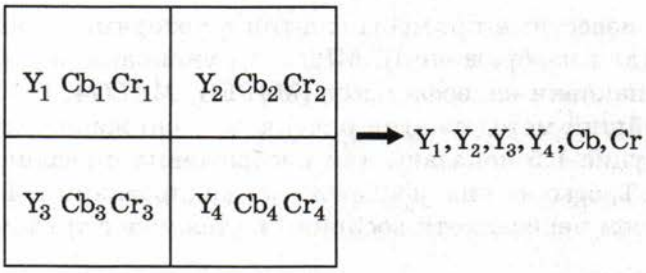


Рис. 1.6

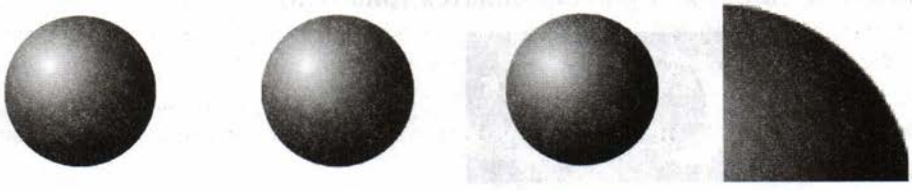
Для определения сохраняемых значений Cb и Cr можно использовать, например, усреднение — принять

$$Cb = (Cb_1 + Cb_2 + Cb_3 + Cb_4)/4,\quad Cr = (Cr_1 + Cr_2 + Cr_3 + Cr_4)/4.$$

Таким образом, вместо 12 чисел мы должны хранить всего 6, данные сжаты в 2 раза. Однако мы не сможем восстановить обратно исходный рисунок, потому что информация и том, какие были значения Cb и Cr для каждого пикселя, безвозвратно утеряна, есть только некоторые средние значения. При выводе такого изображения на экран можно, например, считать, что все 4 пикселя имели одинаковые значения Cb и Cr. Более сложные алгоритмы используют информацию о соседних блоках, сглаживая резкие переходы при изменении этих составляющих. Поэтому при кодировании с помощью алгоритма JPEG изображение несколько размывается.

Но это еще не всё. После этого к оставшимся коэффициентам применяется сложное *дискретное косинусное преобразование*, с помощью которого удаётся выделить информацию, которая мало влияет на восприятие рисунка человеком, и отбросить её. На последнем этапе для сжатия оставшихся данных используются алгоритмы RLE и Хаффмана.

На рисунке 1.7 вы видите результат сжатия изображения шарика в формате JPEG с разным качеством.



Качество 100 (8400 байтов) | Качество 50 (3165 байтов) | Качество 0 (1757 байтов) | Фрагмент (качество 0)

Рис. 1.7

Рисунки, сохранённые с качеством 100 и 50, практически не отличаются внешне, однако второй из них занимает в 2,65 раза меньше места. При снижении качества до минимального явно заметны искажения (*артефакты*), вызванные потерей информации при сжатии. Кажется, что рисунок построен из квадратов размером 8 × 8 пикселей (в действительности именно такие квадраты использует алгоритм в качестве базовых ячеек). Кроме того, искажения особенно заметны там, где происходит резкий переход от одного цвета к другому.

Тот же самый рисунок был сохранён в формате **BMP** без сжатия и с **RLE-сжатием**, а также в форматах **GIF** (сжатие без потерь с помощью **алгоритма LZW**) и **PNG** (сжатие без потерь с помощью **алгоритма DEFLATE**). Размеры полученных файлов (в килобайтах) сравниваются на диаграмме (рис. 1.8).

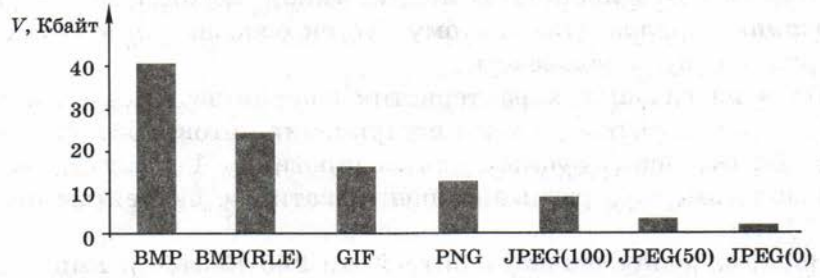


Рис. 1.8

По диаграмме видно, что наибольшее сжатие (наименьший объём файла) обеспечивается алгоритмом JPEG. Заметим, что особенность этого рисунка — плавные переходы между цветами.

Теперь рассмотрим другой рисунок, в котором у объектов есть большие области одного цвета и чёткие границы у объектов. Здесь ситуация несколько меняется (рис. 1.9).

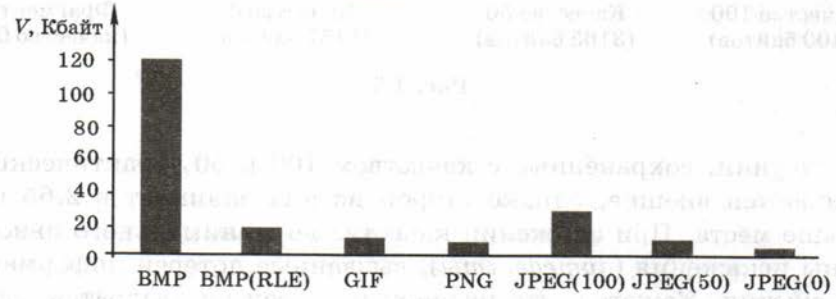


Рис. 1.9

Форматы GIF и PNG обеспечивают степень сжатия, сравнимую с алгоритмом JPEG среднего качества. Учитывая, что JPEG приводит к искажению изображения, а GIF и PNG — нет, для таких рисунков не рекомендуется использовать формат JPEG.

Другой известный пример сжатия с потерями — **алгоритм MPEG-1 Layer 3**, который более известен как **MP3**. В нём отбрасываются некоторые компоненты звука, которые практически неразличимы для большинства людей. Такой подход называется *кодированием восприятия*, потому что он основан на особенностях восприятия звука человеком.

Одна из главных характеристик сжатия звукового файла — **битрейт** (англ. *bitrate* — темп поступления битов). Битрейт — это число битов, используемых для кодирования 1 с звука. Формат MP3 поддерживает разные степени сжатия, с битрейтом от 8 до 320 Кбит/с.

Звук, закодированный с битрейтом 256 Кбит/с и выше, даже профессионал вряд ли отличит от звучания компакт-диска. Вспоминая материал учебника 10 класса, можно подсчитать, что 1 секунда звука занимает на компакт-диске (частота дискретизации 44 кГц, глубина кодирования 16 битов, стерео)

$2 \cdot 88\,000 = 176\,000$ байтов $= 1\,408\,000$ битов $= 1375$ Кбит.

Таким образом, формат MP3 обеспечивает степень сжатия 1375/256 ≈ 5,4 при сохранении качества звучания для человека (хотя часть информации, строго говоря, теряется).

Для сжатия видео может использоваться **алгоритм MJPEG** (англ. *Motion JPEG* — JPEG в движении), когда каждый кадр сжимается по алгоритму JPEG. Кроме того, широко применяют стандарт **MPEG-4**, разработанный специально для сжатия цифрового звука и видео.

Программы (и устройства), которые выполняют кодирование и декодирование звука и видео, называют **кодеками** (англ. *codec — coder/decoder* — кодировщик/декодировщик). Наиболее известные аудиокодеки (программы для преобразования звука) – *MP3*, *Ogg Vorbis* (*OGG*) и *AAC*. Среди видеокодеков чаще всего используются кодеки стандарта MPEG-4 (*DivX*, *Xvid*, *H.264*, *QuickTime*) и кодек *WMV* (Windows Media Video). Самые популярные свободные (англ. *open source*) и бесплатные (англ. *freeware*) кодеки собраны в пакет *K-Lite Codec Pack* (codecguide.com).

Выводы

Мы рассмотрели различные алгоритмы сжатия информации. Все они основаны на том, что в информации есть некоторая *избыточность*, закономерность, которую можно использовать для уменьшения объёма данных. **Хорошо сжимается** информация, в которой большая избыточность:

- тексты, в которых повторяются одинаковые слова и символы имеют разные частоты встречаемости (файлы с расширениями **txt**, **doc**, **docx**);
- документы — тексты с оформлением и, возможно, вставленными рисунками, таблицами и т. п.; электронные таблицы (файлы с расширениями **doc**, **docx**, **xls**);
- рисунки, имеющие большие области одного цвета и записанные без сжатия (файлы **bmp**);
- несжатый звук (файлы **wav**);
- несжатое видео (файлы **avi**)[1].

Плохо сжимаются данные, где избыточность маленькая или ее совсем нет:

[1] Файлы с расширением `avi` могут хранить как сжатое, так и несжатое видео.

- архивы, упакованные со сжатием (**zip**, **rar**, **7z** и др.);
- сжатые рисунки (файлы **gif**, **jpg**, **png**, **tif** и др.);
- сжатый звук (файлы **mp3**, **wma**);
- сжатое видео (файлы **mpg**, **wmv**, **mp4**);
- программы (файлы **exe**).

Данные невозможно сжать, если в них нет никаких закономерностей. Поэтому хуже всего сжимаются случайные числа, например полученные на компьютере. Современным программам-упаковщикам иногда удаётся их немного сжать, но не более, чем на 1–2%. Это происходит потому, что в последовательности псевдослучайных чисел, которые выдает компьютерная программа-генератор, всё же можно выявить какие-то закономерности.

Заметим, что не всегда нужно стремиться к полному устранению избыточности кода. Как вы знаете из предыдущего параграфа, именно избыточность позволяет обнаруживать и исправлять ошибки при передаче данных.

Вопросы и задания

1. За счёт чего удаётся сжать данные без потерь? Когда это сделать принципиально невозможно?
2. Какие типы файлов сжимаются хорошо, а какие — плохо? Почему?
3. Текстовый файл, записанный в однобайтной кодировке, содержит только 33 заглавные русские буквы, цифры и пробел. Ответьте на следующие вопросы:
   - какое минимальное число битов нужно выделить на символ при передаче, если каждый символ кодируется одинаковым числом битов?
   - сколько при этом будет занимать заголовок пакета данных?
   - при какой минимальной длине текста коэффициент сжатия будет больше 1?
4. На чём основан алгоритм сжатия RLE? Когда он работает хорошо? Когда нет смысла его использовать?
5. Что такое префиксный код?
6. В каких случаях допустимо сжатие с потерями?
7. Опишите простейшие методы сжатия рисунков с потерями. Приведите примеры.
8. На чём основан алгоритм JPEG? Почему это алгоритм сжатия с потерями?
9. Что такое артефакты?
10. Для каких типов изображений эффективно сжатие JPEG? Когда его не стоит применять?

11. На чём основано сжатие звука в алгоритме MP3?
12. Что такое битрейт? Как он связан с качеством звука?
13. Какое качество звука принимается за эталон качества на непрофессиональном уровне?
14. Какие методы используются для сжатия видео?

Подготовьте сообщение

а) «Программы для сжатия данных»
б) «Алгоритмы сжатия изображений»
в) «Аудиокодеки»
г) «Видеокодеки»

Задачи

1. С помощью алгоритма RLE закодируйте сообщение «ВАААААВАААРРРРРРРРРР».
2. После кодирования методом RLE получилась следующая последовательность байтов (первый байт — управляющий):
   10000011 10101010 00000010 10101111 11111111 10000101 10101010
   Сколько байтов будет содержать данная последовательность после распаковки?
3. После кодирования методом RLE получилась следующая последовательность байтов (первый байт — управляющий):
   00000011 10101010 00000010 10101111 10001111 11111111
   Сколько байтов будет содержать данная последовательность после распаковки?
4. Раскодируйте сообщение, которое закодировано с помощью приведённого в тексте кода Шеннона–Фано:
   11111000011011111001001101111001.
5. Постройте дерево, соответствующее коду А — 0, Б — 1, В — 00, Г — 01, Д — 10, Е — 11. Является ли этот код префиксным? Как это определить, посмотрев на дерево?

*6. Постройте дерево Хаффмана для фразы «МАМА МЫЛА ЛАМУ». Найдите коды всех входящих в нее символов и закодируйте сообщение. Чему равен коэффициент сжатия в сравнении с равномерным кодом минимальной длины? С однобайтной кодировкой?

§ 4
Информация и управление

Кибернетика

Норберт Винер (1894–1964)

Человек с древних времён хотел использовать предметы и силы природы в своих целях, т. е. **управлять** ими. В середине XX века учёные поняли, что управление неразрывно связано с информацией и информационными процессами. В 1948 г. появилась книга американского математика Норберта Винера «Кибернетика, или Управление и связь в животном и машине», с которой началось развитие новой науки — кибернетики.

Кибернетика (от греч. κυβερνητης — кормчий, рулевой) — это наука, изучающая общие закономерности процессов управления и передачи информации в машинах, живых организмах и обществе.

Основная идея Винера состояла в том, что управление в любых системах — технических, биологических, общественных — определяется одними и теми же законами и тесно связано с обменом информацией. Понятие «система» вам знакомо на «бытовом» уровне, теперь мы определим его более строго.

Что такое система?

Представьте себе, что человек, у которого болит голова, приходит к врачу. Врач может реагировать на его жалобу двумя способами:

- просто дать обезболивающее;
- попытаться выяснить причину головной боли и лечить именно заболевший орган.

В первом случае врач применил простейшую модель: если болит голова, нужно дать таблетку, чтобы она не болела. Во втором варианте врач понимает, что головная боль вызвана какой-то другой болезнью в организме, и лечить нужно именно её. Он рассмат-

ривает человека как единое целое, как *систему*, т. е. использует *системный подход*.

Система (от греч. συστημα — составленный) — это группа объектов и связей между ними, выделенных из среды и рассматриваемых как одно целое (рис. 1.10).

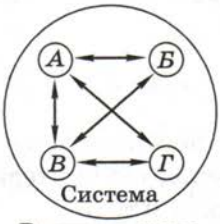


Рис. 1.10

Системный подход состоит в том, что объект исследования рассматривается как система с учётом всех взаимосвязей между её частями.

Системой можно назвать организм животного и человека, автомобиль, компьютер, семью, общество, и вообще любой достаточно сложный объект, который можно разбить на части. В курсе информатики вы познакомились с понятиями «операционная система», «файловая система», «система программирования».

Различают **естественные** (природные) **системы** и **искусственные системы** (созданные человеком). Любая искусственная система создана с определённой *целью*, например русский язык — для общения, дом — для жилья, поезд — для перевозки пассажиров и грузов.

Объекты, из которых строится система, называют её **компонентами**. За счёт связей между частями система обладает особыми свойствами, которых нет ни у одного отдельного компонента. **Системный эффект** состоит в том, что свойства системы нельзя свести к «сумме» свойств её компонентов. Например, скопление клеток — это не живой организм; процессор, память и устройства ввода и вывода, не связанные между собой, — это не компьютер. Все компоненты самолёта тяжелее воздуха, т. е. каждый из них стремится упасть на землю, однако составленная из них система (самолёт) летает.

Одни и те же элементы, связанные по-разному, могут представлять собой совершенно разные системы. Алмаз (самое твёрдое вещество) и графит (из которого делают грифели карандашей) состоят из одних и тех же атомов углерода, но эти вещества имеют разные кристаллические решётки, поэтому обладают разными свойствами.

Если в компоненте системы можно выделить отдельные части и связи между ними, он называется **подсистемой** (системой более низкого уровня). Как правило, для каждой системы существует **надсистема** — система более высокого уровня. Например, процессор — это подсистема в составе компьютера (он состоит из нескольких взаимосвязанных узлов), а сам компьютер — подсистема в составе компьютерной сети (надсистемы).

Цель работы системы задаётся надсистемой.

Например, автомобиль служит для того, чтобы перевозить людей и грузы, цех выполняет задание директора завода, вы учитесь для того, чтобы быть готовыми к жизни после школы и т. д.

Простейшие компоненты системы, которые в данной задаче (!) нет смысла «разбирать» на части, называются её **элементами**. Например, винтик в составе машины можно считать элементом системы. Однако, с точки зрения химии, металл состоит из атомов, связанных в кристаллическую решетку, поэтому при изучении строения вещества тот же «винтик» можно считать системой. На рисунке 1.11 система S состоит из двух подсистем S_1 и S_2, а также элементов Г и Д.

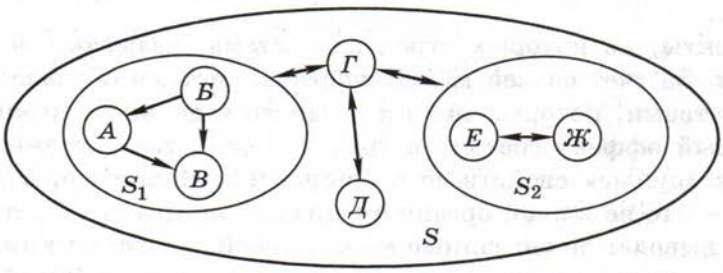


Рис. 1.11

В результате развития системного подхода в середине XX века появилось новое научное направление — **системный анализ**. Задача системного анализа — изучение сложных систем (технических, биологических, экономических, социальных) на основе теории управления и теории информации.

Системы управления

Во многих системах взаимодействие между подсистемами можно рассматривать как управление. Такие системы (их называют **системами управления**) всегда содержат **управляющий объект** (в теории управления его называют *регулятором*) и **управляемый объект** (просто «объект»). Цель управления чаще всего задаётся извне, т. е. регулятор является только исполнителем. Например, водитель служебной машины никогда не решает сам, куда везти своего начальника.

Регулятор действует на объект (управляет им) так, чтобы цель была достигнута (рис. 1.12).

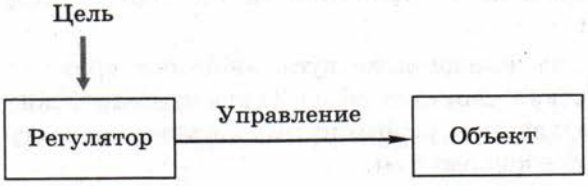


Рис. 1.12

Управление (управляющее воздействие) — это сигнал, который поступает от регулятора к объекту. Это значит, что при управлении передается информация.

На приведённой схеме (см. рис. 1.12) регулятор не получает никакой информации о состоянии объекта, т. е. действует «вслепую». Такие простейшие системы управления называют **разомкнутыми**, в них информация идет только в одну сторону, от регулятора к объекту. Приведём некоторые примеры разомкнутых систем:

- светофор (выдаёт вам световой сигнал, но не получает информации от вас);
- табло на вокзале или в аэропорту;
- процессор выставляет данные для записи в ОЗУ на шину, ждёт некоторое время и затем, без всякого подтверждения от ОЗУ, продолжает работу (считается, что ОЗУ работает надёжно и за это время данные всегда записываются);
- начальник отдаёт приказания, но не проверяет их выполнение.

Разомкнутые системы используются в двух случаях. Во-первых, когда регулятору всё равно, реагирует ли объект на управление (светофор, табло). Во-вторых, когда регулятор настолько

хорошо знает объект (имеет точную модель объекта), что уверен в выполнении своих команд. Например, начальник, уверенный в исполнительности своих подчинённых, может позволить себе (иногда) не контролировать их.

Достоинство разомкнутых систем — в их простоте. Например, во многих случаях частоту вращения электродвигателя регулируют простым реостатом, при этом не нужно ставить сложный и дорогостоящий датчик частоты вращения.

В ответственных случаях разомкнутые системы лучше не применять, потому что они имеют серьезные **недостатки:**

- для достижения цели регулятор должен иметь точную модель объекта; на практике такая модель чаще всего неизвестна;
- на объект всегда действует внешняя среда, и из-за этого воздействия свойства объекта могут непредсказуемо меняться; регулятор в разомкнутой системе не получает об этом никакой информации.

Даже очень исполнительные работники могут в какой-то момент «схалтурить» из-за личных переживаний или неурядиц. Поэтому в разомкнутых системах чаще всего нельзя гарантированно достичь цели управления, т. е. решить поставленную задачу.

Для достижения цели регулятор должен получать информацию от объекта. Этот канал передачи информации называют **обратной связью**, потому что по нему информация передаётся (с помощью датчиков) в обратном направлении, от объекта к регулятору (рис. 1.13).

Системы с обратной связью называют *замкнутыми* системами управления, потому что информация передаётся по замкнутому контуру, циклически.

Рис. 1.13

Задача регулятора — сравнить поставленную цель и реальное состояние объекта, а потом выдать нужный управляющий сигнал. Если цель достигнута, как правило, сигнал управления больше не изменяется.

Например, регулировщик, управляющий движением на перекрёстке, использует обратную связь. Он оценивает (с помощью глаз-датчиков) количество машин, движущихся в разных направлениях, и «открывает» то или другое направление. Человек управляет лучше, чем светофор, потому что учитывает реальную обстановку, которая может изменяться непредсказуемо.

Чаще всего реальные системы управления (в природе, технике, обществе) — это замкнутые системы. Они обладают несомненными **достоинствами**:

- могут решать задачу даже тогда, когда модель объекта неточна или свойства объекта изменяются во времени;
- позволяют учитывать случайные воздействия внешней среды.

За это приходится расплачиваться усложнением системы — нужны датчики, которые передают информацию от объекта.

Представьте себе, что в тёмной комнате упала на пол пуговица, и вы не слышали, как она стукнулась. Сказать точно, где лежит пуговица, не сходя с места, невозможно, потому что вы этого не знаете (модель неточна). Вы можете попробовать искать её на ощупь (используя обратную связь — осязание). Если и это не получается, можно включить свет (зрительная обратная связь даёт больше информации), и тогда пуговица точно будет найдена (если она, конечно, там есть). В этом примере датчиками служат наши органы чувств.

Системы с обратной связью широко используются в технике. Это, например, автопилоты на самолётах и судах, регуляторы частоты вращения турбин, роботы, оборудованные датчиками (в том числе устройствами «компьютерного зрения»).

Обратная связь существует и в обществе. Жалобы граждан должны помогать работе органов управления, сигнализируя о том, что не всё в порядке. Не зря все официальные учреждения имеют контактные телефоны и сайты в Интернете, с которых можно отправить сообщение с просьбой или предложениями по их работе.

Обратная связь, при которой регулятор стремится уменьшить разницу между заданной целью и фактическим состоянием объекта, называется **отрицательной**.

Если нужно быстро перевести объект из одного состояния в другое, иногда применяют **положительную** обратную связь, при которой регулятор стремится увеличить разницу между заданным значением и сигналом обратной связи. Часто в этом случае система переходит в колебательный режим, поэтому положительная обратная связь используется в генераторах колебаний. Другой пример положительной обратной связи — цепные реакции в химии и физике (горение, взрыв, ядерные реакции).

Системы управления делятся на автоматические и автоматизированные. **Автоматические системы** работают без участия человека, например автопилот. В **автоматизированных системах** сбор и предварительную обработку информации выполняет компьютер, а решение по поводу управления принимает человек.

Многие системы умеют «подстраиваться» под изменения внешних условий или изменение свойств объекта управления. Они называются **адаптивными**. Классическая адаптивная система — глаз человека, который изменяет диаметр зрачка в зависимости от освещённости. В технических системах адаптивные регуляторы могут управлять объектом, модель которого очень плохо известна или меняется. Они параллельно решают две задачи — управляют и уточняют имеющуюся модель, что, в свою очередь, позволяет улучшить управление.

Вопросы и задания

1. Объясните, что такое управление.
2. Как связана информация с управлением?
3. Какая наука изучает общие закономерности процессов управления и передачи информации? Кто считается её основоположником?
4. Что такое система? Чем она отличается от группы объектов? Приведите примеры.
5. Что такое подсистема?
6. Всякий ли объект можно считать системой? Почему?
7. Как вы думаете, можно ли назвать русский язык системой? Обоснуйте ваш ответ.
8. Объясните, почему операционная система, файловая система и система программирования – это именно системы?
9. Что такое системный подход?
10. Чем различаются естественные и искусственные системы?
11. Чем различаются понятия «компонент системы», «элемент системы» и «подсистема»? Приведите примеры.
12. Что такое надсистема? Почему цель для системы всегда задает надсистема? Приведите примеры.

13. Что такое система управления? Какие элементы в ней всегда есть?
14. Что такое обратная связь? Зачем она нужна?
15. От чего зависят свойства системы?
16. Приведите примеры разомкнутых систем. В чём их достоинства и недостатки?
17. Почему разомкнутые системы не используют в ответственных случаях?
18. Почему разомкнутые системы всё же используются, несмотря на их недостатки?
19. Перечислите достоинства и недостатки замкнутых систем.
20. Приведите примеры управления с обратной связью из разных областей, из вашей жизни.
21. Что такое отрицательная обратная связь?
22. Когда используют положительные обратные связи?
23. Чем различаются автоматические и автоматизированные системы?
24. Что такое адаптивные системы? Приведите примеры.

Подготовьте сообщение

а) «Вклад Н. Винера в науку»

б) «Системы управления в природе»

в) «Системы управления в обществе»

г) «Отрицательная и положительная обратная связь»

д) «Что такое адаптивная система?»

§ 5
Информационное общество

Что такое информационное общество?

По мере развития человечества постоянно повышалась роль информации в жизни общества и отдельного человека. Самые важные достижения в этой области — это:

- *письменность* (около 3000 лет до н. э., Египет);
- *книгопечатание* (X век — Китай, XV век — Европа);
- *средства связи* (телеграф, телефон, радио, телевидение; конец XIX — начало XX века);
- *компьютеры* (вторая половина XX века).

Сейчас считается, что мы переходим от индустриального общества, в котором главная роль отводится промышленности, к *информационному* (постиндустриальному).

Информационное общество — это такая ступень развития цивилизации, на которой главными продуктами производства становятся информация и знания.

Переход к информационному обществу часто называют **информатизацией**.

Япония, США и некоторые страны Европы (например, Германия) уже приблизились к информационному обществу. Об этом можно судить по следующим признакам:

- внедрение компьютеров и информационных технологий во все сферы жизни;
- развитие *коммуникаций* (средств связи) — компьютерных сетей, сотовой связи и т. п.;
- необходимость компьютерной грамотности для любого человека;
- свобода доступа к информации;
- доступность образования, в том числе дистанционного (через Интернет);
- изменение структуры экономики — все больше людей занимаются не производством товаров, а получением и обработкой информации;
- изменение уклада жизни людей (например, общение через Интернет вместо личной встречи; интернет-магазины и использование электронных денег и т. п.).

Если в результате индустриализации машины заменили человека на производстве, то теперь компьютеры начинают самостоятельно (без участия человека) собирать и обрабатывать информацию, заменяя людей в умственной работе, которую до этого мог сделать только человек.

С одной стороны, переход к информационному обществу облегчает жизнь людей, потому что на всех рутинных и тяжёлых работах их заменяют компьютеры и роботы. С другой стороны, существует немало **негативных** последствий:

- усиление влияния средств массовой информации (СМИ), с помощью которых несколько человек могут влиять на большие массы людей; так широкое освещение террора в СМИ приводит к значительному влиянию террористических организаций на нашу жизнь, поэтому некоторые учёные говорят о том, что общество теряет устойчивость;

- в результате доступности информации разрушается частная жизнь людей и целых организаций (например, в Интернет нередко просачиваются сведения, не предназначенные для всеобщего доступа); чтобы этого не происходило, при обмене личной информацией (например, при телефонных разговорах) используется шифрование данных;
- в гигантском потоке информации очень сложно выбрать качественные и достоверные данные (в Интернете, например, очень много ложной и противоречивой информации);
- личное общение всё больше заменяется общением в Интернете, реальная жизнь становится виртуальной;
- многим людям старшего поколения очень сложно приспособиться к меняющимся условиям, они могут оказаться «за бортом».

Информационные ресурсы

Ресурсами называют любые средства, позволяющие после некоторой «обработки» получить желаемый результат. Традиционно среди ресурсов общества выделяли природные, материальные, энергетические, трудовые, финансовые. Сейчас очень важную роль играют **информационные ресурсы** — документы, в том числе в библиотеках, архивах, банках данных, информационных системах.

Информационные ресурсы стали **товаром**, который сто́ит не меньше (а иногда и больше), чем другие ресурсы. Множество фирм оказывает **информационные услуги**:

- поиск и подбор информации;
- подбор персонала (кадровые агентства);
- обучение (учебные центры);
- рекламные агентства;
- консалтинг (консультации, услуги по оптимизации бизнеса);
- разработка программ и веб-сайтов.

Информационные технологии

Один из признаков информационного общества — широкое внедрение информационных технологий во все сферы жизни.

Любая *технология* — это способ сделать продукт из исходных материалов (с гарантированным результатом).

Новые информационные технологии — это технологии, связанные с использованием компьютеров для хранения, защиты, обработки и передачи информации.

К ним относятся:

- *подготовка документов*;
- *поиск информации*;
- *телекоммуникации* (компьютерные сети, Интернет, электронная почта);
- *автоматизированные системы управления* (АСУ);
- *системы автоматизированного проектирования* (САПР);
- геоинформационные системы (на основе карт, снимков со спутника);
- *обучение* (электронные учебники, компьютерные тренажёры, дистанционное обучение через Интернет).

Остановимся на некоторых информационных технологиях более подробно.

Примером **АСУ** может служить система, предназначенная для автоматизации работы ресторанов. На рабочих местах барменов, официантов, менеджеров и администраторов установлены персональные компьютеры, связанные в единую локальную сеть. При получении заказа информация о нём сразу передаётся в базу данных и распечатывается на принтерах, установленных на кухне.

В промышленности широко используются **автоматизированные системы управления технологическими процессами (АСУ ТП)**. Представим себе организацию, которая отвечает за работу нескольких дизель-генераторов, расположенных в разных местах. Каждый генератор связан с управляющим компьютером, который собирает информацию о работе установки и передаёт ее (по кабельной линии или через радиомодем) в центральный офис, где за обстановкой следит диспетчер. В случае необходимости диспетчер передаёт обратно команду на изменение режима работы генератора или принимает решение о выезде ремонтной бригады.

Современный инженер для подготовки чертежей уже почти никогда не использует традиционные инструменты: кульман, линейку, лекала, перья. Все делается на компьютере в **системах автоматизированного проектирования (САПР**[1]). САПР применяются в машиностроении, строительстве, электронике. Все САПР ис-

[1] Английское название **CAD** — *Computer-Aided Design*.

пользуют векторное кодирование изображений. Наиболее известны системы:

- *AutoCAD* — самая популярная система автоматизированного проектирования и черчения;
- *ArchiCAD* — для проектирования зданий, ландшафтов и мебели;
- *OrCAD* — для проектирования электронных схем;
- *КОМПАС* — отечественная САПР, позволяющая оформлять чертежи в соответствии с российскими стандартами (рис. 1.14).

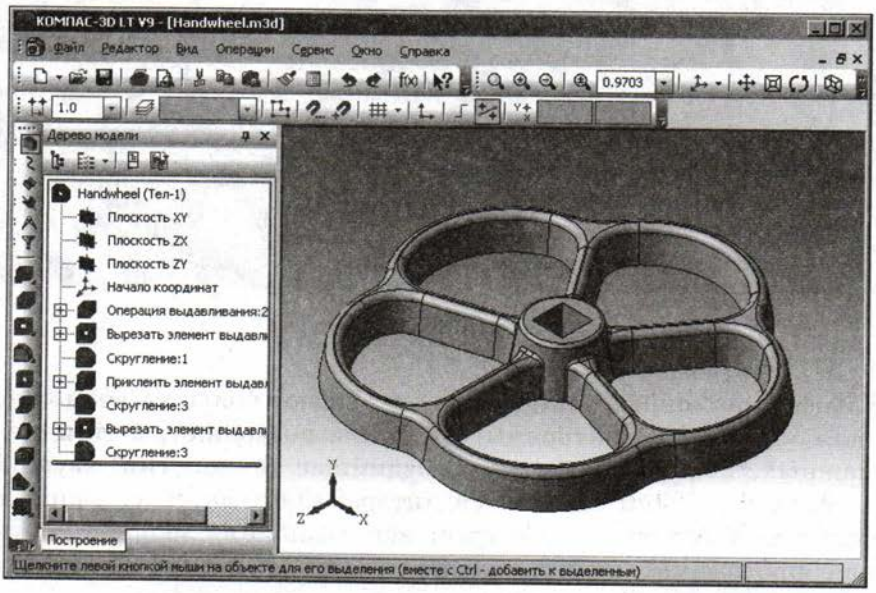


Рис. 1.14

Геоинформационные системы (географические информационные системы, **ГИС**) — это инструменты для работы с цифровыми картами. Они могут строиться на основе карт в векторном формате или спутниковых снимков поверхности Земли. На основной слой «накладываются» растровые и векторные слои с дополнительной информацией: дороги, города и поселки, улицы, другие объекты (рис. 1.15). С помощью ГИС можно измерить расстояние между точками, проложить маршрут по существующим дорогам, и даже посмотреть, как выглядят улицы некоторых крупных городов, — это позволяют сервисы *Яндекс.Панорамы* и *Google Street View*.

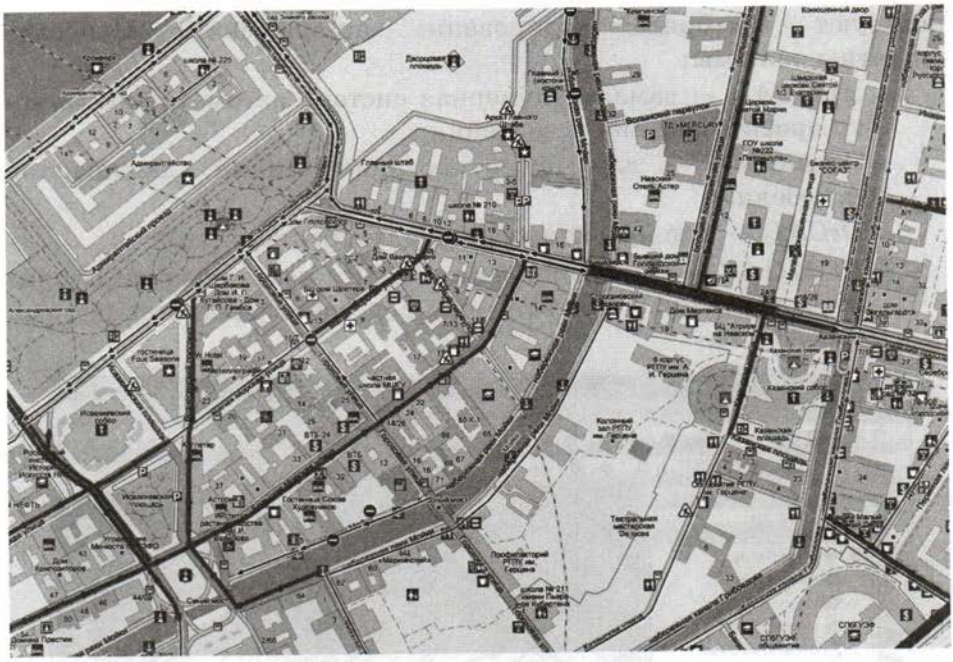


Рис. 1.15 (maps.yandex.ru)

Информационные технологии открывают большие возможности для **обучения**. Электронные учебники позволяют, в отличие от бумажных, использовать **мультимедийные технологии**: звук, видео, анимацию. Они становятся интерактивными, т. е. учащийся может использовать их как тренажёр, выполняя задания. Обучающая программа-учебник строит модель конкретного ученика и выбирает «маршрут», оптимальный именно для него.

С развитием Интернета стало возможно **дистанционное обучение**. На сервере выкладываются учебные материалы и практические задания. Группе учащихся выделяется преподаватель (**тьютор**, от англ. *tutor* — преподаватель, наставник), у которого они могут свободно консультироваться, используя электронную почту, чаты и др. Все проблемные вопросы можно обсудить с однокурсниками на форуме. С помощью такой системы можно осваивать курс университета, не выходя из дома (нужен только доступ в Интернет).

Для отработки навыков работы со сложной техникой широко используют **компьютерные тренажёры**. Они необходимы для обучения персонала в тех случаях, когда ошибка при работе с ре-

Рис. 1.16

Рис. 1.17

альным объектом может привести к тяжёлым последствиям. Очень важны тренажёры для обучения лётчиков (рис. 1.16) и судоводителей (рис. 1.17), они позволяют отработать действия в разнообразных аварийных ситуациях. Часто тренажёры применяются в связке с реальной аппаратурой[1].

Простейшие тренажёры — это компьютерные игры-симуляторы, моделирующие полёт самолета или вождение автомашины. Они строятся на основе математических моделей, которые приближённо описывают реальные объекты.

Информационная культура

Возрастание роли информации в современном обществе требует от каждого человека (и от общества в целом) определённой культуры обращения с информацией и информационными технологиями, т. е. **информационной культуры.**

Информационная культура общества — это способность общества:

- эффективно использовать информационные ресурсы и средства обмена информацией;
- применять передовые достижения в области информационных технологий.

Информационная культура человека — это его умение использовать современные технологии для решения своих задач,

[1] Фотографии тренажёров предоставлены компаниями «Динамика» (www.dinamika-avia.ru) и «Транзас» (www.transas.ru).

связанных с поиском и обработкой информации. Современный человек должен уметь:

- формулировать свою потребность в информации;
- находить нужную информацию, используя различные источники;
- отбирать и анализировать информацию (в том числе конспектировать, составлять рефераты);
- представлять информацию в разных видах;
- обрабатывать информацию и создавать новую информацию;
- использовать информацию для принятия решений.

Это не просто какие-то требования к «идеальному» человеку. Дело в том, что в новом информационном обществе успешность человека и его возможность реализовать все свои способности будут зависеть от его умения грамотно работать с информацией.

Кроме того, понятие «информационная культура» включает в себя этичное поведение при использовании информации. Неэтично подавлять высказывания других, даже если это противоречит вашим убеждениям. Неэтично беспокоить других и угрожать им. При возникновении конфликтов нужно применить все возможные меры, чтобы решить проблему без обращения в суд.

Неэтично распространять высказывания, изображения или мнения других без их согласия. Если вы хотите разместить в общем доступе (например, в социальной сети) информацию, касающуюся частной жизни других людей (фотографии, личную переписку), следует спросить их разрешения.

Многие материалы, размещённые в Интернете, охраняются законами об авторских правах. Поэтому их нельзя взять и использовать просто так, как своё. Если вы включаете какую-то информацию (текст, картинку) в учебную работу (например, реферат), необходимо обязательно привести ссылку на сайт, где вы её взяли. В некоторых случаях, например при желании использовать чужой рисунок или текст на своём сайте или в печатном издании, нужно получить письменное согласие автора.

Использование чужого материала без ссылки (т. е. фактически присвоение чужих результатов) называется **плагиатом**. Если будет доказано, что этим автору причинен крупный материальный ущерб (более 50 тыс. рублей), начнёт действовать Уголовный кодекс (УК РФ, статья 146), предусматривающий денежный штраф, исправительные работы или даже арест на срок до 6 месяцев.

Необходимо понимать, что неправомерный доступ к чужой информации («взлом» сайтов, почтовых ящиков, личных страни-

чек) — это уголовное преступление, которое наказывается лишением свободы на срок до 5 лет (УК РФ, статья 272). То же самое относится и к созданию, использованию и распространению вредоносных компьютерных программ (УК РФ, статья 273, до 7 лет лишения свободы).

С одной стороны, развитие компьютерной техники и коммуникационных технологий предоставляет широкие возможности и (кажущуюся) полную свободу. С другой стороны, в информационном обществе продолжают действовать все нормы права и морали, которые выработало человечество за свою историю.

Вопросы и задания

1. Перечислите важнейшие достижения человечества в области информационных технологий.
2. Назовите негативные последствия информатизации общества. Как, на ваш взгляд, с ними можно бороться?
3. На каких условиях можно использовать информацию, найденную в сети Интернет?
4. Что такое этичное поведение в Интернете?
5. Что такое информационное общество? Чем оно отличается от индустриального?
6. Перечислите характерные черты информационного общества.
7. Что такое информационные ресурсы?
8. Перечислите информационные услуги.
9. Что такое технология? Что такое новые информационные технологии?
10. Что такое АСУи АСУ ТП? Приведите примеры.
11. Что такое САПР? В каких областях они используются?
12. Что такое геоинформационные системы?
13. Какие новые возможности для обучения появляются в информационном обществе?
14. Что такое дистанционное обучение? Обсудите его достоинства и недостатки.
15. Что такое информационная культура общества?
16. Что такое плагиат?
17. Какие действия, связанные с информационными технологиями, являются уголовными преступлениями?
18. Из чего складывается информационная культура человека?
19. Верно ли, что информационная культура и компьютерная грамотность — это одно и то же? Обоснуйте свой ответ.

Подготовьте сообщение

а) «Информатизация общества: плюсы и минусы»
б) «Этика в Интернете»
в) «Интернет и закон»

Практические работы к главе 1

Работа № 1 «Набор и оформление документа»
Работа № 2 «Алгоритм RLE»
Работа № 3 «Сравнение алгоритмов сжатия»
Работа № 4 «Использование архиваторов»
Работа № 5 «Сжатие с потерями»

ЭОР к главе 1 на сайте ФЦИОР (http://fcior.edu.ru)

- Классификация информационных процессов
- Алгоритм оптимального кодирования Хаффмана
- Роль и место информационных технологий в современном обществе

Самое важное в главе 1

- Количество информации зависит от того, насколько ожидаемо полученное сообщение.
- Для того чтобы обнаруживать и исправлять ошибки в сообщении, вызванные помехами при передаче, нужно вводить дополнительные контрольные биты в каждый блок данных.
- Для каждого сообщения существует предельно достижимая степень сжатия. Чем меньше закономерностей в коде сообщения, тем сложнее его сжать.
- Существуют два типа алгоритмов сжатия: сжатие без потерь и сжатие с потерями. Сжатие с потерями используется при передаче рисунков, звука, видео.
- Для эффективного управления необходима обратная связь от управляемого объекта к регулятору.
- Главная черта информационного общества — повышение роли информационных технологий в жизни людей.

Глава 2
Моделирование

§ 6
Модели и моделирование

Введение

При слове «модель» у многих, наверное, появляется мысль о моделях самолётов, кораблей, танков и другой техники, которые стоят на полках магазинов. Однако слово «модель» имеет более широкое значение. Например, игрушки, в которые играют дети всех возрастов, — это модели реальных объектов, с которыми они встречаются в жизни (или встретятся в будущем).

Говоря о модели, мы всегда указываем на какой-то другой объект (процесс, явление), например: «Глобус — это модель Земли». Здесь «другой объект» — это Земля, он называется **оригиналом**. Объект становится моделью только тогда, когда есть оригинал, *модели без оригинала не существует*. Оригиналами могут быть:

- *объекты* (самолёт, здание, ядро атома, кристаллическая решётка металла, галактика);
- *процессы* (изменение климата и экологической обстановки, развитие экономики);
- *явления* (землетрясения, цунами, солнечные затмения).

Зачем нужны модели вообще? Они появляются тогда, когда мы хотим решить какую-то *задачу*, связанную с оригиналом, а изучать оригинал невозможно, потому что:

- оригинал *не существует*; например, учебники истории — это модели общества, которого уже нет; возможные последствия ядерной войны учёные изучали на моделях, потому что ставить реальный эксперимент было бы безумием;
- исследование оригинала *дорого или опасно* для жизни, например, при управлении ядерным реактором, испытании скафандра для космонавтов, создании нового самолёта или корабля;

- *сложно исследовать* непосредственно оригинал, например Солнечную систему, молекулы и атомы, очень быстрые процессы в двигателях внутреннего сгорания, очень медленные движения материков;
- нас интересуют только *некоторые свойства* оригинала; например, чтобы испытать новую краску для самолёта, не нужно строить самолёт.

Итак, модель всегда связана не только с оригиналом, но и с конкретной задачей, которую мы хотим решить с её помощью.

Для любого оригинала можно построить множество разных моделей. Например, моделью человека может служить его фотография, паспорт, генетический код, манекен, рентгеновский снимок, биография. Зачем столько? Дело в том, что каждая из этих моделей отражает только те свойства, которые важны при решении конкретной задачи. Такие свойства в теории моделирования называют **существенными**.

Вместе с тем одна и та же модель может описывать множество самых разных оригиналов. Например, в различных задачах атом, муха, человек, автомобиль, высотное здание, даже планета Земля могут быть представлены как материальные точки (если размеры соседних объектов и расстояния между ними значительно больше).

Теперь можно дать определение модели и моделирования.

Модель — это объект, который обладает существенными свойствами другого объекта, процесса или явления (оригинала) и используется вместо него.

Моделирование — это создание и исследование моделей с целью изучения оригиналов.

Практически всё, что мы делаем с помощью компьютеров, — это моделирование. Например, база данных библиотеки — это модель реального хранилища книг, компьютерный чертёж — это модель детали и т. д.

С помощью моделирования можно решать **задачи** четырёх типов:

- *исследование* оригинала, изучение его строения (чаще всего в научных и учебных целях);

- *анализ* («что будет, если...») — прогнозирование влияния различных воздействий на оригинал;
- *синтез* («как сделать, чтобы...») — управление оригиналом;
- *оптимизация* («как сделать лучше всего...») — выбор наилучшего решения в данных условиях.

Виды моделей

Существует множество классификаций моделей, каждая из которых отражает какое-то одно свойство. Универсальной классификации моделей нет.

По природе модели делятся на **материальные** (физические, предметные) и **информационные** (рис. 2.1). Материальные модели «можно потрогать» — это игрушки, уменьшенные копии самолётов и кораблей, чучела животных, учебные модели молекул и т. п.

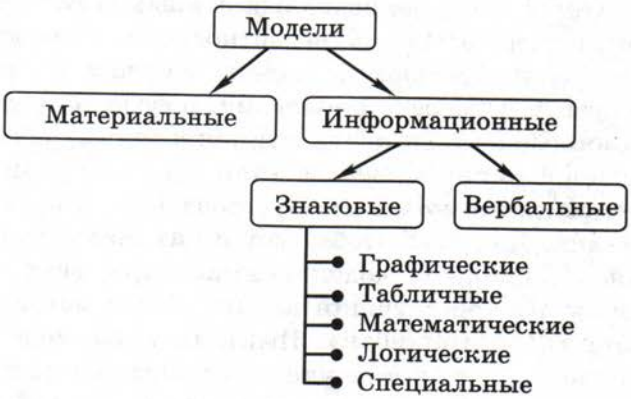


Рис. 2.1

Информационные модели — это информация о свойствах оригинала и его связях с внешним миром. Среди них выделяют **вербальные модели** (словесные, от *лат.* verbalis — словесный) и **знаковые модели**, записанные с помощью какого-то формального языка:

- **графические** (схемы, карты, фотографии, чертежи);
- **табличные**;
- **математические** (формулы);
- **логические** (варианты выбора на основе анализа условий);
- **специальные** (ноты, химические формулы и т. п.).

По фактору времени выделяют статические и динамические модели. **Статические модели** (от греч. στατος — неподвижный) описывают оригинал в состоянии покоя, в данный момент времени (схема сил, действующих на неподвижное тело; фотография; результаты осмотра врача, модель молекулы). **Динамические модели** (от греч. δυναμις — сила) описывают движение, развитие, изменение (модель полёта шарика, модель землетрясения, история болезни, видеозапись события, модель развития химической реакции).

По характеру связей модели делятся на детерминированные (от лат. *determinare* — определять) и вероятностные. В **детерминированных моделях** связи между исходными данными и результатами жёстко заданы, при одинаковых исходных данных всегда получается тот же самый результат (например, расчёт по известным формулам, модель движения тела без учета ветра и т. п.). **Вероятностные модели** учитывают случайность событий в реальном мире, поэтому при одних и тех же исходных данных результаты моделирования могут отличаться. К вероятностным относятся модели броуновского движения частиц, полёта самолета с учётом ветра, движения корабля на морском волнении, поведения человека.

Имитационные модели используются в тех случаях, когда поведение сложной системы нельзя (или крайне трудно) предсказать теоретически, но можно смоделировать её реакцию на внешние воздействия. Для того чтобы найти оптимальное решение задачи, нужно выполнить моделирование при всех возможных вариантах и выбрать наилучший из них. Такой метод часто называют методом «проб и ошибок». Имитационные модели позволяют очень точно описать поведение оригинала, но полученные результаты справедливы только для тех случаев, которые мы моделировали (что случится в других условиях — непонятно). Примеры использования имитационных моделей:

- испытание лекарств на мышах, обезьянах, группах добровольцев;
- модели биологических систем;
- экономические модели управления производством;
- модели систем массового обслуживания (банки, магазины и т. п.).

Для понимания работы процессора можно использовать его имитационную модель, которая позволяет вводить команды в определённом формате и показывает изменение значений регистров (ячеек памяти) процессора. Подобные модели применяют

в том случае, когда нужно написать программу для системы, на которой её невозможно отлаживать (например, для микропроцессора, встроенного в утюг). Такой подход называют «кросс-программирование»: программа пишется и отлаживается в одной системе, а работать будет в другой. В этом случае «другую» систему приходится моделировать с помощью имитационной модели.

Игровые модели позволяют учитывать действия противника, например, при моделировании военных действий, соревнований, конкуренции в бизнесе. Задача игрового моделирования — найти лучшую стратегию в игре — план действий, который даёт наилучшие результаты даже в том случае, когда противник играет безошибочно. Этими вопросами занимается *теория игр* — раздел математики, одним из создателей которого был Джон фон Нейман. В сложных случаях используются имитационные игровые модели.

Адекватность

При моделировании всегда возникает вопрос: можно ли верить полученным результатам? Иначе говоря, будет ли оригинал вести себя так же, как и модель?

Адекватность модели (от лат. *adaequatus* — равный) — это совпадение свойств модели и оригинала в рассматриваемой задаче.

Адекватность означает, что результаты моделирования:

- не противоречат выводам теории, например законам сохранения (вещества, энергии и т. п.);
- подтверждаются экспериментом с реальным объектом (оригиналом).

Таким образом, адекватность модели можно окончательно доказать только экспериментом: если мы сможем решить задачу, используя результаты моделирования, то модель адекватна. На практике модель считается адекватной, если расхождения между численными результатами моделирования и эксперимента не превышают 10%.

Нужно понимать, что любая модель отличается от оригинала, поэтому она может быть адекватна только при определённых условиях — в той задаче, для решения которой она создавалась. Например, модель деления амёб (через некоторый интервал вре-

мени каждая амёба делится надвое) адекватна только при малом количестве амёб и небольших интервалах наблюдения, иначе амебы заполнили бы все пространство.

Во многих случаях результаты моделирования — это некоторые числа, измеренные или рассчитанные по результатам эксперимента с моделью. Это могут быть, например, сила, расстояние, скорость, ускорение, давление и др. Чаще всего эти величины для модели и оригинала будут различаться, поэтому нужно уметь пересчитывать «модельные» данные в соответствующие значения для оригинала. Этими вопросами занимается *теория подобия*. Простейший пример — работа с картой. Расстояние, измеренное по карте, нужно умножить на масштабный множитель, тогда получится соответствующее расстояние на реальной местности.

Вопросы и задания

1. Что такое модель? Зачем нужны модели?
2. Что вы думаете по поводу другого определения модели: «Модель — это упрощённое представление реального объекта, процесса или явления»?
3. Приведите примеры моделей объектов, процессов и явлений.
4. Приведите примеры разных моделей Земли. В каких задачах они используются?
5. Приведите примеры разных моделей человека. Для каких задач они предназначены?
6. Приведите примеры, когда одна модель используется для представления разных объектов-оригиналов.
7. Приведите примеры моделей, с которыми мы работаем на компьютерах.
8. Что такое моделирование?
9. Назовите типичные задачи, которые могут решаться с помощью моделирования.
10. Что такое анализ и синтез? Какой из этих типов задач более сложен?
11. Приведите примеры задач анализа и синтеза.
12. Что такое оптимизация?
13. Как вы думаете, почему нет единой классификации моделей?
14. К какому типу (типам) можно отнести следующие модели:
    а) «Каляка — это маляка с тремя грымзиками»;
    б) $a^2 + b^2 = c^2$;
    в) «Если горит красный свет, то стой. Если горит зёленый свет — иди»;
    г) $2H_2 + O_2 = 2H_2O$?
    Используйте разные классификации.

15. Объясните, чем различаются статические и динамические модели.
16. Что такое вероятностные модели? Зачем они могут понадобиться?
17. Как называются модели, в которых не используются случайные события?
18. Назовите достоинства и недостатки вероятностных и детерминированных моделей.
19. Какую модель — вероятностную или детерминированную — вы рекомендуете выбрать для исследования движения судна в шторм? Почему?
20. Что такое имитационные модели? Подумайте, какие достоинства и недостатки у них есть по сравнению с теоретическими моделями.
21. Что такое метод проб и ошибок?
22. Приведите примеры задач из вашей практики, для которых имитационная модель позволяет быстрее получить результат, чем теоретическая.
23. Какие модели называют игровыми?
24. Верно ли, что модели, используемые при создании компьютерных игр, — это игровые модели? Обоснуйте вашу точку зрения.
25. Приведите примеры детерминированных и вероятностных игровых моделей.
26. Может ли существовать вербальная динамическая имитационная игровая модель? Обоснуйте свою точку зрения.
27. Что такое адекватность модели? Как можно убедиться, что модель адекватна?
28. Почему ни одна модель не может быть полностью адекватна оригиналу?

Подготовьте собщение

а) «Анализ и синтез»

б) «Детерминированные и вероятностные модели»

в) «Игровые модели»

г) «Адекватность моделей»

Задачи

1. Площадь леса на карте масштаба 1:200 000 равна 5 см^2. Сколько квадратных километров составляет площадь реального леса?
2. Напишите программу, которая моделирует работу процессора. Процессор имеет 4 регистра, они обозначаются R0, R1, R2 и R3. Все команды состоят из трёх десятичных цифр: код операции, номер первого регистра и номер второго регистра или число от 0 до 9. Коды команд и примеры их использования приведены в таблице.

Код операции	Описание	Пример	Псевдокод
1	запись константы	128	`R2:=8`
2	копирование значения	203	`R3:=R0`
3	сложение	331	`R1:=R1+R3`
4	вычитание	431	`R1:=R1-R3`

Обратите внимание, что результат записывается во второй регистр. Команды вводятся последовательно как символьные строки. После ввода каждой строки программа показывает значения всех регистров.

3. Добавьте в систему команд в задаче 2 умножение, деление и логические операции с регистрами — И, ИЛИ, исключающее ИЛИ.

*4. Добавьте в систему команд в задаче 2 логическую операцию НЕ. Подумайте, как можно использовать второй регистр.

*5. Сделайте так, чтобы в команде с кодом 1 (задача 2) можно было использовать шестнадцатеричные значения констант (0–9, A–F).

6. Добавьте в задаче 2 обработку ошибок типа «неверная команда», «неверный номер регистра», «деление на ноль».

*7. Добавьте в задаче 2 команду «СТОП», которая прекращает работу программы. Введите строковый массив, моделирующий память, и запишите в него программу — последовательность команд. Ваша программа должна последовательно выполнять эти команды, выбирая их из «памяти», пока не встретится команда «СТОП».

*8. Подумайте (задача 2), как можно было бы организовать условный переход: перейти на N байтов вперёд (или назад), если результат последней операции — ноль.

§ 7
Системный подход в моделировании

Как вы знаете из § 4, системный подход состоит в том, что объект исследования (моделирования) рассматривается как система с учётом всех взаимосвязей между ее частями.

Модели могут обладать свойством системности, а могут не обладать. В таблице 2.1 приведены примеры моделей-«несистем» и моделей-систем для одних и тех же объектов.

Таблица 2.1

Оригинал	Модель-«несистема»	Модель-система
Пос. Орехово	Фотографии	Карта, видеофильм
Метро	Список станций	Схема метро
Рыбы в озере	Независимые модели развития щук и карасей	Модель, учитывающая, что щуки едят карасей
Автомобиль	Чертежи отдельных деталей	Сборочный чертёж
Солнечная система	Независимое движение планет	Движение планет под действием сил всемирного тяготения

Поскольку модель-система состоит из отдельных компонентов и связей между ними, можно говорить о структуре модели, т. е. о том, как именно связаны её компоненты. Далее мы рассмотрим наиболее важные структуры моделей-систем.

Табличные модели

Табличные модели используются тогда, когда нужно в наглядной форме представить **информацию об объектах**, имеющих одинаковый набор свойств (таблица типа **«объект–свойства»**) (табл. 2.2).

Таблица 2.2

Фамилия	Имя	Год рождения	Место отдыха
Иванов	Кузьма	1955	о. Валаам
Кузьмин	Сидор	1978	о. Ольхон
Сидоров	Иван	1990	о. Кипр

В виде таблиц оформляются расписания (уроков, поездов, самолётов), статистические данные (например, сколько произведено чугуна и стали на душу населения в разных странах). Функция тоже может быть задана в виде таблицы. С помощью таблицы Менделеева устанавливается связь между свойствами химического элемента и зарядом атомного ядра.

Таблица может определять **отношения между объектами** (таблица типа **«объект–объект»**). Например, в табл. 2.3 показано, кто в каком городе живёт.

Таблица 2.3

	Вася	Петя	Коля	Маша	Даша	Глаша
Москва	✓				✓	
Санкт-Петербург		✓		✓		
Пермь			✓			✓

Таблицы — это основной способ хранения информации в базах данных. Кроме того, для обработки табличных данных предназначены специальные программы — табличные процессоры.

В учебнике для 10 класса было показано, как таблицы можно использовать при решении логических задач. Здесь мы рассмотрим ещё один тип задач, который требует анализа табличных данных: определение оптимального маршрута поездки.

Задача. Путешественник прибыл в посёлок Берёзовое в 8 утра по местному времени и увидел следующее расписание автобусов (табл. 2.4).

Таблица 2.4

Отправление из	Прибытие в	Время отправления	Время прибытия
Берёзовое	Лесное	07:30	10:00
Берёзовое	Осиновое	11:50	14:10
Лесное	Березовое	12:50	15:20
Полевое	Лесное	13:20	14:40
Осиновое	Полевое	14:00	17:15
Лесное	Осиновое	14:20	15:30
Осиновое	Лесное	14:40	15:50
Берёзовое	Полевое	16:00	17:50
Лесное	Полевое	16:10	17:30
Полевое	Осиновое	17:40	19:55

Определите самое раннее время, когда он может попасть в Полевое, и как ему нужно ехать.

Решение. Из расписания видно, что автобусы ходят между четырьмя населёнными пунктами. Нарисуем схему, показывающую все возможные способы переезда из посёлка Берёзовое в посёлок Полевое. Буквы в кружках обозначают посёлки (Б — Берёзовое, П — Полевое, Л — Лесное и О — Осиновое), а слева и справа от

них записано время отправления и прибытия автобусов согласно расписанию (рис. 2.2).

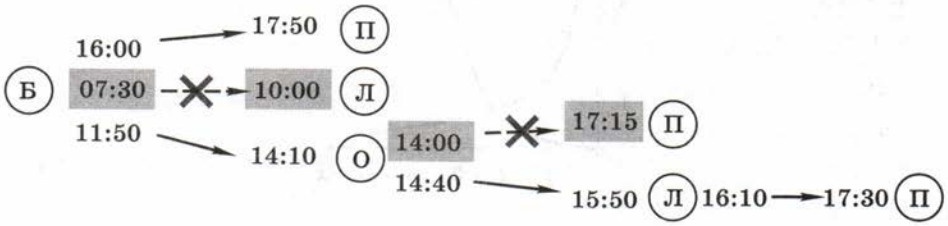


Рис. 2.2

Штриховыми линиями обозначены маршруты, на которые путешественник не успевает (поэтому дальнейшие варианты мы даже не рассматривали). Действительно, когда он приехал в Берёзовое в 8 утра, автобус в Лесное уже ушёл (в 7:30). Приехав в Осиновое в 14:10, он не успеет на автобус в Полевое, уходящий в 14:00.

Таким образом, остаются два варианта: ждать прямого автобуса в Полевое (прибытие в 17:50) или ехать с двумя пересадками через Осиновое и Лесное (прибытие в 17:30). Второй вариант позволяет доехать немного раньше.

Диаграммы

Воспринимая числовые данные, человек вынужден в уме анализировать эту информацию и делать выводы. Это требует значительных усилий, особенно если чисел много. Чтобы облегчить восприятие информации, её представляют в виде диаграмм (греч. διαγραμμα — рисунок, чертёж) — графических моделей, построенных по числовым данным, которые часто хранятся в таблицах.

Диаграммы позволяют быстро сравнить значения, увидеть изменения, сделать выводы на основании большого количества данных.

Первыми диаграммами, с которыми вы работали на уроках математики, были графики функций. По горизонтальной оси откладываются значения аргумента, а по вертикальной — значения функции (рис. 2.3).

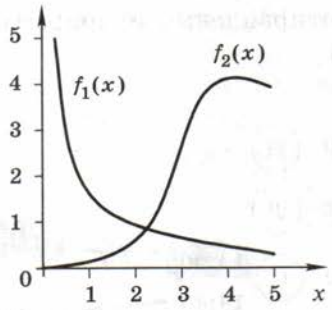


Рис. 2.3

Рассмотрим таблицу, в которой записано количество разных домашних животных у трёх жителей деревни (табл. 2.5).

Таблица 2.5

	Овцы	Кролики	Куры
Аськин	1	2	5
Баськин	4	2	5
Сенькин	2	3	4

Чтобы изобразить эти данные, можно использовать **столбчатую диаграмму (гистограмму)** (рис. 2.4).

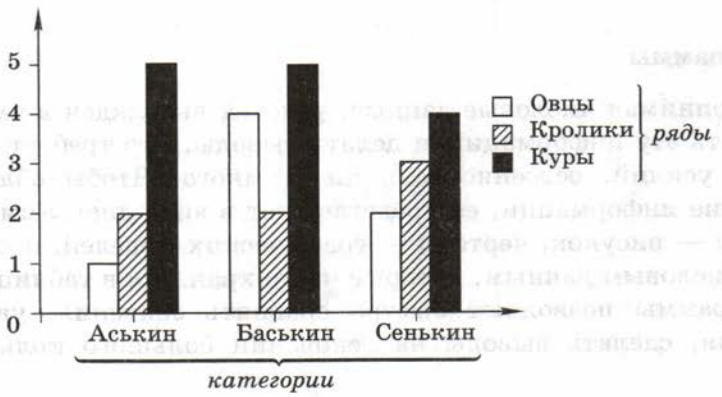


Рис. 2.4

Здесь на горизонтальной оси откладываются не числа, а заголовки строк (или столбцов) таблицы, они называются **категориями**.

Столбики одного цвета — это **ряд данных**, представляющий столбец таблицы. На этой диаграмме показаны три ряда данных — овцы, кролики и куры. Справа от диаграммы размещена **легенда** — список условных обозначений (цвет столбиков для каждого ряда).

На представленной диаграмме мы можем сразу увидеть ответы на вопросы типа «Каких животных больше всего у Аськина (Баськина, Сенькина)?».

По тем же данным можно построить ещё одну столбчатую диаграмму, у которой ряды данных размещаются не в столбцах, а в строках (рис. 2.5).

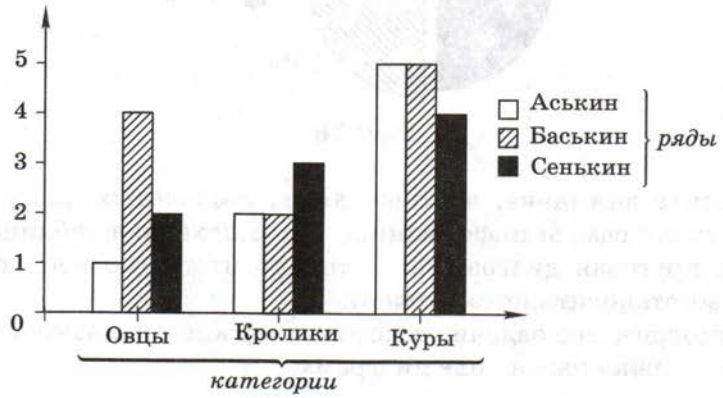


Рис. 2.5

По этой диаграмме сразу видно, у кого больше овец (кроликов, кур). Таким образом, по одним данным можно построить разные диаграммы.

Тип диаграммы выбирается так, чтобы было лучше видно то, что хочет показать автор.

Таблицу 2.5 можно немного расширить, посчитав общее количество овец, кроликов и кур, а также общее количество животных у каждого жителя (табл. 2.6).

Таблица 2.6

	Овцы	Кролики	Куры	Всего
Аськин	1	2	5	8
Баськин	4	2	5	11
Сенькин	2	3	4	9
Всего	7	7	14	28

Всего получилось 28 животных, из них 7 овец, 7 кроликов и 14 кур. Чтобы наглядно показать доли составляющих в целом, используют **круговые диаграммы** (рис. 2.6). В данном случае четверть всех животных — овцы, еще четверть — кролики, и оставшаяся половина — куры.

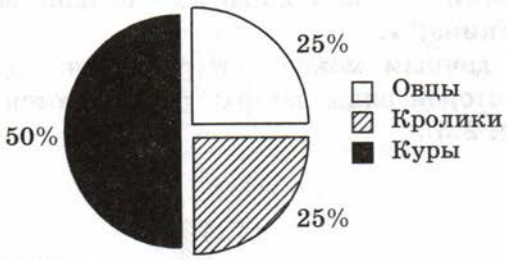


Рис. 2.6

Обратите внимание, что каждая из столбчатых диаграмм содержит ту же самую информацию, что и исходная таблица с данными, а круговая диаграмма — только итоги (по ней исходные данные восстановить невозможно).

Рассмотрим две задачи, в которых нужно анализировать данные, представленные в виде диаграмм.

Задача 1. Биологи пересчитали лосей, белок и зайцев на трёх участках заповедника и построили диаграмму (рис. 2.7).

Рис. 2.7

Какая из следующих диаграмм правильно отражает соотношение общего числа животных разных видов по всему заповеднику (рис. 2.8)?

Рис. 2.8

Решение. Сначала нужно «снять» данные со столбчатой диаграммы и записать их в таблицу (табл. 2.7).

Таблица 2.7

	Участок I	Участок II	Участок III
Лоси	15	30	15
Белки	30	20	10
Зайцы	10	15	15

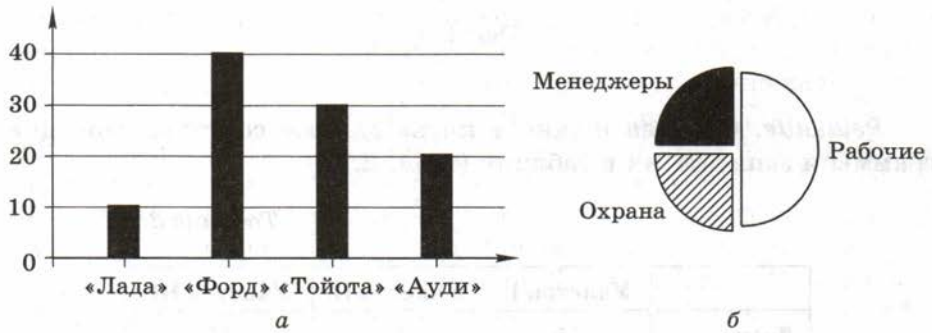
Теперь считаем, сколько было всего животных каждого вида и их общее количество (табл. 2.8).

Таблица 2.8

	Участок I	Участок II	Участок III	Всего
Лоси	15	30	15	60
Белки	30	20	10	60
Зайцы	10	15	15	40
Всего				160

Поскольку было обнаружено по 60 лосей и белок, соответствующие секторы на круговой диаграмме должны быть равны. Этому условию удовлетворяют диаграммы б) и в). Кроме того, количество зайцев составляет четверть от общего числа животных, это условие выполняется для диаграмм а) и б). Таким образом, правильный ответ — б).

Задача 2. В некоторой фирме работают менеджеры, рабочие и охрана. Они ездят на машинах четырёх марок: «Лада», «Форд», «Тойота» и «Ауди», каждый имеет ровно одну машину. На диаграмме (*а*) показано количество работников, имеющих машины определённой марки, а на диаграмме (*б*) — соотношение менеджеров, рабочих и охраны (рис. 2.9).



а *б*

Рис. 2.9

Какие из этих утверждений следуют из анализа диаграмм:

а) все «Форды» могут принадлежать менеджерам;

б) все охранники могут ездить на «Ауди»;

в) все «Тойоты» могут принадлежать рабочим;

г) все рабочие могут ездить на «Фордах»?

Решение. Сначала по данным диаграммы 1 найдём общее количество работников фирмы:

$10 + 40 + 30 + 20 = 100$ человек.

Из второй диаграммы следует, что рабочие составляют половину от общего количества, т. е. 50, а менеджеры и охранники — по четверти, т. е. по 25 человек. Теперь рассмотрим предложенные утверждения:

а) все «Форды» (40 штук) не могут принадлежать менеджерам, так как менеджеров только 25, и каждый имеет одну машину;

б) все охранники (25 человек) не могут ездить на «Ауди», потому что этих машин всего 20;

в) все «Тойоты» (30 штук) **могут** принадлежать рабочим (их 50 человек);

г) все рабочие (50 человек) не могут ездить на «Фордах» (их всего 40 штук).

Таким образом, верно только утверждение в).

Иерархические модели

Иерархические модели (деревья) описывают многоуровневую структуру (вспомните материал 10 класса). Это может быть, например, схема управления фирмой, структура организации, классификация животных, файловая система, генеалогическое дерево (родословная) и т. п. Оглавление книги — тоже иерархическая модель (разделы, главы, параграфы, пункты). С помощью дерева можно задать порядок вычисления арифметического или логического выражения. Любую систему, состоящую из подсистем, можно представить в виде иерархии.

Отношения между уровнями могут быть самые разные. Например, в схеме управления — это отношение «подчиняется» (бухгалтер подчиняется директору), в классификации — отношение «подмножество» (подмножествами отряда Хищные являются подотряды Псообразные и Кошкообразные), при описании структуры — отношение «состоит из» (компьютер состоит из процессора, памяти и внешних устройств), в генеалогическом дереве — отношения «сын» («дочь») и «родитель».

В одной из рассмотренных выше задач исходная табличная модель оказалась неудобной для решения проблемы — поиска оптимального маршрута из посёлка Березовое в посёлок Полевое. Поэтому мы построили иерархическую модель (дерево), которая показывает все возможные маршруты. После этого сразу стало ясно, какой маршрут будет наилучшим.

Сетевые модели

В сетевых моделях (**графах**) каждый узел может быть связан со всеми другими. Знакомые вам сетевые модели — это схемы дорог, компьютерных сетей, электрических цепей.

Графы позволяют очень наглядно представить информацию, однако они неудобны для автоматической обработки. Поэтому в памяти компьютера информация о графах обычно хранится в виде табличных моделей — матриц смежности и весовых матриц (вспомните материал учебника для 10 класса).

Сетевые модели широко применяются для планирования производства, есть даже специальный термин «*сетевое планирование*». Предположим, что изготовление аппарата МУХ-8-ККВ включает 8 операций, причём некоторые из них можно выполнять одновременно. Чтобы определить время изготовления, строят схему (граф, сеть), на которой узлы обозначают события (когда можно начинать очередную операцию), дуги — работы, а числа

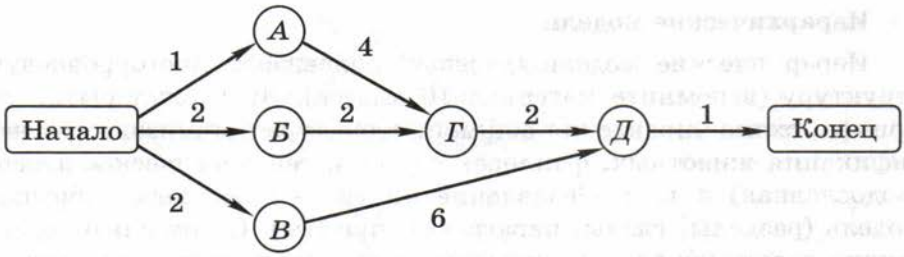


Рис. 2.10

около дуг (веса) — длительность этих работ, например, в днях (рис. 2.10).

По этой схеме видно, что в самом начале можно выполнять три работы параллельно. Чтобы начать работу *Г–Д*, нужно закончить работы *А–Г* и *Б–Г*, на это требуется 5 дней. Чтобы выполнить последнюю операцию и получить готовое изделие, нужно закончить работы *Г–Д* и *В–Д*, на это требуется 8 дней. Поэтому аппарат будет готов только через 9 дней с момента начала работ.

Для представления *знаний* применяют специальные сетевые модели, которые называются **семантическими сетями** (семантика изучает смысл сообщений). В них узлы — это объекты (понятия, процессы, явления), а дуги — связи (отношения) между ними (рис. 2.11).

Рис. 2.11

Семантические сети наглядны, с их помощью удобно анализировать фразы на естественном языке, они соответствуют современным представлениям об организации памяти человека. Однако пока такие структуры плохо приспособлены для автоматической обработки информации и поиска решений.

Сейчас делаются попытки на основе сети Интернет создать *семантическую паутину* — распределённую базу знаний. Для этого в веб-страницы нужно будет добавить специальную смысловую информацию, понятную компьютерным системам (так называемые *метаданные*).

Игровые стратегии

Как вы уже знаете из § 6, игровые модели — это модели, которые описывают соперничество двух (или более) сторон, каждая из которых стремится к выигрышу, т. е. преследует свою цель. Часто цели участников противоречивы — выигрыш одного означает проигрыш других.

Построением и изучением игровых моделей занимается **теория игр** — раздел прикладной математики. Задача состоит в том, чтобы найти **стратегию** (алгоритм игры), который позволит тому или другому участнику получить наибольший выигрыш (или, по крайней мере, наименьший проигрыш) в предположении, что соперники играют безошибочно.

Во многих простых играх, в которых игроки ходят по очереди, есть не так много вариантов развития событий, и их можно рассмотреть полностью, однозначно определив, кто выиграет в заданной начальной ситуации, если оба соперника не будут ошибаться.

Все **позиции** (игровые ситуации) делятся на выигрышные и проигрышные. **Выигрышная позиция** — это такая позиция, в которой игрок, делающий первый ход, может гарантированно выиграть при любой игре соперника, если сам не сделает ошибку. При этом говорят, что у него есть **выигрышная стратегия** — алгоритм выбора очередного хода, позволяющий ему выиграть.

Если игрок начинает играть в **проигрышной позиции**, он обязательно проиграет, если ошибку не сделает его соперник. В этом случае говорят, что у него нет выигрышной стратегии. Таким образом, общая стратегия игры состоит в том, чтобы своим ходом создать проигрышную позицию для соперника.

Выигрышные и проигрышные позиции можно охарактеризовать так:

- позиция, из которой все возможные ходы ведут в выигрышные позиции, — *проигрышная*;
- позиция, из которой хотя бы один из возможных ходов ведёт в проигрышную позицию, — *выигрышная*, при этом стратегия игрока состоит в том, чтобы перевести игру в эту проигрышную (для соперника) позицию.

Для примера рассмотрим игру с камнями, в которой участвуют два игрока. Вначале перед игроками лежит куча из некоторого количества камней (обозначим его S). За один ход игрок может добавить в кучу один камень (ход «+1») или увеличить количество камней в куче в два раза (ход «*2»). Например, имея кучу из 5 камней, за один ход можно получить кучу из 6 или 10 камней. У каждого игрока есть неограниченное количество камней. Победителем считается игрок, первым получивший кучу, в которой 14 камней или больше.

Рассмотрим возможный результат игры при разном начальном количестве S камней в куче. Очевидно, что при $S > 6$ первый игрок (т. е. игрок, делающий первый ход) выигрывает сразу, удвоив число камней в куче. Начнём заполнять таблицу, в которой для каждого значения S будем указывать, выигрышная это позиция или проигрышная, и через сколько ходов завершается игра:

S	1	2	3	4	5	6	7	8	9	10	11	12	13
							B_1	B_1	B_1	B_1	B_1	B_1	B_1

Здесь «B_1» обозначает выигрыш за один ход.

При $S = 6$ у первого игрока есть два хода: ход «+1» даёт кучу из 7 камней, а ход «*2» — кучу из 12 камней. Выиграть за один ход он не может, оба возможных хода ведут в выигрышные (для второго!) позиции, поэтому первый игрок проиграет, если второй не ошибётся. Позицию $S = 6$ отметим в таблице как «x_1» (проигрыш за 1 ход):

S	1	2	3	4	5	6	7	8	9	10	11	12	13
						x_1	B_1	B_1	B_1	B_1	B_1	B_1	B_1

Вспомним, что задача игрока — перевести игру в проигрышную для соперника позицию. Если $S = 5$ или $S = 3$, первый игрок может получить (ходом «+1» или «*2» соответственно) кучу из 6 камней, т. е. создать проигрышную позицию. Этого достаточно для выигрыша, но выиграть можно только за 2 хода:

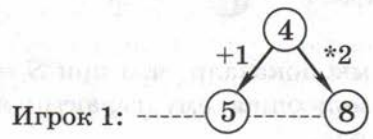
S	1	2	3	4	5	6	7	8	9	10	11	12	13
			B_2		B_2	x_1	B_1	B_1	B_1	B_1	B_1	B_1	B_1

Рассуждая аналогично, выясняем, что позиция $S = 4$ — проигрышная, так как возможные ходы ведут в выигрышные позиции (соперник выиграет за 1 или за 2 хода). При $S = 2$ первый игрок может своим ходом «*2» перевести игру в проигрышную позицию ($S = 4$), поэтому он выиграет. А при $S = 1$ он проиграет, потому что может своим ходом получить только кучу из 2 камней:

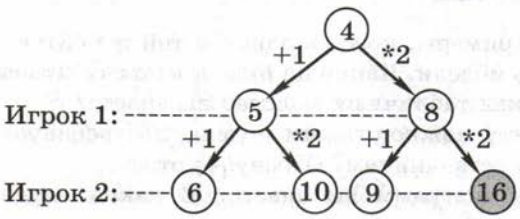
S	1	2	3	4	5	6	7	8	9	10	11	12	13
	x_3	B_3	B_2	x_2	B_2	x_1	B_1	B_1	B_1	B_1	B_1	B_1	B_1

Полученная таблица показывает результат игры первого игрока в том случае, если второй не будет ошибаться. Если игра начинается в проигрышной позиции, первый игрок проиграет, а если в выигрышной — его стратегия состоит в том, чтобы на каждом шаге своим ходом создавать проигрышную позицию для соперника.

Для полного исследования всех вариантов игры можно построить дерево, содержащее все возможные ходы. Предположим, что сначала в куче 4 камня (эта позиция будет корнем дерева). Тогда в результате первого хода может получиться куча из 5 или 8 камней:

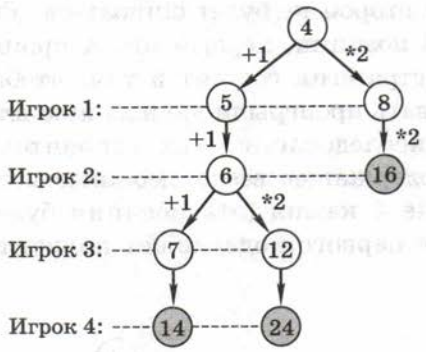


Следующий уровень дерева показывает все возможные позиции после ответного хода второго игрока:



Мы видим, что второй игрок может выиграть своим первым ходом (получив 16 камней), если первый построит кучу из 8 камней. В остальных случаях игра продолжается, и дерево можно строить дальше по тому же принципу.

Как мы уже показали ранее с помощью таблицы, при $S = 4$ выигрывает второй игрок. Чтобы доказать это с помощью дерева, не нужно строить полное дерево игры. Достаточно рассмотреть все возможные ходы соперника и для каждого из них найти один (!) выигрышный ход второго игрока. Вариант с выигрышем в один ход мы уже разобрали, теперь посмотрим, что произойдёт, если первый игрок получит кучу из 5 камней. Как следует из построенной выше таблицы, для кучи из 5 камней выигрышный ход второго игрока — «+1», он переводит игру в проигрышную позицию. При любом ответе первого игрока второй выигрывает своим вторым ходом «*2»:



Таким образом, мы доказали, что при $S = 4$ у второго игрока есть стратегия, позволяющая ему гарантированно выиграть, по крайней мере, за 2 хода.

Вопросы и задания

1. Приведите примеры, когда в одной и той же ситуации люди используют разные модели. Какие из них можно считать системами?
2. Какие два типа табличных моделей вы знаете?
3. К какому типу можно отнести модель, построенную при решении задачи с путешественником? Обоснуйте ответ.
4. Какие типы диаграмм вы знаете? В каких случаях используется каждый из них?

*5. Изучите другие типы диаграмм, которые можно построить в табличных процессорах. Зачем они используются? Приведите примеры.

6. Объясните, почему любую систему, состоящую из подсистем, можно представить в виде иерархии.

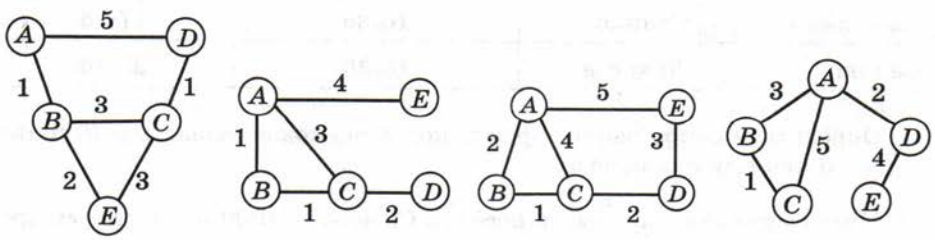
7. Вспомните, что такое матрица смежности и весовая матрица графа (см. главу 1 в учебнике для 10 класса).
8. Зачем нужны сетевые модели при планировании производства?
9. Что такое семантические сети? В чем их достоинства и недостатки?
10. Что такое семантическая паутина? Можно ли её создать на основе существующих веб-страниц? Обоснуйте свой ответ.
11. Что такое выигрышная стратегия в игре?
12. Как доказать, что заданная позиция в игре является выигрышной (или проигрышной)? Как вы думаете, в каких случаях это сделать не удаётся?
13. Почему для того, чтобы доказать выигрыш какого-то игрока в заданной начальной позиции, не нужно строить полное дерево игры?

Подготовьте сообщение

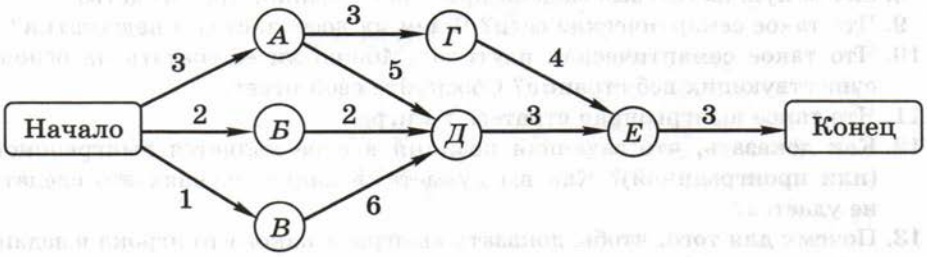
а) «Типы диаграмм»
б) «Сетевое планирование»
в) «Семантические сети»
г) «Интеллект-карты (mind maps)»
д) «Диаграммы Ганта»
е) «Использование ленты времени»

Задачи

1. В графе 9 узлов, причём каждый узел связан со всеми другими. Сколько всего связей в этой модели?
2. Система состоит из трёх подсистем по три элемента в каждой. Все элементы в каждой подсистеме связаны со всеми другими, кроме того, каждая подсистема связана со всеми другими подсистемами. Сколько всего связей в этой системе? Сравните ответы этой и предыдущей задач, сделайте выводы.
3. Постройте матрицы смежности и весовые матрицы для следующих графов.



4. Изготовление прибора «Заря-М» описывается следующей сетевой моделью (веса дуг обозначают длительность работ в днях).



Определите, через сколько дней после начала работ будет изготовлен прибор.

5. Постройте семантическую сеть на основе текста: «Кошачьи — семейство млекопитающих отряда хищных. Кроме кошек к ним относятся, например, львы и тигры. У кошачьих развиты слух и зрение. У нас дома живёт кошка Мурка. У неё рыжая шерсть».

6. Путешественник прибыл в посёлок Луковое в полночь по местному времени и увидел следующее расписание автобусов.

Отправление из	Прибытие в	Время отправления	Время прибытия
Васильево	Панино	05:10	07:20
Панино	Луковое	09:15	11:20
Луковое	Панино	10:35	12:15
Санино	Васильево	11:05	13:10
Васильево	Луковое	11:35	15:20
Панино	Васильево	12:05	14:25
Луковое	Васильево	12:30	16:10
Луковое	Санино	14:20	16:00
Васильево	Санино	16:25	17:15
Санино	Луковое	18:30	20:40

Определите самое раннее время, когда он может попасть в Васильево, и как ему нужно ехать.

7. Путешественник прибыл в посёлок Сычёво в 10:00 по местному времени и увидел следующее расписание автобусов.

Отправление из	Прибытие в	Время отправления	Время прибытия
Сычёво	Грибное	09:00	10:15
Мухино	Сычёво	09:15	10:25
Рогатое	Сычёво	10:10	12:25
Рогатое	Мухино	10:25	11:25
Сычёво	Рогатое	10:30	13:00
Грибное	Рогатое	10:40	11:45
Сычёво	Мухино	10:35	11:30
Грибное	Сычёво	10:55	11:25
Мухино	Рогатое	11:50	12:50
Рогатое	Грибное	12:00	13:20

Определите самое раннее время, когда он может попасть в посёлок Рогатое, и как ему нужно ехать.

8. Путешественник прибыл в посёлок Кунцево в полночь по местному времени и увидел следующее расписание автобусов.

Отправление из	Прибытие в	Время отправления	Время прибытия
Марьино	Кунцево	09:00	09:50
Кунцево	Борисово	09:55	11:00
Ручьи	Марьино	10:45	11:55
Ручьи	Кунцево	10:50	13:10
Ручьи	Борисово	10:55	12:00
Кунцево	Ручьи	11:00	13:20
Кунцево	Марьино	11:05	12:00
Борисово	Кунцево	11:20	12:25
Марьино	Ручьи	12:10	13:15
Борисово	Ручьи	12:25	13:25

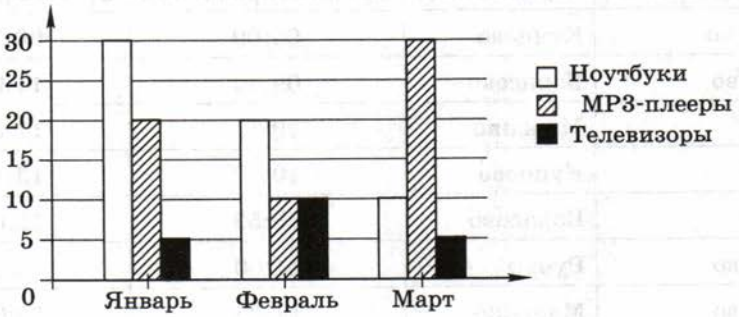
Определите самое раннее время, когда он может попасть в посёлок Ручьи, и как ему нужно ехать.

9. Путешественник прибыл в посёлок Моховое в полночь по местному времени и увидел следующее расписание автобусов.

Отправление из	Прибытие в	Время отправления	Время прибытия
Моховое	Лесное	07:40	08:50
Озёрное	Моховое	07:50	09:05
Лесное	Грибное	08:00	09:10
Лесное	Озёрное	09:15	10:25
Моховое	Грибное	09:25	10:30
Моховое	Озёрное	09:30	10:30
Лесное	Моховое	09:45	10:45
Грибное	Лесное	10:15	11:25
Озёрное	Лесное	11:15	12:25
Грибное	Моховое	11:50	12:55

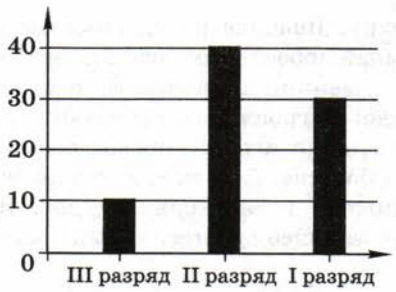
Определите самое раннее время, когда он может попасть в посёлок Лесное, и как ему нужно ехать.

10. На диаграмме показано, сколько ноутбуков, MP3-плейеров и телевизоров продала некоторая фирма в первые три месяца года (I квартал).



Какая из следующих диаграмм правильно отражает соотношение общего количества проданных товаров разных видов за весь I квартал?

11. В соревнованиях участвовали спортсмены из Москвы, Санкт-Петербурга и Мурманска, каждый из них имеет III, II или I разряд. На диаграмме 1) показано количество спортсменов, имеющих разные разряды, а на диаграмме 2) — соотношение спортсменов из разных городов.



Какие из этих утверждений следуют из анализа диаграмм:

а) все спортсмены, имеющие II разряд, могут быть москвичами;

б) все спортсмены из Мурманска могут иметь II разряд;

в) все спортсмены из Санкт-Петербурга могут иметь I разряд;

г) все спортсмены III разряда могут быть из Москвы?

12. В салоне продаются автомашины «Лада», «УАЗ» и «Ока» трёх цветов: красного, синего и зелёного. На диаграмме 1) показано количество машин разного цвета, а на диаграмме 2) — количество машин разных марок.

Какие из этих утверждений следуют из анализа диаграмм:

а) все автомобили «УАЗ» — зелёные;

б) среди автомобилей «Ока» нет красных;

в) все автомобили «Ока» — синие;

г) среди автомобилей «Лада» есть синие?

13. Два игрока играют в следующую игру. Вначале перед ними лежит куча из некоторого количества камней (обозначим его S). За один ход игрок может добавить в кучу 2 камня или увеличить количество

камней в куче в два раза. У каждого игрока есть неограниченное количество камней. Победителем считается игрок, первым получивший кучу, в которой 25 камней или больше. Для каждого значения S ($1 \le S \le 24$) определите, кто выиграет и за сколько ходов. Для $S = 7$ постройте дерево игры, показывающее стратегию выигрывающего игрока.

14. Два игрока играют в следующую игру. Вначале перед ними лежит куча из некоторого количества камней (обозначим его S). За один ход игрок может добавить в кучу 1 камень или увеличить количество камней в куче в три раза. У каждого игрока есть неограниченное количество камней. Победителем считается игрок, первым получивший кучу, в которой 55 камней или больше. Для каждого значения S ($1 \le S \le 54$) определите, кто выиграет и за сколько ходов. Для $S = 16$ постройте дерево игры, показывающее стратегию выигрывающего игрока.
15. Два игрока играют в следующую игру. Вначале перед ними лежит куча из некоторого количества камней (обозначим его S). За один ход игрок может добавить в кучу два камня, добавить в кучу три камня или увеличить количество камней в куче в два раза. У каждого игрока есть неограниченное количество камней. Победителем считается игрок, первым получивший кучу, в которой 30 камней или больше. Для каждого значения S ($1 \le S \le 29$) определите, кто выиграет и за сколько ходов. Для $S = 9$ постройте дерево игры, показывающее стратегию выигрывающего игрока.
16. **Игра Баше.** Два игрока играют в следующую игру. Вначале перед ними лежит куча из некоторого количества камней (обозначим его S). За один ход игрок может взять из кучи 1, 2 или 3 камня. Выигрывает тот, кто возьмет последний камень. Для каждого значения S ($1 \le S \le 15$) определите, кто выиграет и за сколько ходов. Для $S = 12$ постройте дерево игры, показывающее стратегию выигрывающего игрока.

§ 8
Этапы моделирования

Постановка задачи

Этап постановки задачи — самый важный при моделировании. Если здесь допущена ошибка, то фактически рассматривается совсем не та задача, которую нужно решить, и после завершения моделирования всё придётся начать заново.

Напомним, что с помощью моделирования можно решать задачи четырёх типов: это исследование оригинала, анализ, синтез и оптимизация. Для того чтобы задачу можно было решить, она должна быть **хорошо поставлена**, это значит, что:

- должны быть заданы все связи между исходными данными и результатом;
- должны быть известны все исходные данные;
- решение должно существовать;
- решение должно быть единственным.

Приведём примеры **плохо поставленных (некорректных)** задач.

Задача 1. «Уроки в школе начинаются в 8:00. В 10:00 к школе подъехал красный автомобиль. Определите, когда Вася выйдет играть в футбол». В этой задаче совершенно непонятна связь между исходными данными (время начала уроков, красный автомобиль) и результатом.

Задача 2. «Вася бросает мяч со скоростью 12 м/с. Где мяч впервые ударится о землю?» В этой задаче можно применить модель движения тела, брошенного под углом к горизонту. Но угол неизвестен, поэтому задача плохо поставлена (не хватает данных). Кроме того, неизвестно, где и куда Вася бросает мяч — если он находится в квартире, то возможно много вариантов развития событий.

Задача 3. «Решить уравнение $\sin x = 4$». Это уравнение не имеет решений.

Задача 4. «Найти функцию, график которой проходит через точки (0,0) и (1,1)». Через эти точки проходит бесконечное множество разных графиков, поэтому непонятно, какую именно функцию из всех возможных мы ищем.

Что делать, если полученная задача плохо поставлена? Решить её нельзя, поэтому остается уточнять условия и исходные данные. Если и это невозможно, нужно вводить *допущения* — упрощающие предположения, которые позволят сделать задачу хорошо поставленной.

Все дальнейшие рассуждения мы будем проводить для конкретной задачи: «Спортсмен Вася в синей кепке бросает белый мяч со скоростью 12 м/с. Под каким углом к горизонту ему нужно бросить мяч, чтобы попасть в жёлтую мишень?»

Ясно, что в таком виде задача плохо поставлена. Мы не можем её решить, потому что не знаем, где расположена мишень и из какой точки вылетел мяч. Поэтому надо дополнить условие:

«Мишень расположена на высоте 4 м на расстоянии 10 м от Васи. В момент броска мяч находится на высоте 2 м».

Всегда ли существует решение? Очевидно, что нет: если мишень находится очень далеко или очень высоко, Вася просто до неё не докинет мяч. Поэтому мы должны ввести допущение о том, что решение существует.

Единственно ли решение? Возможно, что нет (можно попасть в мишень, бросая мячик под разными углами), но нас интересует любое решение, поэтому считаем, что задача теперь хорошо поставлена.

Разработка модели

На этапе разработки информационной модели нужно:

- определить исходные данные, *существенные* для решения данной задачи;
- выбрать тип модели;
- построить формальную модель, отражающую только существенные свойства оригинала;
- разработать алгоритм исследования формальной модели;
- построить компьютерную модель.

Существенные данные. В первую очередь нужно выяснить, что влияет на полёт шарика, а что — нет. Легко понять, что наличие кепки и её цвет, а также цвета мячика и мишени никак не влияют на результат, поэтому включать их в модель не нужно[1]. Кроме того, для упрощения введём следующие *допущения*:

- мяч и мишень — материальные точки;
- мишень неподвижна;
- сопротивление воздуха не учитывается.

Выбор типа модели. При решении задачи могут использоваться несколько моделей разных типов. Например, для лучшего понимания полезно построить *графическую модель* задачи (рис. 2.12).

За начало координат удобно принять точку, откуда вылетает мяч. Обозначим через v_0 начальную скорость мяча, через H разницу высот ($H = 4 - 2 = 2$ м), а через S — расстояние до мишени ($S = 10$ м). Однако графическая модель не даёт ответа на постав-

[1] В другой ситуации они могут быть важны, например, когда Васю интересует в первую очередь свой внешний вид, а не попадание в мишень.

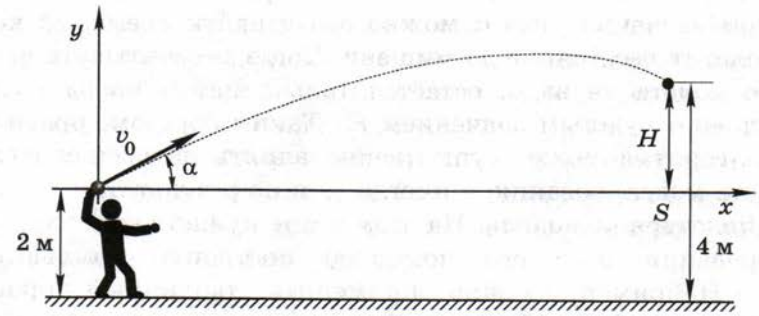


Рис. 2.12

ленный вопрос, поэтому для окончательных расчётов требуется построить математическую модель.

Формальная модель. В этой задаче формальная модель — это математическая модель движения тела, брошенного под углом к горизонту. Изменения координат описываются формулами:

$$x = v_0 \cdot t \cdot \cos\alpha, \quad y = v_0 \cdot t \cdot \sin\alpha - \frac{gt^2}{2},$$

где $g \approx 9{,}81$ м/с^2 — ускорение свободного падения. Задача сводится к тому, чтобы найти два неизвестных, t и α, при которых $x = S$ и $y = H$, т. е.

$$S = v_0 \cdot t \cdot \cos\alpha, \quad H = v_0 \cdot t \cdot \sin\alpha - \frac{gt^2}{2}.$$

Эти формулы и представляют собой математическую модель задачи. В них нет упоминания о Васе, мяче, мишени и т. п., есть только условные обозначения и связывающие их формулы, это формальная модель, записанная на языке математических формул.

Алгоритм исследования модели. Алгоритм — это чётко определённый план действий, который приводит к решению задачи. В данном случае требуется разработать алгоритм использования модели: как с помощью полученных уравнений найти угол, под которым Васе нужно бросить мяч?

Например, можно постепенно увеличивать угол, начиная с нуля, и каждый раз строить всю траекторию полёта. Если при каком-то значении угла мяч пролетел ниже мишени, а при следующем — выше неё, дальше можно применить метод половинного деления для уточнения решения.

Но лучше поступить более грамотно, значительно сократив вычисления. Дело в том, что из первого уравнения $S = v_0 \cdot t \cdot \cos\alpha$ при

известном значении угла α можно сразу найти время, за которое мяч пролетит расстояние до мишени. Тогда рассчитывать всю траекторию полёта не надо, остаётся только найти координату y и сравнить её с нужным значением H. Таким образом, правильный выбор алгоритма может существенно влиять на вычислительную сложность моделирования, а иногда — и на результат.

Компьютерная модель. На этом этапе нужно выбрать средство моделирования и с его помощью построить компьютерную модель. Например, можно применить табличный процессор (OpenOffice.org Calc, Microsoft Excel и т. п.), написать собственную программу на одном из языков программирования или использовать готовую среду для моделирования (например, Simulink или VisSim).

Тестирование модели

После построения модели её обязательно нужно протестировать (проверить).

Тестирование — это проверка модели на простых исходных данных с известным результатом.

Например, модель, описывающую сложение многозначных чисел, сначала нужно проверить на небольших числах, которые легко сложить вручную. Если получен неверный ответ, модель ошибочна, и её нужно переделывать. Другой пример: при моделировании накопления денег в банке сумма не должна меняться при нулевой ставке. Тестирование модели движения судна тоже начинается с простых задач: если штурвал поворачивают влево, судно должно уходить влево и наоборот.

Нужно понимать, что удачное тестирование модели не гарантирует, что она правильна; тестирование может только установить ошибочность модели. Чтобы доказать правильность модели, нужно проверить её при всех допустимых исходных данных (в том числе и при тех, для которых правильный ответ неизвестен), а это практически невозможно.

Выполним тестирование математической модели, построенной для нашей задачи:

$$x = v_0 \cdot t \cdot \cos\alpha, \quad y = v_0 \cdot t \cdot \sin\alpha - \frac{gt^2}{2}.$$

Согласно этим формулам:

- при $t = 0$ мяч находится в начале координат;
- при нулевой начальной скорости мяч падает вертикально вниз (координата x не меняется, а координата y уменьшается);
- при бросании вертикально вверх ($\alpha = 90°$, $\cos\alpha = 0$) координата x не меняется;
- при некотором t координата y начинает уменьшаться (парабола «загибается» вниз, мяч опускается).

Эти результаты не противоречат теории, поэтому можно считать, что тестирование прошло успешно. Тем не менее нельзя считать, что этим мы доказали правильность модели (подумайте, почему).

Эксперимент с моделью

Эксперимент — это испытание модели в тех условиях, которые нас интересуют (т. е. результатов мы заранее не знаем). Например, для модели накопления денег в банке задаётся ненулевая ставка (процент ежегодного увеличения); движение судна моделируется с учётом случайных помех — морского волнения и ветра; модель сложения «длинных» чисел применяется к многоразрядным числам и т. п.

Можно ли слепо верить результатам моделирования? Конечно нет, ведь при тестировании мы не проверяли (и не могли проверить!) работу модели в этих условиях. Поэтому необходим анализ результатов.

Анализ результатов

Во-первых, нужно убедиться, что результаты моделирования не противоречат известным из теории фактам, например не нарушаются законы сохранения.

Во-вторых, необходимо проверить результаты моделирования на реальном объекте — провести эксперимент с оригиналом. Если нам удалось решить поставленную задачу (поведение оригинала соответствует[1] поведению модели), можно считать модель адекватной и работу законченной.

Если же результаты нас не устраивают (поведение объекта и оригинала значительно различаются, задачу решить не уда-

[1] Часто считается, что допустимо расхождение результатов эксперимента и моделирования не более 10%.

лось), требуется вернуться к одному из предыдущих этапов и повторить моделирование, например:

- изменить алгоритм или условия моделирования;
- изменить модель: учесть дополнительные свойства, которые ранее считались несущественными;
- изменить постановку задачи (если выяснилось, что решили не ту задачу, которую нужно было решать).

Возможно, что несоответствие вызвано принятыми допущениями. Так в задаче с полётом мяча полезно ответить на следующие вопросы, которые помогут выяснить причину неудачи:

- Всегда ли Вася сможет попасть в мишень?
- Что изменится, если начальная скорость будет отличаться от заданной?
- Что изменится, если мяч и мишень не считать материальными точками?
- Насколько сильно сопротивление воздуха влияет на результат?
- Что изменится, если мишень качается? И т. д.

Вопросы и задания

1. Что такое хорошо и плохо поставленные задачи?
2. Что делать, если задача плохо поставлена?
3. Приведите примеры хорошо и плохо поставленных задач (кроме тех, что даны в учебнике).
4. Что входит в понятие «разработка модели»?
5. Как выделить существенные свойства, которые нужно учесть в модели?
6. Какие исходные данные в формулировке приведённой задачи существенны, а какие — нет? Каких данных не хватает?
   «Электрик в зелёном комбинезоне едет на красном автомобиле «Лада-Калина» из Москвы в Воронеж. Его средняя скорость равна 90 км/ч, причём каждые 2 часа электрик отдыхает по 15 минут. Когда он приедет в Воронеж?»
7. Как вы думаете, почему при решении задачи часто используется несколько моделей разных типов? Приведите примеры.
8. Что такое алгоритм моделирования? Почему он важен?
9. Что такое тестирование модели? Почему оно важно?
10. Что такое эксперимент? Чем он отличается от тестирования?
11. Можно ли доказать правильность (или ошибочность) модели с помощью тестирования? Приведите примеры и обоснуйте ответ.
12. Зачем нужен анализ результатов эксперимента? Какие выводы могут быть сделаны на этом этапе?

13. В чём может быть причина неудачи при решении задачи с помощью моделирования?
14. Что делать, если после анализа результатов моделирования был сделан вывод, что решить задачу не удалось?

Подготовьте сообщение:

а) «Зачем и как вводить допущения при моделировании?»

б) «Зачем тестировать модель?»

в) «Программные средства для моделирования»

§ 9
Моделирование движения

На уроках физики вы изучали в основном две модели движения: равномерное (когда равнодействующая всех сил равна нулю) и равноускоренное (когда равнодействующая постоянна). Например, движение тела, брошенного под углом к горизонту, обычно раскладывается на равномерное движение по горизонтали и равноускоренное движение по вертикали (под действием силы тяжести).

В реальных ситуациях силы, действующие на систему, постоянно изменяются, поэтому ускорение тоже будет переменным, и упомянутые простейшие модели использовать нельзя — они неадекватны. Для примера мы разберём задачу, в которой важную роль играют силы сопротивления среды.

Движение с сопротивлением

Рассмотрим движение мяча, который представляет собой шар радиуса r и массы m, брошенного вертикально вверх со скоростью v_0. Нужно найти, на какую высоту поднимется мяч, и скорость, с которой он упадёт на землю.

На мяч в полёте действуют две силы (рис. 2.13):

- сила тяжести $\vec{G}$ (она направлена вертикально вниз);
- сила сопротивления $\vec{F}$, которая направлена противоположно вектору текущей скорости $\vec{v}$.

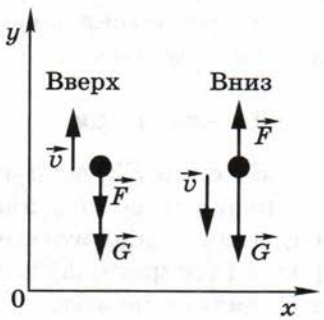


Рис. 2.13

Будем работать с проекциями всех сил на вертикальную ось, за положительное направление примем направление вверх. Тогда проекция силы тяжести $G = -mg$, где g — ускорение свободного падения.

Сила сопротивления зависит от формы и размеров тела, его скорости и свойств среды. На малых скоростях сила сопротивления пропорциональна скорости, а при увеличении скорости пропорциональна её квадрату:

$$F = \frac{\rho v^2}{2} \cdot C \cdot S,$$

где ρ — плотность среды (для воздуха $\rho \approx 1{,}23$ кг/м^3), v — скорость тела, S — площадь поперечного сечения (для шара $S = \pi \cdot r^2$), а коэффициент сопротивления C зависит от формы тела и определяется экспериментально (для шара $C \approx 0{,}4$).

Чтобы показать, что вектор силы сопротивления всегда направлен противоположно вектору скорости, часто используют такую запись:

$$\vec{F} = -\frac{\rho |v| \vec{v}}{2} \cdot C \cdot S.$$

Ускорение определяется по второму закону Ньютона:

$$\vec{a} = \frac{\vec{G} + \vec{F}}{m}.$$

Сила сопротивления F зависит от скорости, которая меняется. Поэтому ускорение тоже будет переменным, и модель равноускоренного движения применить нельзя. Исследовать такое движение теоретически довольно сложно, поэтому мы используем *имитационную модель*.

Дискретизация

В конце XX века задачи моделирования законов движения решались с помощью *аналоговых* вычислительных машин, которые работали с аналоговыми сигналами. Мы будем использовать цифровой (дискретный) компьютер, поэтому нужно выполнить **дискретизацию** — перейти к дискретной модели, которая описывает движение мяча в отдельные моменты времени с некоторым шагом δ:

$$t = 0,\ \delta,\ 2\delta,\ 3\delta,\ \ldots$$

Нам нужно построить такую модель, которая позволит вычислить скорость и высоту мяча в момент $t_{i+1} = (i + 1)\cdot\delta$ при известных данных в момент $t_i = i\cdot\delta$. Чтобы использовать известные формулы для равноускоренного движения, мы предположим, что *на отрезке* $[t_i, t_{i+1}]$ *ускорение изменяется очень мало* и можно считать его примерно постоянным. Конечно, фактически это не так, но при маленьком интервале δ ошибка будет небольшой.

Как выбрать δ? Понятно, что чем меньше δ, тем точнее модель и меньше ошибка моделирования. Вместе с тем при уменьшении δ возрастает объём расчётов, поэтому нужно попытаться найти «золотую середину». Можно, например, начать с большого δ (допустим, 1 с) и постепенно уменьшать его, повторяя моделирование до тех пор, пока результаты практически перестанут изменяться.

Итак, предположим, что в момент t_i скорость мяча[1] равна v_i, и он находится на высоте y_i. Определим силу сопротивления и ускорение в этот момент:

$$F_i = -\frac{\rho|v_i|v_i}{2}\cdot C\cdot S, \quad a_i = \frac{G + F_i}{m} = -g + \frac{F_i}{m}.$$

Предполагая, что мяч летел с постоянным ускорением a_i в течение всего интервала величиной δ, вычислим его скорость и координату в момент t_{i+1} по формулам для равноускоренного движения:

$$v_{i+1} = v_i + a_i\cdot\delta, \quad y_{i+1} = y_i + v_i\cdot\delta + \frac{a_i\cdot\delta^2}{2}.$$

Эта дискретная модель описывает изменение скорости и высоты мяча через интервал δ.

Компьютерная модель

Теперь можно выполнить моделирование, используя табличный процессор или собственную программу. Каждый из этих подходов имеет преимущества и недостатки: в программе удобнее изменять шаг моделирования, в табличном процессоре удобно строить графики.

Если использовать язык программирования, в начале программы нужно определить все исходные данные (как константы или переменные). Затем задаются начальные значения времени, скорости и координаты (высоты):

```
t:=0; v:=v0; y:=0;
```

[1] Здесь и далее мы фактически рассматриваем проекции скорости, ускорения и сил на вертикальную ось.

Цикл должен завершиться в тот момент, когда высота станет отрицательной (это значит, что мяч упал на землю).

```
нц пока y>=0
  F:=-ro*abs(v)*v*C*S/2          | сила сопротивления
  a:=-g+F/m                      | ускорение
  y:=y+v*delta+a*delta*delta/2   | координата (высота)
  v:=v+a*delta                   | скорость
  t:=t+delta                     | время
кц
```

В переменной *delta* хранится шаг дискретизации δ, а в переменной ro — плотность воздуха. Обратите внимание, что нужно сначала изменить координату y, а только потом — скорость v, иначе (при обратном порядке вычислений) для расчёта высоты будет использоваться *новое* значение скорости, и в модель будет добавлена еще одна неточность.

Что мы получим после окончания цикла? В переменной v будет скорость в момент приземления (отрицательная величина, потому что мячик летит вниз), а в переменной t — время полёта. Для того чтобы найти максимальную высоту h, на которую поднялся мяч, нужно добавить в цикл оператор

```
если y>h то
  h:=y
все
```

Начальное значение для h можно задать равным 0.

Вопросы и задания

1. Почему во многих задачах не учитывают сопротивление среды? В каких случаях это можно делать?
2. Что такое дискретизация? Почему она необходима?
3. Какие допущения использовались при дискретизации рассмотренной в параграфе модели?
4. Как выбрать шаг дискретизации?
5. Какие средства можно использовать для компьютерного моделирования в рассмотренной задаче? В чём их достоинства и недостатки?
6. Измените приведённую в параграфе программу так, чтобы в ней использовался цикл с постусловием. Сравните это решение с вариантом в параграфе.
7. Объясните, как в программе определить максимальную высоту подъёма мяча.

Подготовьте сообщение

«Программные средства для моделирования движения»

Задачи

1. Напишите программу, которая моделирует полёт мяча при:
$r = 33$ мм, $m = 150$ г, $v_0 = 20$ м/с, $\delta = 0{,}1$ с.
Остальные необходимые данные есть в тексте параграфа. Выполните следующие задания.
   а) Определите время полёта, максимальную высоту подъёма мяча и скорость в момент приземления.
   б) Вычислите время полёта и максимальную высоту подъёма мяча, используя модель движения без сопротивления воздуха:
$$t = \frac{2v_0}{g}, \quad h = \frac{v_0^2}{g}, \quad v = -v_0.$$
   Сравните эти результаты с полученными при моделировании с учётом сопротивления. Можно ли в этой задаче пренебречь сопротивлением?
   в) С помощью табличного процессора постройте траекторию движения мяча, а также графики изменения скорости, ускорения и силы сопротивления.
   г) Уменьшите шаг до 0,01 с и повторите моделирование; сделайте выводы по поводу выбора шага в данной задаче.

*2. Выполните моделирование движения мяча, брошенного под углом 45° к горизонту:
   - определите время полёта, максимальную высоту и дальность полёта мяча, скорость в момент приземления;
   - сравните результаты со случаем, когда сопротивление воздуха не учитывается; сделайте выводы.

3. Парашютист массой 90 кг разгоняется в свободном падении до скорости 10 м/с и на высоте 50 м раскрывает парашют, площадь которого 55 м^2. Коэффициент сопротивления парашюта равен 0,9. Выполните следующие задания.
   а) Постройте графики изменения скорости и высоты полета в течение первых 4 секунд.
   б) Определите, с какой скоростью приземлится парашютист.
   в) Сравните результаты моделирования с установившимся значением скорости, вычисленным теоретически.

§ 10
Математические модели в биологии

Многие идеи в области моделирования были предложены биологами. Достаточно вспомнить, что родоначальником общей теории систем (системного подхода) был биолог Людвиг фон Берталанфи. В этом параграфе мы познакомимся с моделями некоторых биологических систем.

Модель неограниченного роста

Одна из задач, которые решают биологи, — изучение изменения численности животных в некоторой области. Обычно их пересчитывают раз в год, поэтому модель изменения численности получается *дискретной* — с её помощью можно определить численность с интервалом 1 год.

Обозначим через N_0 начальную численность, а через N_i — численность в i-й год с момента начала наблюдений. Количество родившихся и умерших животных пропорционально численности, поэтому годовой прирост равен $k_р \cdot N_i - k_с \cdot N_i$, где $k_р$ и $k_с$ — коэффициенты рождаемости и смертности. Тогда количество животных в $(i + 1)$-й год может быть вычислено через их количество N_i в предыдущему году:

$$N_{i+1} = N_i + k_р \cdot N_i - k_с \cdot N_i = (1 + K) \cdot N_i,$$

где $K = k_р - k_с$ — коэффициент прироста. Это и есть математическая модель развития **популяции** («населения») **животных**. Коэффициенты $k_р$ и $k_с$ (так же как и другие коэффициенты в рассматриваемых далее моделях) обычно определяются экспериментально.

При $K = 0$ (рождаемость равна смертности) количество животных не меняется, при $K < 0$ ($k_р < k_с$) животные вымирают, а при $K > 0$ ($k_р > k_с$) их число бесконечно увеличивается (рис. 2.14).

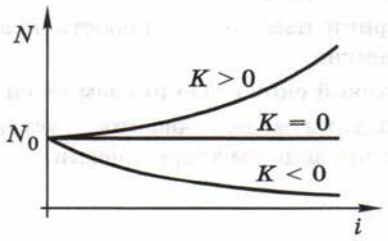


Рис. 2.14

Поэтому эту модель называют **моделью неограниченного роста.** Английский учёный и экономист Томас Мальтус использовал её для описания роста населения Земли, поэтому эту модель иногда называют *моделью Мальтуса.*

Недостаток этой модели в том, что она не учитывает ограниченность ресурсов (например, пищи), влияние других видов и изменяющихся природных условий, из-за которых изменяются коэффициенты рождаемости и смертности. Поэтому модель Мальтуса адекватна только при небольших интервалах наблюдения.

Модель ограниченного роста

Бельгийский математик Пьер Ферхюльст предложил ввести *максимальную численность популяции* L и построить модель так, чтобы численность животных не превышала этой величины. Как только численность приближается к L, коэффициент прироста уменьшается и рост замедляется. Модель Ферхюльста основана на той же самой формуле, что и модель Мальтуса:

$$N_{i+1} = (1 + K_L) \cdot N_i,$$

но теперь коэффициент прироста K_L зависит от численности N_i:

$$K_L = K \cdot \frac{L - N_i}{L},$$

где K — начальный коэффициент (при нулевой численности). Видно, что при увеличении N_i коэффициент K_L уменьшается и при $N_i = L$ становится равен нулю.

Нередко человек искусственно разводит животных, например, в рыбоводческих хозяйствах или на зверофермах. В этом случае ежегодно часть животных (обозначим её R) отлавливается, а оставшиеся размножаются и поддерживают популяцию. Для того чтобы определить допустимый отлов, при котором популяция сохраняется (не вымирает), используют **модель с отловом:**

$$N_{i+1} = (1 + K_L) \cdot N_i - R, \quad K_L = K \cdot \frac{L - N_i}{L}.$$

На рисунке 2.15 показаны графики изменения численности популяции для разных моделей при $N_0 = 100$, $K = 0{,}5$, $L = 1000$ и $R = 40$. Видно, что в самом начале модели ограниченного и неограниченного роста дают близкие результаты, т. е. модель неограниченного роста адекватна.

Рис. 2.15

Согласно модели ограниченного роста, со временем численность популяции приближается к L, а при отлове устанавливается численность немного меньше L. Вы можете рассчитать это значение самостоятельно, приняв, что в состоянии равновесия $N_{i+1} = N_i$.

Взаимодействие видов

Все модели, которые мы рассматривали выше, описывали изменение одного вида. На самом деле на одной территории всегда живут несколько видов животных, которые соперничают друг с другом. В простейшем случае это соперничество за еду (например, между белками и бурундуками). Однако наиболее интересна **модель «хищник — жертва»**, в которой один вид — хищники (например, щуки), а второй — их пища (караси).

Сначала построим модели двух видов по отдельности. Изменение численности карасей N_i описывается уже известной нам моделью ограниченного роста:

$$N_{i+1} = (1+K_L)\cdot N_i, \qquad K_L = K\cdot\frac{L-N_i}{L}.$$

Будем считать, что щуки могут питаться только карасями[1]. Тогда без карасей они просто погибают, и модель изменения их численности имеет вид:

$$Z_{i+1} = (1 - D)\cdot Z_i,$$

где D — коэффициент смертности щук (для простоты будем считать его постоянным).

---

[1] Конечно, в самом деле, это предположение чаще всего неверно, но мы используем его для простоты модели.

Объединив две модели, мы не получаем *систему*, потому что никак не учитывается связь между численностью карасей и щук, живущих в одном водоёме. В результате численность карасей со временем станет равна L, а щуки вымрут, хотя вокруг полно еды.

Чтобы построить системную модель, предположим, что частота встреч карасей и щук пропорциональна произведению их численностей: $N_i \cdot Z_i$. В результате этих встреч каждый год гибнет $b_N \cdot N_i \cdot Z_i$ карасей, но появляется $b_Z \cdot N_i \cdot Z_i$ новых щук. Здесь b_N и b_Z — это некоторые коэффициенты, которые определяются экспериментально. Таким образом, полная *модель-система* состоит из двух связанных уравнений:

$$N_{i+1} = (1 + K_L - b_N \cdot Z_i) \cdot N_i,$$

$$Z_{i+1} = (1 - D + b_Z \cdot N_i) \cdot Z_i.$$

На рисунке 2.16 показаны графики изменения численности карасей и щук, полученные при $L = 100$, $N_0 = 50$, $Z_0 = 10$, $d = 0{,}8$, $b_N = 0{,}01$ и $b_Z = 0{,}012$. Сначала мы видим «переходный период», когда численность карасей и щук довольно сильно меняется, а затем наступает равновесие — количество рыб обоих видов остаётся примерно постоянным. Обратите внимание, что численность карасей не достигает предельного значения L — щуки мешают. Вместе с тем количество щук не растёт бесконтрольно — не хватает еды. Таким образом, в природных системах соблюдается равновесие.

Рис. 2.16

Обратная связь. Саморегуляция

Модели неограниченного и ограниченного роста популяции животных, которые изучались в предыдущем пункте, можно изобразить в виде схем следующим образом (рис. 2.17).

В первом случае (модель неограниченного роста) коэффициент прироста K жёстко задан и не меняется. Такая модель адекватна только при малых интервалах наблюдения.

В модели ограниченного роста коэффициент прироста K_L не фиксирован, а меняется в зависимости от ситуации. На него вли-

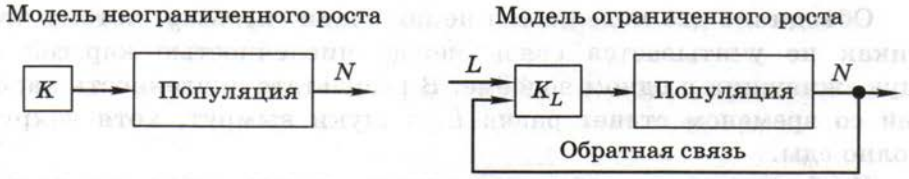


Рис. 2.17

яют две величины — максимальная численность популяции L и текущая численность N. Переходя на язык теории управления (см. § 4), величину L можно рассматривать как «цель», заданную природой, она определяется природными условиями (количеством корма, погодой и т. п.).

Зависимость коэффициента K_L от численности N — это *обратная связь*, с помощью которой регулируется численность. Если $N > L$, т. е. фактическая численность меньше «заданной», начинается прирост численности ($K_L > 0$). Ситуация, когда $N > L$, — это перенаселение, при котором $K_L < 0$ и животные начинают вымирать. Как мы видели, при $N = L$ получается нулевой прирост, численность не меняется.

Рис. 2.18

Если в какой-то момент популяция резко сокращается (например, рыбы отравились химическими веществами, сброшенными в воду), после устранения проблемы численность снова восстанавливается и достигает максимального уровня L (рис. 2.18). Так работает саморегуляция.

Саморегуляция — это способность системы поддерживать своё внутреннее состояние за счёт связей между элементами.

Саморегуляция — очень важное свойство живых организмов, которое позволяет им выживать при изменениях внешних условий. Например, теплокровные животные поддерживают постоянную температуру тела, бактерии и водоросли поддерживают постоянную солёность и состав морской воды.

Строго говоря, если рассмотреть блок «популяция» на схеме как подсистему, мы тоже найдём в ней внутреннюю обратную связь, поскольку численность на очередном шаге N_{i+1} зависит от численности на предыдущем шаге N_i. На схеме (рис. 2.19) квадратик с крестиком обозначает умножение, а блок с буквой τ — *запаздывание* (задержку), которое позволяет в течение одного шага помнить последнее значение N_i.

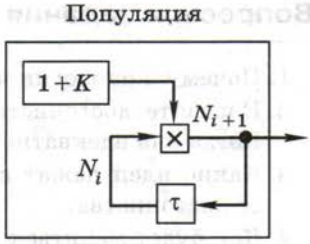


Рис. 2.19

Несмотря на обратную связь, возможности саморегуляции в такой системе очень ограничены: при $K > 0$ численность будет бесконечно расти (нет саморегуляции), а при $K < 0$ популяция вымирает (равновесие наступает только при $N = 0$).

Пример саморегуляции в системе «хищник — жертва» мы рассматривали в предыдущем пункте. В этой системе три канала обратной связи (не считая внутренних, в каждой популяции) (рис. 2.20).

Рис. 2.20

Саморегуляция «работает» только при небольших отклонениях от состояния равновесия. Если серьезно нарушить баланс, система не сможет восстановиться. В последние столетия человек все активнее вмешивается в жизнь природы, в результате сокращаются леса и зоны с плодородной почвой, изменяется климат, вымирают все новые и новые виды животных. Нарушение саморегуляции в масштабе Земли может поставить под угрозу жизнь всего человечества.

Вопросы и задания

1. Почему в биологии часто используются дискретные модели?
2. Назовите достоинства и недостатки модели неограниченного роста. Когда она адекватна?
3. Какие идеи лежат в основе модели ограниченного роста? Назовите её достоинства.
4. Как будет меняться численность согласно модели ограниченного роста, если в начальный момент $N > L$? Объясните это с точки зрения биологии.
5. Как с помощью моделирования определить допустимый отлов?
6. Объясните, как строится модель «хищник — жертва».
7. Можно ли считать модель «хищник — жертва» системной? Почему?
8. Как вы думаете, возможны ли другие модели взаимного влияния видов? Приведите примеры.
9. Что такое обратная связь? В чём достоинство моделей с обратной связью?
10. Объясните, как работает обратная связь в модели ограниченного роста.
11. Что такое саморегуляция? Приведите примеры.
12. Можно ли сказать, что в модели ограниченного роста присутствует обратная связь? А саморегуляция?
13. Какие обратные связи существуют в модели «хищник — жертва»?

Подготовьте сообщение:

а) «Модели взаимодействия видов в биологии»

б) «Что такое саморегуляция?»

Задачи

1. Постройте графики изменения численности популяции для моделей ограниченного и неограниченного роста при $N = 100$, $K = 0{,}5$ и $L = 1000$. Определите, через сколько интервалов наблюдения модель неограниченного роста перестаёт быть адекватной (отклонение от модели ограниченного роста составляет более 10%).
2. Для предыдущей задачи выполните моделирование с учётом отлова ($R = 40$):
   - определите количество животных в состоянии равновесия теоретически и по результатам моделирования; зависит ли оно от начальной численности?
   - определите, на что влияет начальная численность животных;
   - найдите максимальный отлов R, при котором популяция не вымирает.

*3. При эпидемии гриппа число больных N изменяется по формуле

$$N_{i+1} = N_i + Z_i - V_i,$$

где Z_i — количество заболевших в i-й день, а V_i — количество выздоровевших в тот же день. Число заболевших рассчитывается согласно модели ограниченного роста:

$$Z_{i+1} = K \cdot \frac{L - N_i - W_i}{L} \cdot N,$$

где L — общая численность жителей, K — коэффициент роста и W_i — число переболевших (тех, кто уже переболел и выздоровел и поэтому больше не заболеет):

$$W_{i+1} = W_i + V_i.$$

Считается, что все заболевшие выздоравливают через 7 дней и больше не болеют. Выполните моделирование при $L = 1000$ и $K = 0{,}5$, считая, что в начале эпидемии заболел 1 человек. Ответьте на следующие вопросы:

- Когда закончится эпидемия?
- Сколько человек переболеет, а сколько вообще не заболеет гриппом?
- Каково максимальное число больных в один день?

4. Выполните моделирование системы «хищник — жертва» при параметрах, указанных в тексте параграфа. Ответьте на следующие вопросы:
   - Сколько карасей и щук живут в водоёме в состоянии равновесия?
   - Что влияет на количество рыб в состоянии равновесия: начальная численность или значения коэффициентов модели?
   - На что влияет начальная численность?
   - При каких значениях коэффициентов модель становится неадекватна? В чём это выражается?
   - При каких значениях коэффициентов щуки вымирают, а численность карасей достигает предельно возможного значения? Как вы можете объяснить это с точки зрения биологии?

5. Белки и бурундуки живут в одном лесу и едят примерно одно и то же (конкурируют за пищу). Модель, описывающая изменение численности двух популяций, имеет вид:

$$N_{i+1} = \left[1 + K_N \cdot \frac{L_N - N_i}{L_N}\right] \cdot N_i - D_N \cdot M_i,$$

$$M_{i+1} = \left[1 + K_M \cdot \frac{L_M - M_i}{L_M}\right] \cdot M_i - D_M \cdot N_i.$$

Здесь N и M — численность белок и бурундуков; L_N и L_M — их максимальные численности; K_N и K_M — коэффициенты прироста; D_N и D_M — коэффициенты взаимного влияния. Выполните моделирование при $N_0 = 10$, $M_0 = 20$, $L_N = 70$, $L_M = 50$, $K_N = K_M = 0{,}7$ и $D_N = D_M = 0{,}1$. Постройте графики изменения численности обоих видов. Ответьте на следующие вопросы:

- Является ли эта модель системной?
- Какова численность белок и бурундуков в состоянии равновесия?
- Что влияет на состояние равновесия?
- На что влияет начальная численность животных?
- При каком значении коэффициента D бурундуки вымрут через 25 лет? (Используйте подбор параметра.)
- При каких значениях коэффициентов модель становится неадекватной?
- Какую аналогичную модель взаимного влияния трёх видов вы можете предложить?

§ 11
Системы массового обслуживания

Что такое системы массового обслуживания?

Методы математического моделирования проникают во все сферы, в том числе и в экономику. Предприятия сферы обслуживания (магазины, банки, узлы телефонной связи, станции «скорой помощи» и т. п.) — очень интересные, но очень сложные объекты. На вход таких систем поступает поток **заявок на обслуживание** — это могут быть, например, покупатели, входящие в магазин. Все заявки нужно обработать, для этого предприятие имеет несколько точек обслуживания (например, касс в магазине). При этом большую роль играет **случайность**:

- заявки поступают через случайные промежутки времени;
- время обслуживания — случайная величина.

Такие системы называют **системами массового обслуживания**, для их исследования используются **вероятностные модели**.

Теоретические модели этих систем, как правило, достаточно сложны, поэтому мы будем использовать имитационные модели, которые позволяют найти решение задачи с помощью многократных компьютерных экспериментов.

Модель обслуживания в банке

Рассмотрим достаточно простую модель работы банка. Клиенты входят через случайные промежутки времени, их обслуживают несколько кассиров, причём время обслуживания – также случайная величина. Требуется определить, сколько кассиров нужно для того, чтобы клиент стоял в очереди не более M минут.

В таком виде задача некорректна (плохо поставлена) — неясно, как определить длину очереди. Поэтому нужно вводить допущения, которые позволят сделать задачу хорошо поставленной и решить её.

Сначала построим **детерминированную модель**, в которой случайность не учитывается. Будем считать, что за 1 минуту входит ровно P клиентов, причём каждый клиент обслуживается ровно T минут.

Если количество касс равно K, за T минут будет обслужено ровно K клиентов. За это же время в банк войдут $P \cdot T$ новых клиентов. Несложно понять, что если $K < P \cdot T$, клиенты входят быстрее, чем их успевают обслуживать, поэтому очередь будет постоянно расти. Если же $K \geq P \cdot T$, очереди вообще не будет, из этого условия и нужно выбирать количество касс. Например, при $P = 2$ и $T = 5$ для обслуживания нужно не менее 10 касс.

Теперь усложним модель: введем в неё случайные события. Будем считать, что:

- за 1 минуту в банк входит случайное число клиентов, от 0 до P_{max};
- на обслуживание клиентов требуется от T_{min} до T_{max} минут.

Для моделирования нужно ещё определить, как распределены случайные данные (количество входящих за 1 минуту и время обслуживания) внутри заданных интервалов. Для простоты мы будем считать, что в обоих случаях распределение *равномерное*. Тогда можно применить стандартные генераторы случайных чисел, которые дают именно равномерно распределённые значения.

Будем использовать имитационную модель — выполним моделирование ситуации для достаточно большого интервала времени L (например, для 8-часового рабочего дня $L = 8 \cdot 60 = 480$ минут).

Число клиентов, находящихся в помещении банка, изменяется по закону:

$$N_{i+1} = N_i + P_i - R_i,$$

где P_i — количество клиентов, вошедших за i-ю минуту, а R_i — количество клиентов, обслуженных за это время. Считаем, что N клиентов равномерно распределяются по K кассам, так что средняя длина очереди равна $Q = \frac{N}{K}$, а среднее время ожидания равно $Q \cdot T$.

Количество вошедших P_i — это случайное целое число в интервале от 0 до P_{max}. На школьном алгоритмическом языке и на Паскале его можно получить так:

```
P:=irand(0,PMax)                    P:=random(PMax + 1);
```

Определить число обслуженных R_i оказывается не так просто. Если кассир обслуживает клиента за T минут, то можно считать, что за 1 минуту он сделает часть работы, равную $\frac{1}{T}$. Если предположить, что скорость работы кассиров одинакова, то K касс за 1 минуту обслужат $\frac{K}{T}$ клиентов. Чтобы учесть случайное время обслуживания, величину T будем определять заново для каждой минуты случайным образом. Это случайное вещественное число в интервале от T_{min} до T_{max}:

```
T:=rand(Tmin,Tmax)                  T:=Tmin+random*(Tmax-Tmin);
```

Тогда можно рассчитать значение среднего времени ожидания по формуле

$$\Delta t = Q \cdot T = \frac{N}{K} \cdot T.$$

Это значение будет меняться, поскольку N и T — случайные величины. Именно поэтому невозможно *гарантировать*, что клиент точно не будет ждать больше положенного времени M. Вместо этого мы можем (с помощью имитационного моделирования) подсчитать, какую *долю времени* оценка интервала ожидания Δt будет больше, чем M минут. Для определения этой доли нужно подсчитать количество «плохих» минут и разделить его на общее время моделирования L.

Если доля «плохого» времени получилась, например, 0,95 (95%), то клиент практически всегда вынужден будет ждать слишком долго, и количество касс нужно увеличивать. Если эта доля равна 0,05, то время ожидания будет больше допустимого только в 5% случаев, такой результат можно считать приемлемым.

Таким образом, нужно составить программу, которая запрашивает количество касс, выполняет моделирование работы банка в течение рабочего дня и выдает долю «плохих» минут. Сначала задаём начальные значения переменных:

```
Pmax:=4   | максимальное число входящих за 1 мин
Tmin:=1   | минимальное время обслуживания
Tmax:=9   | максимальное время обслуживания
L:=480    | период моделирования в минутах (8 часов)
M:=15     | допустимое время ожидания
N:=0      | сначала в банке никого нет
count:=0  | счетчик "плохих" минут
```

Затем нужно ввести количество кассиров K. Основной цикл моделирует работу банка в течение L минут:

```
нц для i от 1 до L
  P:=irand(0,PMax)
  T:=rand(Tmin,Tmax)
  R:=int(K/T)
  N:=N+P-R
  если N<0 то N:=0 все
  dT:=N/K*T
  если dT>M то
    count:=count+1
  все
кц
```

```
for i:=1 to L do begin
  P:=random(PMax+1);
  T:=Tmin+random*(Tmax-Tmin);
  R:=round(K/T);
  N:=N+P-R;
  if N<0 then N:=0;
  dT:=N/K*T;
  if dT>M then
    count:=count+1;
end;
```

На каждом шаге последовательно вычисляются:

- случайное число входящих клиентов P;
- случайное время обслуживания T;
- число клиентов R, обслуженных за эту минуту;
- число клиентов в помещении банка N; если оно получилось отрицательным, то используется нулевое значение;
- время ожидания; если оно больше допустимого времени M, увеличивается счетчик «плохих» минут.

После окончания работы цикла остаётся вывести результат — отношение $count/L$, которое представляет собой долю «плохих» минут. Нужно провести серию экспериментов с моделью и выбрать минимальное значение K, при котором доля «плохих» минут будет менее 5%. Помните, что в модели используются случайные числа, поэтому при каждом новом запуске программы результаты расчётов будут немного меняться. Теперь вы можете написать полную программу и найти с её помощью нужное число касс.

Обратите внимание, что мы выбрали исходные данные так, чтобы среднее число входящих клиентов и среднее время обслуживания совпадали с теми же данными для детерминированной модели. Поэтому интересно сравнить результаты, полученные с помощью моделей двух типов: детерминированной и вероятностной. Скорее всего, они окажутся близкими, хотя и будут немного различаться. Тогда возникает вопрос: какая модель даёт более точный результат? Ответить на него можно только с помощью *эксперимента*, проведённого в *реальном банке*.

Вопросы и задания

1. Какие системы называются системами массового обслуживания?
2. В чем сложность исследования систем массового обслуживания?
3. Как вы думаете, какая модель точнее описывает ситуацию в системе массового обслуживания: детерминированная или вероятностная? Обоснуйте свой ответ.
4. Как моделируются случайные события в вероятностной модели работы банка?
5. Как вы думаете, верно ли, что число входящих за минуту клиентов и время обслуживания имеют равномерное распределение?
6. Объясните, как вычисляется среднее время ожидания в вероятностной модели.
7. Объясните, как с помощью вероятностной модели определить нужное количество касс.

Задача

Используя вероятностную модель работы банка, определите необходимое количество кассиров при исходных данных, приведённых в тексте параграфа.

Практические работы

Работа № 6 «Моделирование работы процессора»
Работа № 7 «Моделирование движения»
Работа № 8 «Моделирование популяции»
Работа № 9 «Моделирование эпидемии»
Работа № 10 «Модель "хищник — жертва"»
Работа № 11 «Саморегуляция»
Работа № 12 «Моделирование работы банка»

ЭОР к главе 2 на сайте ФЦИОР (http://fcior.edu.ru)

- Построение информационных моделей ИС
- Назначение и виды информационных моделей

Самое важное в главе 2

- Модель — это объект-заменитель, который используется для изучения другого объекта, процесса или явления.
- Для каждого объекта можно построить множество разных моделей, соответствующих разным задачам. Одна и та же модель может описывать различные объекты.
- Адекватность модели — это соответствие свойств модели и оригинала в рассматриваемой задаче. Доказать адекватность можно только экспериментом с оригиналом.
- Система — это группа объектов и связей между ними, выделенных из среды и рассматриваемых как одно целое. За счёт связей система обладает особыми свойствами, которые не выводятся из свойств её компонентов.
- Основные этапы моделирования — это постановка задачи, разработка модели, тестирование модели, эксперимент и анализ результатов.
- Необходимый шаг при построении компьютерных моделей — дискретизация.
- Модели сложных систем, как правило, должны учитывать случайность событий в реальной жизни.

Глава 3
Базы данных

§ 12
Информационные системы

Основные понятия

Первоначально вычислительные машины разрабатывались для выполнения расчётных задач — решения систем уравнений, определения траекторий полёта снарядов и спутников и т. п. Однако сейчас значительную часть времени компьютеры заняты обработкой нечисловых данных — текстов, изображений, звука и видео. В нашу жизнь вошли информационные системы, с помощью которых мы узнаём прогноз погоды и расписание поездов, определяем маршруты путешествий, заказываем билеты на самолёты, бронируем номера в гостиницах и т. п.

Информационная система (ИС) в широком смысле — это аппаратные и программные средства, предназначенные для того, чтобы своевременно обеспечить пользователей нужной информацией.

У информационной системы две основные задачи — она должна обеспечивать:

- хранение данных;
- доступ к данным, т. е. возможность искать и изменять данные.

Массивы данных, с которыми работают информационные системы, обычно имеют большой объём (нередко несколько гигабайтов и даже терабайтов) и размещаются во внешней памяти компьютера. Данные хранятся в таком виде, чтобы их было легко искать и изменять. Такие наборы данных называются базами данных.

База данных (БД) — это специальным образом организованная совокупность[1] данных о некоторой предметной области, хранящаяся во внешней памяти компьютера.

Данные сами по себе бесполезны, если мы не умеем с ними работать. Поэтому необходимо специальное программное обеспечение, которое позволяет искать и изменять данные.

Система управления базой данных (СУБД) — это программные средства, которые позволяют выполнять все необходимые операции с базой данных.

Хотя термины «база данных» и СУБД обозначают различные понятия, они неразрывно связаны: свойства базы данных определяются СУБД, которая ею управляет, и наоборот. Комплекс «БД + СУБД» называется *системой базы данных* (англ. *database system*) или *информационной системой* в узком смысле.

Вы знаете, что данные в компьютерном формате — это двоичные коды, которые могут обозначать всё, что угодно. Поэтому СУБД должна «знать» формат файлов (что где записано). В первых информационных системах каждая база данных имела свой собственный формат, который придумывал её автор. Это очень неудобно, потому что для каждой ИС нужно разрабатывать специальную программу для работы с базой данных. Более того, при любых изменениях в БД (например, при увеличении размера данных) надо переделывать все работающие с ней программы и переводить старые данные в новый формат.

Поэтому постепенно перешли к использованию специальных программ (они и получили название СУБД), которые занимались *только* хранением и обработкой данных, независимо от их содержания, и могли применяться в самых разных задачах. При этом вместе с основными данными нужно хранить описание их структуры — так называемые *метаданные*, «сведения о данных», которые использует СУБД для обращения к данным.

[1] Термин «совокупность» обозначает множество элементов, обладающих общими свойствами.

СУБД решают все задачи, связанные с управлением данными, в том числе:

- поиск данных;
- редактирование данных;
- выполнение несложных расчётов;
- обеспечение *целостности* (корректности, непротиворечивости) данных;
- восстановление данных после сбоев.

Как правило, пользователь работает с СУБД не напрямую, а через прикладную программу, в которой предусмотрен удобный ввод данных и оформление результатов (рис. 3.1).

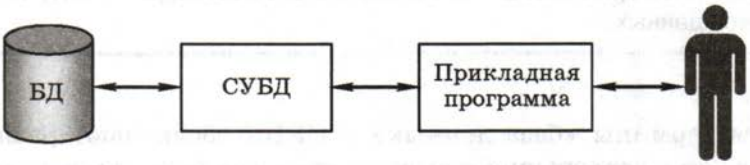


Рис. 3.1

Иногда функции СУБД и прикладной программы объединяются в одной программе (например, в OpenOffice.org Base или Microsoft Access).

Классификация

В простейшем случае база данных и СУБД находятся на одном компьютере. Такая информационная система называется **локальной**, с ней работает один пользователь. В современных браузерах (Google Chrome, Safari, Mozilla Firefox) есть встроенные средства, позволяющие создавать и использовать локальные ИС.

Преимущество локальных ИС — *автономность*, т. е. независимость от работы локальных и глобальных сетей. Их недостатки проявляются тогда, когда с БД должны работать несколько пользователей:

- базу данных нужно обновлять на каждом компьютере;
- невозможно «стыковать» изменения, вносимые пользователями.

В локальных ИС разработчики иногда применяют свои собственные форматы хранения данных, однако в этом случае *теряется переносимость* — возможность использования базы данных в других информационных системах.

Как правило, в современных информационных системах используют **удалённые базы данных**, расположенные на серверах (специально выделенных компьютерах) локальной или глобальной сети. В этом случае несколько пользователей могут одновременно работать с базой и вносить в неё изменения.

СУБД, работающие с удалёнными базами данных, можно разделить на два типа по способу работы с файлами:

- файл-серверные СУБД;
- клиент-серверные СУБД.

Файл-серверные СУБД (например, Microsoft Access) работают на компьютерах пользователей (они называются рабочими станциями) (рис. 3.2). Это значит, что сервер только хранит файлы, но не участвует в обработке данных. Когда пользователь вносит изменения в базу, СУБД с его рабочей станции блокирует файлы на сервере, чтобы их не могли изменить другие пользователи.

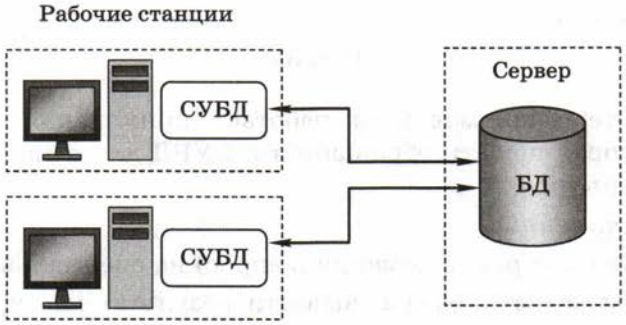


Рис. 3.2

При большом количестве пользователей проявляются недостатки файл-серверных ИС:

- обработку данных выполняют СУБД на рабочих станциях, поэтому компьютеры пользователей должны быть достаточно мощными;
- при поиске данных вся БД копируется по сети на компьютер пользователя, это создает значительную лишнюю нагрузку на сеть;
- слабая защита от неправомерного доступа к данным (защита устанавливается на рабочих станциях, а не в едином центре);
- ненадёжность при большом количестве пользователей, особенно если они вносят изменения в базу данных.

Чтобы избавиться от этих недостатков, нужно переместить СУБД на сервер.

Клиент-серверная СУБД (рис. 3.3) расположена на том же компьютере, где находится база данных. Она полностью берёт на себя всю работу с данными, т. е. читать и изменять данные в базе можно только с помощью этой СУБД.

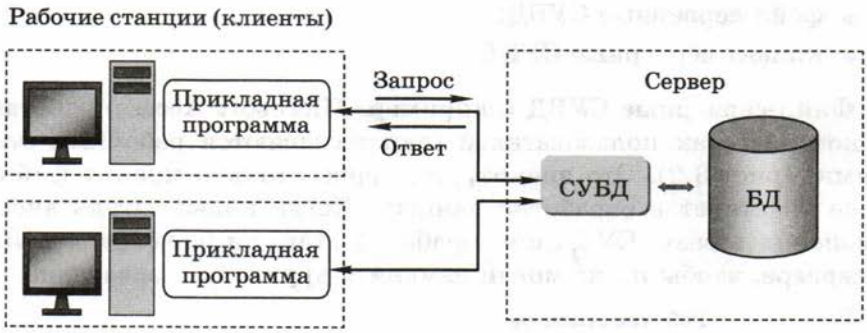


Рис. 3.3

На компьютере пользователя работает прикладная программа-клиент, которая по сети обращается к СУБД для выполнения операций с данными.

Задачи клиента:

- отправить серверу команду (запрос) на специальном языке;
- получив ответ сервера, вывести результат на экран пользователя или на печать.

В современных клиент-серверных СУБД для управления данными чаще всего используют **язык SQL** (англ. *Structured Query Language* — язык структурных запросов). Он содержит все команды, необходимые работы с данными, включая получение нужной информации, создание и изменение базы данных.

Задачи сервера:

- ожидать запросы клиентов по сети;
- при поступлении запроса поставить его в очередь на выполнение;
- выполнить запрос;
- передать результаты клиенту.

Отметим, что серверная и клиентская части могут быть установлены на одном компьютере.

Самые известные среди коммерческих клиент-серверных СУБД — *Microsoft SQL Server* и *Oracle*. Существуют бесплатные клиент-серверные СУБД: *Firebird* (www.firebirdsql.org), *PostgreSQL* (www.postgresql.org), *MySQL* (www.mysql.com). Свободная СУБД MySQL часто используется для управления базами данных на небольших веб-сайтах.

Достоинства клиент-серверных СУБД:

- основная обработка данных выполняется на сервере, поэтому рабочие станции могут быть маломощными;
- проще модернизация (достаточно увеличить мощность сервера);
- надёжная защита данных (устанавливается на сервере);
- снижается нагрузка на сеть, так как передаются только нужные данные (запросы и результаты выполнения запросов);
- надёжная работа при большом количестве пользователей (запросы ставятся в очередь).

Их *недостатки* — повышенные требования к мощности сервера и высокая стоимость коммерческих СУБД (Microsoft SQL Server и Oracle).

Многие современные информационные системы (например, поисковые системы в Интернете) работают с огромными объёмами данных, которые невозможно разместить на одном компьютере. Поэтому появились **распределённые базы данных**, расположенные на множестве компьютеров, и соответствующие СУБД для управления ими. Пользователь работает с распределённой базой данных точно так же, как и с обычной (нераспределённой). Главная проблема в этой области — обеспечение целостности и непротиворечивости данных, особенно при разрыве связи между компьютерами.

Транзакции

Сейчас в базах данных нередко хранится очень важная информация, прежде всего финансовая. Поэтому необходимо обеспечить максимально возможную защиту данных от различных сбоев (например, при отключении питания).

Предположим, что в банке нужно перевести 100 000 рублей со счёта 12345 на счёт 54321. Эта банковская операция включает несколько действий с базой данных:

1) прочитать сумму на счету 12345;
2) уменьшить её на 100 000 рублей и записать результат обратно;
3) прочитать сумму на счету 54321;
4) увеличить её на 100 000 рублей и записать результат обратно.

Представьте себе, что случится, если после выполнения первых двух действий произойдет сбой питания и действия 3 и 4 не будут выполнены. Ответ ясен — первый клиент просто потеряет 100 000 рублей со своего счёта, т. е. данные станут некорректными.

Чтобы этого не произошло, либо все шаги операции перевода денег должны быть выполнены, либо ни один шаг не должен быть выполнен. Кроме того, между отдельными шагами операции никакие другие действия не должны выполняться. Такие сложные многошаговые операции называются транзакциями (от англ. *transaction* — сделка).

Транзакция — это группа операций, которая представляет собой одно законченное действие. Транзакция должна быть выполнена целиком или не выполнена вообще.

Как же обеспечить выполнение транзакций? Для этого часто используется *журналирование* по тому же принципу, что и в файловых системах. Перед внесением изменений в базу данных СУБД создаёт копии всех данных, которые будут изменяться, и записывает в специальный файл (журнал) все операции, которые нужно выполнить. Затем эти операции выполняются фактически и, если всё завершено успешно, запись удаляется из журнала. Если произошёл сбой, в журнале будет найдена информация о тех операциях, которые уже были завершены, и база данных восстанавливается в исходное состояние (транзакция не выполнена).

Некоторые СУБД, например Firebird, не используют журнал, а при внесении изменений создают в базе данных новые записи, которые отмечаются как «рабочие» только тогда, когда транзакция завершена.

Вопросы и задания

1. Что такое информационная система? Из каких компонентов она состоит?
2. Что такое база данных? Какими свойствами она должна обладать?

3. Является ли базой данных бумажная картотека в библиотеке? Ответ обоснуйте.
4. Какие функции выполняет СУБД?
5. Почему произошёл переход от множества разнообразных форматов хранения данных к использованию универсальных СУБД? Приведите примеры и обоснуйте.
6. Что такое метаданные?
7. Объясните схему работы пользователя с базой данных.
8. Назовите достоинства и недостатки локальных ИС.
9. Назовите достоинства и недостатки файл-серверных СУБД.
10. В каких ситуациях вы могли бы рекомендовать использование файл-серверных СУБД? Ответ обоснуйте. Подготовьте сообщение.
11. Назовите достоинства и недостатки клиент-серверных СУБД.
12. Как разделяются функции между клиентской и серверной программами?
13. Что такое SQL?
14. Что такое распределённые базы данных?
15. Что такое транзакция?
16. Как обеспечивается защита данных в случае сбоев при использовании механизма транзакций?

Подготовьте сообщение

а) «Информационные системы вокруг нас»

б) «Технология клиент — сервер»

в) «Бесплатные СУБД»

г) «Коммерческие и бесплатные СУБД: плюсы и минусы»

§ 13 Таблицы

Основные понятия

Данные, хранящиеся в современных базах данных, чаще всего удобно представлять в виде таблиц[1]. Например, так называемый «список контактов» (сведения о друзьях и знакомых) может выглядеть, как показано на рис. 3.4.

[1] В середине XX века широко применялись иерархические и сетевые базы данных, но сейчас они редко встречаются на практике. Самый известный современный пример иерархической базы данных — реестр в операционной системе Windows, где хранятся настройки самой системы и программ.

Фамилия	Имя	Адрес	Телефон
Иванов	Петр	Суворовский пр., д. 32, кв. 11	275-75-75
Петров	Василий	Кутузовский пр., д. 12, кв. 20	276-76-76
Васильев	Иван	Нахимовский пр., д. 23, кв. 33	277-77-77

Рис. 3.4

Столбцы таблицы называются **полями**, а строки — **записями**. Таблица на рис. 3.4 относится к типу «объект — свойства», т. е. запись — это описание некоторого объекта (в данном случае — человека), а поля содержат свойства этого объекта. В этой таблице четыре поля: *Фамилия, Имя, Адрес* и *Телефон* и три записи.

Количество и состав полей определяет разработчик базы данных, пользователи могут только изменять записи (добавлять, удалять, редактировать).

Любое поле должно иметь уникальное (неповторяющееся) имя. Например, нельзя назвать два поля *Фамилия*, но можно одно назвать *Фамилия*, а второе — *Девичья Фамилия*.

Каждое поле имеет свой **тип**. Как правило, СУБД поддерживают следующие типы данных:

- целые числа;
- вещественные числа;
- денежные суммы;
- логические значения (битовые поля);
- текстовые данные;
- время, дата;
- произвольные двоичные данные, например закодированный звук, видео и т. д.

Некоторые поля могут быть обязательными для заполнения. Если обязательное поле не заполнено, СУБД не внесёт изменения в базу данных и выдаст сообщение об ошибке.

Ключ

При чтении и изменении данных в таблице очень важно убедиться, что мы обращаемся именно к нужной записи. Это означает, что для надёжной работы каждая запись должна содержать какое-то уникальное значение, отличающее её от всех остальных.

Ключ — это поле или комбинация полей, однозначно определяющая запись.

Это значит, что ключ обладает свойством *уникальности*: в таблице не может быть двух записей, у которых одинаковое значение ключа. Например, ключом может быть номер паспорта, номер мобильного телефона, регистрационный номер автомобиля, адрес электронной почты и т. п.

Иногда можно выделить в таблице несколько ключей, например номер внутреннего паспорта и номер заграничного паспорта. В этом случае один из них выбирается в качестве основного и называется **первичным ключом**.

Ключ, состоящий из одного поля, называется **простым**, а соответствующее поле таблицы — **ключевым полем**.

Ключ, который состоит из нескольких полей, называется **составным**. Представим себе базу данных метеостанции, на которой через каждые 3 часа измеряются температура, влажность воздуха, скорость ветра и т. п. (рис. 3.5).

Дата	Время	Температура	Влажность	Скорость ветра
21.07.2012	12	25	75	4
21.07.2012	15	23	70	3
...	...	...	...	...

Рис. 3.5

Здесь ни одно поле не может быть ключом, потому что значения в каждом из них могут повторяться. Однако для каждой пары *Дата* + *Время* в таблице может быть только одна запись, поэтому комбинация полей *Дата* и *Время* — это ключ.

Второе свойство ключа — *несократимость*. Заметим, что в рассмотренном примере к паре *Дата* + *Время* можно добавить и другие поля таблицы, но такая группа полей уже не будет считаться ключом, потому что она сократима, т. е. из неё можно исключить все поля, кроме полей *Дата* и *Время*, сохранив свойство уникальности.

Теперь посмотрим на приведённую на рис. 3.5 таблицу «список контактов». Ни фамилия, ни имя не могут быть ключом, потому что есть много однофамильцев и людей с одинаковыми именами. Составить ключ из полей *Фамилия* и *Имя* тоже не получит-

ся (могут быть однофамильцы и тёзки одновременно). Адрес и домашний телефон также не могут быть ключом, потому что в одной квартире могут жить несколько человек, с которыми вы общаетесь. В принципе может получиться так, что ключом будет комбинация *всех* полей записи.

Работать с составными ключами при выполнении операций с базой данных на практике очень неудобно. В таких случаях часто добавляют в таблицу ещё одно поле — так называемый *суррогатный* (т. е. неестественный) ключ, например номер записи. Во многих СУБД есть возможность заполнять его автоматически при добавлении каждой новой записи (рис. 3.6). При этом пользователю не нужно задумываться об уникальности такого ключа.

Номер	Фамилия	Имя	Адрес	Телефон
1	Иванов	Петр	Суворовский пр., д. 32, кв. 11	275-75-75
2	Петров	Василий	Кутузовский пр., д. 12, кв. 20	476-76-76
3	Васильев	Иван	Нахимовский пр., д. 23, кв. 33	477-77-77

Рис. 3.6

Индексы

Представим себе, что в таблице с адресами и телефонами, о которой мы говорили выше, записаны данные большого количества людей (в реальных базах данных могут быть миллионы записей). Как найти всех Ивановых?

Учтём, что записи расположены в таблице в том порядке, в котором они вводились (возможно, без всякого порядка). Если эту задачу решает человек, он, скорее всего, будет просматривать поле *Фамилия* с первой до последней записи, отмечая всех с фамилией Иванов. Такой поиск называется **линейным** — он требует просмотра всех записей (вспомните линейный поиск элемента в массиве). Если в таблице 1024 записи, придётся сделать 1024 сравнения, в больших базах линейный поиск будет работать очень медленно.

Теперь представим себе, что записи **отсортированы** по фамилии **в алфавитном порядке**. Возьмём запись, стоящую в середине таблицы (если в таблице 1024 записи, возьмём 512-ю) и сравним фамилию с нужной. Допустим, в 512-й записи фамилия — Коню-

хов. Поскольку нам нужны Ивановы, все они заведомо находятся в верхней половине таблицы, поэтому всю вторую половину можно вообще не смотреть. Таким образом, мы в два раза уменьшили область поиска. Далее проверяем среднюю из оставшихся записей (256-ю) и повторяем процесс до тех пор, пока не найдём Иванова или в области поиска не останется больше записей. Как вы знаете, такой **поиск** называется **двоичным** (вспомните материал учебника 10 класса). Он позволяет искать очень быстро: если в базе N записей, то придется сделать всего около $\log_2 N$ сравнений. Например, для $N = 1024$ получаем 11 сравнений вместо 1024, т. е. мы смогли ускорить поиск почти в 100 раз! Для бóльших N ускорение будет еще более значительным.

Однако у двоичного поиска есть и недостатки:

- записи должны быть предварительно отсортированы по нужному полю;
- можно искать только по одному полю.

На практике искать нужно по нескольким полям в каждой таблице и нет возможности каждый раз сортировать записи (это очень долго для больших баз). Как же в такой ситуации обеспечить быстрый поиск? Как часто бывает в программировании, можно увеличить скорость работы алгоритма за счёт расхода дополнительной памяти.

Наверное, вы видели, что для ускорения поиска во многие книги включают *индекс* — список ключевых слов с указанием страниц, где они встречаются. В базах данных специально для поиска создаются дополнительные таблицы, которые также называются индексами.

Индекс — это вспомогательная таблица, которая служит для ускорения поиска в основной таблице.

Простейший индекс — это таблица, в которой хранится значение интересующего нас поля основной таблицы (например, *Фамилия*) и список номеров записей, где такое значение встречается. Записи в индексе упорядочены (отсортированы) по нужному полю, т. е. для приведённой на рис. 3.6 таблицы индекс по полю *Фамилия* выглядит так, как на рис. 3.7.

Фамилия	№
Васильев	3
Иванов	1
Петров	2

Рис. 3.7

Теперь, если нам нужны люди с фамилией *Иванов*, мы можем искать номера Ивановых в индексе, используя быстрый *двоичный* поиск (там фамилии стоят по алфавиту!). Затем, когда номера нужных записей определены, данные Ивановых можно взять из основной таблицы. Заметим, что искать что-то в основной таблице уже не нужно. Таким образом, можно использовать двоичный поиск по разным полям (даже по комбинациям нескольких полей). Однако нужно учитывать, что:

- для каждого типа поиска приходится строить новый индекс, он будет занимать дополнительное место во внешней памяти (например, на жёстком диске);
- при любом изменении таблицы (добавлении, изменении и удалении записей) необходимо перестраивать индексы так, чтобы сохранить требуемый порядок сортировки; к счастью, современные СУБД делают это автоматически без участия пользователя, но это может занимать достаточно много времени.

Для тренировки самостоятельно постройте вручную индексы по полям *Имя*, *Адрес* и *Телефон*.

Целостность базы данных

Целостность базы данных означает, что БД содержит полную и непротиворечивую информацию и удовлетворяет всем заданным ограничениям.

Прежде всего нужно обеспечить *физическую целостность БД*, т. е. защитить данные от разрушения в случае отказа оборудования (например, при отключении питания или выходе из строя жёстких дисков).

Очень важно, что все изменения данных выполняются с помощью *транзакций*, которые позволяют в случае сбоя «откатить назад» все начатые операции.

Периодически (например, раз в неделю) администраторы создают резервные копии всех данных на дисках и ведут журнал изменений. При сбое восстанавливается самая последняя сохранённая версия БД.

Также используют так называемые RAID-массивы жёстких дисков (англ. *Redundant Array of Independent Disks* — избыточный массив независимых дисков), где информация дублируется

и может быть автоматически восстановлена в случае выхода из строя одного или даже нескольких дисков.

Теперь представьте себе, что в базе данных отдела кадров по ошибке у работника указан 1698 год рождения, а в поле *Зарплата* введено отрицательное число. В этих случаях нарушается *логическая целостность*, т. е. непротиворечивость данных. Чтобы этого не произошло, вводят ограничения на допустимые значения полей (контроль данных):

- каждое поле имеет свой тип; например, СУБД не даст записать в поле целого типа произвольный текст — будет выведено сообщение об ошибке;
- некоторые поля (в первую очередь первичный ключ) объявляются обязательными для заполнения;
- для полей, значения которых не могут повторяться, строятся уникальные индексы (при повторении значения выдаётся сообщение об ошибке);
- вводятся условия, которые должны выполняться для значений отдельных полей: например, можно потребовать, чтобы значение поля, в котором хранится количество учеников в классе, было положительным;
- для сложных данных используются шаблоны ввода: например, для ввода семизначного номера телефона можно использовать шаблон ###-##-##, где # означает любую цифру;
- вводятся условия, которые должны выполняться для нескольких полей каждой записи: например, дата увольнения работника не может быть более ранней, чем дата приёма на работу.

Заметим, что целостность БД не гарантирует *достоверность* данных, а только означает, что выполнены все установленные ограничения на эти данные и таким образом исключены явные противоречия.

Вопросы и задания

1. Объясните значения слов «поле», «запись».
2. Зачем каждому полю присваивают свой тип?
3. Какие типы данных поддерживаются в современных СУБД?
4. Что такое ключ таблицы? Назовите и объясните два свойства ключа.
5. Чем отличаются простой и составной ключи?

6. Чем отличаются понятия «ключ» и «первичный ключ»?
7. Какие из следующих данных могут быть ключом, а какие не могут:
   а) фамилия;
   б) имя;
   в) номер читательского билета;
   г) адрес электронной почты;
   д) адрес веб-сайта;
   е) марка автомобиля?
8. Объясните, когда одни и те же данные в одной ситуации могут быть ключом, а в другой — нет (например, адрес электронной почты, марка стиральной машины и т. п.). Приведите примеры.
9. В каких случаях в качестве первичного ключа используют номер записи? Можно ли применять такой подход, если в таблице есть другое уникальное поле?
10. Какие методы поиска данных вы знаете?
11. Чем различаются линейный и двоичный поиск? Назовите их достоинства и недостатки.
12. Что такое индекс? Как он строится?
13. Можно ли для одной и той же таблицы построить несколько индексов?
14. Объясните принцип поиска с помощью индекса.
15. Что такое целостность базы данных? Какие виды целостности вы знаете?
16. Как обеспечивается физическая целостность данных?
17. Как обеспечивается логическая целостность данных?

Подготовьте сообщение

а) «Типы данных, хранящиеся в БД»
б) «Суррогатные ключи: за и против»
в) «Поиск с помощью индексов»
г) «Что такое транзакция?»
д) «Что такое RAID-массив?»

Задачи

1. В базе данных хранится $1\,048\,576 = 2^{20}$ записей. Оцените количество сравнений, которое придётся сделать при использовании линейного и двоичного поиска по одному из полей. Во сколько раз быстрее работает двоичный поиск?
2. В таблице три поля: *Дата*, *Номер заказа*, *Товар* и *Количество*. Что можно выбрать в качестве первичного ключа? Какие индексы можно построить?
3. В школьной базе данных хранятся сведения о выданных аттестатах. Таблица включает поля *Фамилия*, *Имя*, *Отчество*, *Дата рождения*, *Год выпуска*, *Номер паспорта*, *Номер аттестата*. Что можно выбрать в качестве первичного ключа этой таблицы?

4. Постройте индексы по полям *Дата*, *Товар* и *Количество* для следующей таблицы:

Номер	Дата	Заказ	Товар	Количество, т
1	12.09.10	12	Ананасы	12
2	12.09.10	13	Апельсины	12
3	13.09.10	14	Ананасы	15
4	13.09.10	15	Бананы	13
5	13.09.10	16	Апельсины	11

*5. Напишите программу, работающую с однотабличной базой данных. Содержание придумайте сами (например, видеотека, база данных зоопарка или что-либо другое). В программе должны быть предусмотрены:

- просмотр записей;
- добавление записей (в конец таблицы);
- удаление записей (по номеру);
- сортировка по одному полю.

Для хранения записей используйте текстовый файл.

§ 14 Многотабличные базы данных

Почему не собрать всё в одной таблице?

Мы только что рассмотрели простейшую структуру, в которой все данные сведены в одну таблицу и поэтому искать информацию достаточно просто. Однако у такой модели есть и *недостатки*:

- *дублирование данных*; например, в базе данных школьной библиотеки будет много раз храниться фамилия автора «Пушкин»;
- при изменении каких-то данных (например, адреса фирмы), возможно, придётся *менять несколько записей*;
- нет *защиты от ошибок ввода* (опечаток).

Однотабличная база данных — это аналог картотеки, в которой все карточки имеют одинаковую структуру. В то же время на практике в одной базе нужно хранить данные, относящиеся к объектам разных типов, которые связаны между собой. Поэтому

возникает вопрос: какую модель лучше использовать для описания и хранения этих данных?

Посмотрим, как можно организовать базу данных, в которой хранятся данные об альбомах музыкальных групп. Вот что получается, если свести все данные в одну таблицу (рис. 3.8).

Альбомы

Код альбома	Название	Группа	Год	Число композиций
1	Реки и мосты	Машина времени	1987	16
2	В круге света	Машина времени	1988	11
3	Группа крови	Кино	1988	11
4	Последний герой	Кино	1989	10

Рис. 3.8

В данном случае в качестве первичного ключа можно выбрать пару свойств *Название* + *Группа*, но работать с таким составным ключом неудобно, поэтому мы ввели суррогатный ключ — дополнительное поле *Код альбома* (целое число), оно обозначено знаком 🔑.

Сразу видим, что в этой таблице есть дублирование — название группы (символьная строка) повторяется для каждого альбома этой группы. Причина в том, что в данных таблицы на самом деле есть сведения не только об альбомах, но и о группах — объектах совершенно другого класса. Поэтому для хранения всей информации о группах нужно сделать отдельную таблицу (рис. 3.9).

Группы

Код группы	Название	Год создания
1	Машина времени	1969
2	Кино	1981

Рис. 3.9

Здесь первичным ключом может быть название группы (символьная строка), но для ускорения работы введён суррогатный ключ — *Код группы* (целое число). В таблицу можно добавить другие данные о группе, например фамилию лидера, город и т. п.

В таблице *Альбомы* теперь будут храниться не названия групп, а их коды (рис. 3.10).

Альбомы

Код альбома	Название	Код группы	Год	Число композиций
1	Реки и мосты	1	1987	16
2	В круге света	1	1988	11
3	Группа крови	2	1988	11
4	Последний герой	2	1989	10

Рис. 3.10

Таким образом, база данных состоит из двух таблиц и хранит сведения о двух классах объектов. Эти таблицы связаны, их связь можно показать на схеме (рис. 3.11).

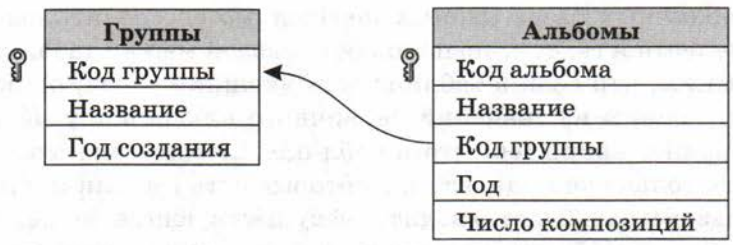


Рис. 3.11

На рисунке 3.11 перечислены поля каждой из таблиц и показана их связь: код группы в таблице *Альбомы* должен совпадать с кодом одной из групп в таблице *Группы*.

Внешний ключ (англ. *foreign key*) — это неключевое поле таблицы, связанное с первичным ключом другой таблицы.

Таким образом, *Код группы* — это внешний ключ в таблице *Альбомы*.

Итак, мы получили две связанные таблицы вместо одной. При этом:

- устранена избыточность (повторно хранятся только числовые коды);
- все изменения нужно выполнять только в одном месте;

- есть некоторая защита от ошибок при вводе данных — можно сделать так, что при заполнении таблицы *Альбомы* название группы будет выбираться из уже готового списка групп.

Однако кое-что *ухудшилось* из-за того, что данные разбросаны по разным таблицам:

- базами данных, в которых более 40–50 таблиц, сложно управлять из-за того, что разработчику трудно воспринимать информацию в таком «раздробленном» виде;
- при поиске приходится «собирать» нужные данные из нескольких таблиц.

Ссылочная целостность

Одна из важнейших задач СУБД — поддерживать *целостность базы данных*. В предыдущем параграфе мы уже говорили о том, как обеспечивается *физическая и логическая целостность*. В многотабличных базах данных необходимо обеспечить еще **ссылочную целостность**, т. е. правильность связей между таблицами.

Напомним, что если в таблице есть внешний ключ, он должен совпадать с одним из значений первичного ключа в другой таблице. Это значит, например, что в таблице *Альбомы* мы можем использовать только те коды групп, которые есть в таблице *Группы*. Если пользователь будет вводить несуществующие коды, СУБД должна выдать сообщение об ошибке и отказаться вносить изменения в базу данных.

При изменении кода группы (первичного ключа) нужно изменить соответствующие коды для всех альбомов этой группы.

Если пользователь удаляет какую-то группу из таблицы *Группы*, то для всех альбомов этой группы значение поля Код группы оказывается недействительным и целостность нарушается. В таких случаях возможно несколько вариантов действия СУБД:

- *запретить* удаление записи (из таблицы Группы) до тех пор, пока в базе есть связанные с ней подчинённые записи (в таблице *Альбомы*);
- выполнить *каскадное удаление*, т. е. вместе с удаляемой записью удалить также все связанные с ней подчинённые записи в других таблицах;
- *игнорировать* проблему, т. е. разрешить внести изменения.

Очевидно, что в последнем случае ссылочная целостность нарушается.

Типы связей

В базе данных, которую мы только что построили, использовались связь **«один ко многим»** (она обозначается также как «1:*N*» или «1–∞»). Это значит, что с одной записью в первой таблице могут быть связаны сколько угодно записей во второй таблице. Так в нашем случае каждый код группы может встречаться сколько угодно раз в таблице *Альбомы* (рис. 3.12).

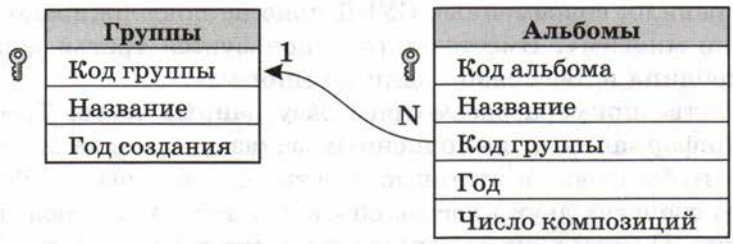


Рис. 3.12

При этом, как правило, на стороне «1» в связи участвует ключевое поле таблицы, а на стороне «N» — неключевое (для второй таблицы это внешний ключ).

В некоторых (достаточно редких) случаях используют связь **«один к одному»** (или «1:1»): каждой записи в первой таблице соответствует ровно одна запись в связанной таблице (рис. 3.13).

Сотрудники

Код	Фамилия	Имя	Отчество
1	Иванов	Петр	Сидорович
2	Петров	Сидор	Иванович
3	Сидоров	Иван	Петрович

Секретно

Код	Зарплата
1	20 000 р.
2	30 000 р.
3	40 000 р.

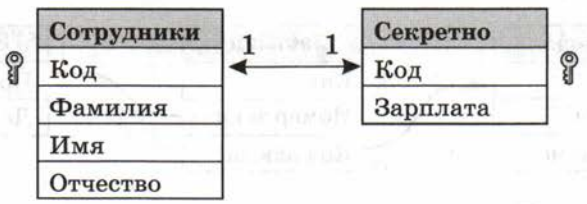


Рис. 3.13

В этом случае таблицы связаны через свои ключевые поля. Такую связь можно использовать для разделения большой таблицы на две части. Это важно, например, когда часть свойств объек-

та нужно сделать доступной всем, а другую часть — скрыть от некоторых пользователей.

Нередко при анализе исходных данных появляются связи типа «многие ко многим» (они обозначаются также «*M*:*N*» или «∞–∞»). Например, в школе каждый учитель может преподавать несколько разных предметов и каждый предмет обычно ведут несколько учителей. Поэтому связь между учителями и предметами — это связь «многие ко многим».

Как правило, современные СУБД явно не поддерживают связи «многие ко многим». Вместо этого используется третья дополнительная таблица и две связи «один ко многим».

В качестве примера рассмотрим базу данных кафе. Требуется хранить информацию о выполненных заказах, их составе и цене для того, чтобы строить итоговые отчёты автоматически. Все данные — это свойства двух классов объектов: заказов и блюд, входящих в меню. Поэтому будем использовать две таблицы (рис. 3.14).

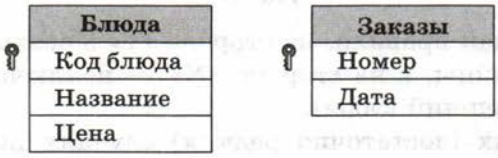


Рис. 3.14

Как они связаны? Очевидно, что каждое блюдо может быть включено во многие заказы. Вместе с тем каждый заказ может состоять из нескольких блюд. Таким образом, здесь существует связь «многие ко многим», которая в большинстве СУБД не поддерживается. Чтобы решить проблему, введём еще одну таблицу *Заказано*, которая связана с первыми двумя с помощью двух связей «1:*N*» (рис. 3.15).

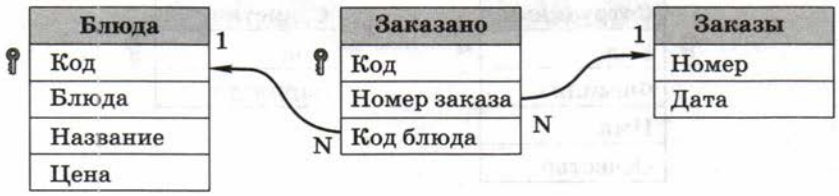


Рис. 3.15

Вот пример заполнения таблиц в такой базе данных (рис. 3.16).

Заказы

Номер	Дата
1	11.12.12
2	12.12.12

Заказано

Код	Номер заказа	Код блюда
1	1	1
2	1	3
3	1	4
4	2	1
5	2	2
6	2	2
7	2	5

Блюда

Код	Название	Цена
1	Борщ	80 руб.
2	Бифштекс	110 руб.
3	Гуляш	70 руб.
4	Чай	10 руб.
5	Кофе	50 руб.

Рис. 3.16

Как видно из этого примера, в состав заказа № 1 вошли борщ, гуляш и чай, а в заказ № 2 — борщ, два бифштекса и кофе. Обратите внимание, что в таблице *Заказано* было необходимо ввести дополнительное ключевое поле *Код* (суррогатный ключ), потому что в одном заказе может быть несколько одинаковых блюд.

Вопросы и задания

1. Почему собирать все данные в одной таблице во многих случаях невыгодно?
2. По какому принципу данные разбиваются на несколько таблиц?
3. Что такое внешний ключ таблицы?
4. Что такое ссылочная целостность БД? Как она обеспечивается?
5. Какие типы связей используются в многотабличных базах данных?
6. Когда применяется связь «1:1»? Какие поля при этом связываются?
7. Когда применяется связь «1:*N*»? Какие поля при этом связываются?
8. Таблицы *А* и *Б* связаны через свои ключевые поля. Что это за связь?
9. Связь между таблицами *А* и *Б* установлена через ключевое поле таблицы *Б* и неключевое поле таблицы *А*. Что это за связь?
10. Между таблицами *А* и *Б* по смыслу существует связь «многие ко многим». Как добиться этого в СУБД, которая такую связь не поддерживает?
11. Подумайте, какие изменения можно внести в базу данных, описанную в конце параграфа, если клиенты часто заказывают несколько одинаковых блюд.

Задачи

1. Фирма ведёт базу данных заказчиков, состоящую из двух связанных таблиц:

Заказчики

Код	Название	Код города
1	Иванов Т.М.	3
2	Пановко И.Т.	2
3	Черненко И.А.	3
4	Пановко А.И.	2
5	Иванова А.И.	1

Города

Код	Название
1	Москва
2	Санкт-Петербург
3	Пермь
4	Воронеж
5	Липецк

Сколько заказчиков располагается в Перми?

2. Для учёта отгруженных товаров к базе данных из предыдущего задания добавили ещё две таблицы:

Заказы

Накладная	Код заказчика	Артикул	Количество упаковок
1011	3	7576	10
1012	5	7576	20
1013	4	3889	25
1014	1	7825	30
1015	3	7576	10

Товары

Артикул	Название	Цена за упаковку
7576	Бумага	150 руб.
2325	Карандаши	200 руб.
3889	Фломастеры	350 руб.
2987	Дневники	400 руб.
7825	Пеналы	250 руб.

Определите:

а) Какие товары отправлены в каждый из городов?
б) Сколько бумаги отправлено в каждый из городов?
в) Какова общая стоимость товаров, отправленных в каждый из городов?

3. Во фрагменте базы данных представлены сведения о родственных отношениях. Определите на основании приведённых данных фамилию и инициалы:

Персоны

Код	ФИО	Пол
71	Иванов Т.М.	М
85	Пановко И.Т.	М
13	Черненко И.А.	Ж
42	Пановко А.И.	Ж
23	Иванова А.И.	Ж
96	Пановко Н.Н.	Ж
82	Черненко А.Н.	М
95	Фукс Т.Н.	Ж
10	Фукс Н.А.	М
...	...	...

Дети

Код родителя	Код ребенка
23	71
13	23
85	23
82	13
95	13
85	42
82	10
95	10
...	...

а) бабушки А. И. Ивановой;
б) родного брата И. А. Черненко;
в) прадеда Т. М. Иванова;
г) внука И. Т. Пановко.

4. Во фрагменте базы данных представлены сведения о родственных отношениях. Определите на основании приведённых данных фамилию и инициалы:

Персоны

Код	ФИО	Пол
86	Сизых И.Т.	М
83	Сизых А.И.	М
50	Малых А.Т.	Ж
79	Сидоров Т.М.	М
23	Сидоров А.Т.	М
13	Малых И.И.	Ж
98	Симоняк Т.Н.	Ж
11	Симоняк Н.И.	М
...	...	...

Дети

Код родителя	Код ребенка
98	83
86	13
79	50
86	83
13	50
79	23
13	23
98	13
86	11
...	...

а) племянника Н. И. Симоняка (*примечание*: племянник — сын сестры или брата);
б) всех родных братьев и сестёр Н. И. Симоняка;
в) бабушки А. Т. Малых;
г) дедушки А. Т. Сидорова.

5. Во фрагменте базы данных представлены сведения о родственных отношениях. Определите на основании приведённых данных фамилию и инициалы:

Персоны

Код	ФИО	Пол
11	Косарева Л.П.	Ж
12	Левитин И.А.	М
24	Шумахер А.Ф.	Ж
45	Бланш А.А.	М
56	Васильева М.А.	Ж
83	Левитин Б.И.	М
94	Левитина В.И.	Ж
115	Кузнецов А.П.	М
140	Левитина Р.Б.	Ж
162	Левитин Л.Б.	М
171	Гайдарова З.Н.	Ж
186	Мурина С.А.	Ж
201	Кузнецов П.А.	М
...	...	...

Дети

Код родителя	Код ребенка
11	83
11	94
12	83
12	94
24	115
56	140
56	162
83	140
83	162
94	186
94	201
115	186
115	201
...	...

а) всех внуков и внучек И. А. Левитина;
б) родной сестры А. П. Кузнецова;
в) родного брата С. А. Муриной;
г) бабушки Р. Б. Левитиной.

6. Рыболов решил хранить сведения о своей добыче в базе данных. Он хочет сохранять следующую информацию о каждой рыбалке: дату, место, погоду, вес пойманной рыбы. Помогите рыболову грамотно построить многотабличную базу данных.
7. Рыболов из задачи 6 решил ещё запоминать, сколько каких рыб он поймал на каждой рыбалке. Как нужно изменить базу данных?
8. Строительной фирме нужно хранить в базе данных информацию о составе бригад рабочих (во главе с бригадиром) и о том, какая бригада какие заказы выполняла. Помогите грамотно построить многотабличную базу данных.
9. Альпинисты хотят сохранить в базе данных информацию о горных вершинах и о том, кто из них на какую вершину поднимался и в каком году. Помогите им грамотно построить многотабличную базу данных.
10. Вася получил задание составить базу данных, в которой хранится школьное расписание. Нужно учесть, что каждый предмет преподают несколько учителей, и каждый учитель может вести занятия по нескольким предметам. Помогите Васе грамотно построить многотабличную базу данных.

§ 15
Реляционная модель данных

Математическое описание базы данных

В середине XX века программное обеспечение для работы с базами данных было жёстко «привязано» к внутреннему представлению данных во внешней памяти компьютера, т. е. к структуре файлов с данными. Это означало, что при изменении формата файлов нужно было переделывать все работающие с ними программы. Поэтому возникли следующие задачи:

- разработать строгое математическое описание баз данных, независимое от способа хранения данных;
- разработать методы управления этими данными (поиска, изменения, добавления и т. п.), основанные на использовании математических операций.

Эти задачи удалось решить в 1970 г. англичанину Эдгару Кодду, который работал в фирме IBM. Он предложил новую модель данных, основанную на следующих идеях:

- все данные представляют собой свойства некоторых объектов;
- объекты делятся на классы (в теории баз данных они называются *сущностями*). Например, при описании данных о музыкальных группах можно использовать классы *Группа*, *Альбом*, *Песня* и т. п.;
- данные о некотором объекте — это набор свойств (*атрибутов*), в котором каждое свойство задаётся в виде пары «название — значение»; например, сведения о музыкальной группе «Кино» можно записать так:

(Название: «Кино», Лидер: «В. Цой», Год создания: 1981).

Такой набор данных, описывающий свойства одного объекта, называется *кортежем*.

- Порядок перечисления свойств в кортеже не имеет значения.
- Все объекты одного класса имеют одинаковый набор свойств.

- Множество кортежей, описывающих объекты одного класса, называется *отношением* (англ. *relation*); например, отношение *Группы* можно записать в виде множества кортежей:

  (Название: «Машина времени», Лидер: «А. Макаревич», Год создания: 1969)
  (Название: «Кино», Лидер: «В. Цой», Год создания: 1981)
  (Название: «Аквариум», Лидер: «Б. Гребенщиков», Год создания: 1972)
  ...

- В отношении нет двух одинаковых кортежей.
- Порядок кортежей в отношении не определён.

«Кортеж» и «отношение» — это математические понятия, которые описывают связанные данные. Поэтому с ними можно работать, используя известные операции теории множеств и математической логики. Э. Кодд предложил набор операций с данными, представленными в виде отношений, который служит основой для работы большинства современных СУБД. Модель данных, введённая Коддом, получила название **реляционной модели данных** (от англ. *relation* — отношение).

Реляционные базы данных

Реляционная база данных — это база данных, которая основана на реляционной модели, т. е. представляет собой набор отношений.

Математическая теория Кодда никак не связана с тем, как именно хранятся данные. Однако несложно понять, что отношение удобно представлять в виде прямоугольной таблицы (рис. 3.17).

Группы

Название	Лидер	Год создания
Машина времени	А. Макаревич	1969
Кино	В. Цой	1981
Аквариум	Б. Гребенщиков	1972

Рис. 3.17

Эта таблица описывает отношение *Группы*. Здесь кортежи представлены в виде **записей** (строк), а атрибуты — это **поля** (столбцы) в таблице.

Идеи реляционной теории Кодда легко перевести на «табличный язык»:

- каждая таблица описывает один класс объектов;
- порядок расположения полей в таблице не имеет значения;
- все значения одного поля относятся к одному и тому же типу данных;
- в таблице нет двух одинаковых записей;
- порядок записей в таблице не определён.

Поэтому на практике часто используют ещё одно, более простое определение:

Реляционная база данных — это база данных, которую можно представить в виде набора таблиц.

Однако не любой набор таблиц можно считать реляционной базой данных. Как мы уже говорили, БД и СУБД неразрывно связаны, и для того чтобы отнести систему базы данных (БД + СУБД) к определённому типу, необходимо выяснить, какие методы управления данными используются в соответствующей СУБД.

Согласно реляционной теории, порядок перечисления свойств в кортеже (порядок столбцов в таблице) не определён, так же как и порядок кортежей в отношении (порядок строк в таблице). Поэтому методы работы с данными в реляционной БД не должны предполагать, что столбцы и строки таблиц расположены в каком-то порядке.

Для управления данными в большинстве современных информационных систем используется **язык SQL**, в который включены команды для:

- создания новых таблиц;
- добавления новых записей;
- изменения записей;
- удаления записей;
- выборки записей из одной или нескольких таблиц в соответствии с заданным условием и некоторые другие.

Команды языка SQL позволяют управлять данными, не «привязываясь» к формату их хранения, т. е. к порядку расположения столбцов и строк в таблицах. Для выполнения операций (выборки, вставки, удаления, изменения) используются только названия столбцов и таблиц. С помощью команд SQL можно выполнить все основные операции, введённые Коддом, поэтому **СУБД** (и соответствующие системы баз данных), которые используют язык SQL, традиционно называют **реляционными**[1].

Нормализация

Представим себе, что все данные о рейсах авиакомпании «ZX-Аэро» собраны в одной таблице:

Рейс	От	До	Самолёт	Дата
ZX 001	Москва	Берлин	Boeing 737	11.12.2012
ZX 002	Москва	Санкт-Петербург	Airbus A321	12.12. 2012
ZX 003	Санкт-Петербург	Берлин	Boeing 737	13.12. 2012

Мы сразу видим, что в таблице есть *избыточность* (дублирование): многие данные (названия городов и типов самолётов) хранятся несколько раз. При этом для хранения данных-дубликатов расходуется дополнительное место на диске, что может привести к его преждевременному заполнению и выходу из строя всей информационной системы.

Есть и другая проблема: при вводе всех данных вручную оператор может сделать опечатку и, например, вместо «Санкт-Петербург» ввести «СанктПетербург». В этом случае нарушается *целостность* базы данных, потому что в ней хранится название несуществующего города.

Чтобы избежать этих проблем, при проектировании базы данных обычно выполняют её *нормализацию*.

---

[1] В то же время язык SQL нередко критикуют за то, что он не полностью соответствует реляционной модели данных. Например, в SQL используется понятие «таблица» вместо «отношение»; порядок расположения столбцов таблицы задаётся при её создании; нет запрета на создание одинаковых строк.

Нормализация — это изменение структуры базы данных, которое устраняет избыточность и предотвращает возможные нарушения целостности.

В теории реляционных баз данных существует несколько уровней нормализации (так называемых нормальных форм). Мы покажем некоторые принципы нормализации на примерах.

1. **Любое поле должно быть неделимым.** Это значит, что таблицу вида, показанного на рис. 3.18, необходимо переделывать.

Сотрудник	Телефоны
Иванов Пётр Сидорович	123-45-67, (901) 111-22-33
Петров Сидор Иванович	345-67-89, (902) 222-33-44

Рис. 3.18

Здесь поле *Сотрудник* нужно разделить на три: *Фамилия*, *Имя* и *Отчество*. Поле *Телефоны* содержит два телефона: домашний и мобильный. Поэтому нужно изменить таблицу, по крайней мере, так (рис. 3.19).

Фамилия	Имя	Отчество	Телефон-Дом	Телефон-Моб
Иванов	Петр	Сидорович	123-45-67	(901) 111-22-33
Петров	Сидор	Иванович	345-67-89	(902) 222-33-44

Рис. 3.19

2. **Любое неключевое поле должно зависеть от ключа таблицы.** Например, в таблице на рис. 3.20 ключ — это регистрационный номер автомобиля.

Номер	Автомобиль	Владелец	Телефон
A123AA47	«Лада-Калина»	Иванов	155-77-23
T234TT78	«Ока»	Петров	277-34-67
B345BB98	«Мерседес»	Васильев	322-98-44
A365CC47	«Ауди»	Иванов	155-77-23

Рис. 3.20

Понятно, что телефон зависит не от регистрационного номера, а от владельца. Поэтому его нужно выносить в другую таблицу (рис. 3.21).

Автомобили

Номер	Автомобиль	Владелец
А123АА47	«Лада-Калина»	1
Т234ТТ78	«Ока»	2
В345ВВ98	«Мерседес»	3
А365СС47	«Ауди»	1

Владельцы

Код	Фамилия	Телефон
1	Иванов	155-77-23
2	Петров	277-34-67
3	Васильев	322-98-44

Рис. 3.21

Заметим, что здесь мы присвоили каждому владельцу уникальный числовой код и ввели в таблицу *Владельцы* суррогатный (искусственный) ключ.

3. **Не должно быть одинаковых по смыслу полей.** Например, фирма торгует бананами, апельсинами и яблоками и хранит данные о ежедневных продажах этих товаров (в килограммах) в такой таблице (рис. 3.22).

Дата	Бананы	Апельсины	Яблоки
21.05.2011	120	78	101
22.05.2011	153	99	65
23.05.2011	87	55	123

Рис. 3.22

Недостаток этой таблицы в том, что поля *Бананы*, *Апельсины* и *Яблоки* — одинаковые по смыслу, это товары. Если фирма начнёт работать с новым видом товара, придется добавлять новый столбец, т. е. менять структуру таблицы (это делает уже не пользователь, а администратор базы данных). Поэтому нужно выделить все товары в отдельную таблицу (рис. 3.23).

Продажи

Дата	Товар	Продано
21.05.2011	1	120
21.05.2011	2	78
21.05.2011	3	101
22.05.2011	1	153
...	...	...

Товары

Код	Название
1	Бананы
2	Апельсины
3	Яблоки

Рис. 3.23

С одной стороны, число строк в таблице увеличилось, но с другой — организация данных улучшилась. Теперь для добавле-

ния в базу нового товара достаточно добавить одну запись в таблицу *Товары*, структуру таблиц переделывать не нужно.

4. **Не нужно хранить данные, которые могут быть вычислены.** Предположим, что бухгалтер фирмы вводит в таблицу доходы и расходы фирмы за каждый месяц (в тысячах рублей), а также прибыль — разницу между доходом и расходом (рис. 3.24).

Дата	Доходы	Расходы	Прибыль
03.2011	155	128	27
02.2011	178	105	73
01.2011	194	159	35

Рис. 3.24

В самом деле, поле *Прибыль* нужно удалить из таблицы, поскольку это значение можно рассчитать как разность двух других. Во-первых, оно занимает лишнее место на диске, во-вторых, появляется возможность нарушения целостности — при ошибке ввода может оказаться, что прибыль не равна разности доходов и расходов, и данные станут противоречивыми. Позже вы узнаете, что все вычисления в базах данных можно делать с помощью *запросов*, которые СУБД выполняет по требованию пользователя, т. е. тогда, когда результаты этих вычислений действительно нужны.

Нужно понимать, что нормализация имеет свои недостатки. Например, выборка данных из нескольких таблиц может выполняться очень долго, и для ускорения работы иногда приходится применять денормализацию — специально вводить избыточность, нарушая требования нормализации. Например, в предельном случае можно все данные свести в одну таблицу. Однако при этом необходимо принимать меры для поддержки целостности базы данных.

Вопросы и задания

1. Опишите проблемы, возникавшие при работе с базами данных в середине XX века. Приведите примеры. Подготовьте сообщение.
2. Расскажите об основных идеях, на которых строится реляционная модель данных.
3. Объясните понятия «кортеж», «отношение».

4. Какие ограничения накладываются на операции с реляционной базой данных?
5. Как связана реляционная модель данных и табличное представление?
6. Какими свойствами должны обладать таблицы в реляционной базе данных?
7. Какие базы данных называются реляционными?
8. Что такое нормализация? Каковы её цели?
9. Как вы понимаете выражение «поле должно быть неделимым»? Приведите примеры.
10. Почему нужно стараться, чтобы структура базы данных (состав таблиц, количество и состав полей) не менялась во время её использования?
11. Почему не нужно хранить в БД данные, которые могут быть вычислены через другие данные?

Задачи

1. Выполните нормализацию однотабличной базы данных заповедника:

Год	Животные	Район	Количество
2009	Белки	Нижняя Балка	12
2009	Бурундуки	Верхняя Балка	5
2010	Еноты	Нижняя Балка	7
2010	Еноты	Овраг	3
2010	Белки	Верхняя Балка	10

2. Выполните нормализацию однотабличной базы данных по военным кораблям:

Год спуска на воду	Название	Проект	Экипаж
1980	Удалой	1155	220 чел.
1985	Адмирал Трибуц	1155	220 чел.
1987	Североморск	1155	220 чел.
1982	Москва	1164	510 чел.
1983	Варяг	1164	510 чел.

3. Выполните нормализацию однотабличной базы данных по автомобилям:

Год	Изготовитель	Город	Модель	Скорость	Цена
2007	ВАЗ	Тольятти	1119	165 км/ч	120000 руб.
1995	ВАЗ	Тольятти	11113	130 км/ч	50000 руб.
1992	КАМАЗ	Набережные Челны	5320	90 км/ч	200000 руб.
2006	КАМАЗ	Набережные Челны	55102	90 км/ч	450000 руб.
2007	БелАЗ	Жодино	75600	64 км/ч	1200000 руб.

4. Выполните нормализацию однотабличной базы данных по участникам музыкального конкурса:

Страна	Фамилия	Инструмент	Автор произведения	Место
Россия	Иванов	Фортепьяно	Рахманинов	1
Россия	Петров	Флейта	Лист	2
Германия	Шмидт	Скрипка	Моцарт	3
США	Смит	Скрипка	Рахманинов	4
США	Браун	Гобой	Моцарт	5

§ 16
Работа с таблицей

Далее для работы с базами данных мы будем использовать свободно распространяемую СУБД Base из пакета OpenOffice.org. Аналогичные приёмы работы применяются и в популярной коммерческой СУБД Microsoft Access (которая обладает ещё бóльшими возможностями), поэтому практические задания можно выполнять в любой из этих программ. В тексте мы будем обращать внимание на некоторые различия между ними.

В СУБД OpenOffice.org Base вся база данных представляет собой один файл с расширением odb. В нём находятся:

- **таблицы**, в которых хранятся данные;
- **формы** — диалоговые окна, с помощью которых пользователь вводит и изменяет данные;

- **запросы** — обращения к базе данных, в результате которых отбираются нужные данные или выполняются какие-то другие действия, например, изменение или удаление записей;
- **отчёты** — шаблоны документов, предназначенных для вывода данных на печать.

Просмотр таблицы

После открытия файла появляется основное окно базы данных (рис. 3.25).

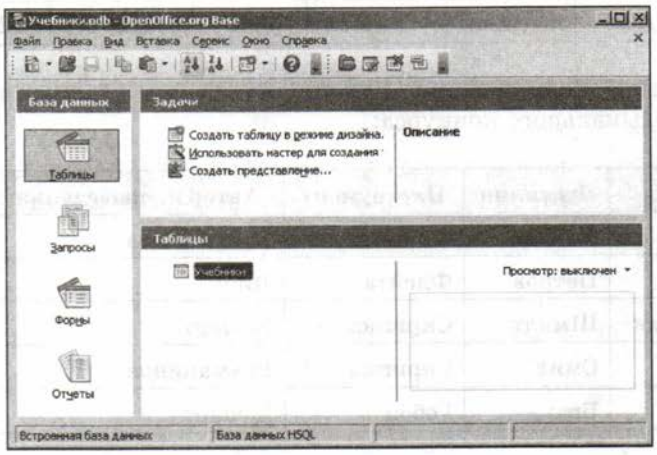


Рис. 3.25

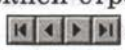
Сейчас в левой части выбрана вкладка *Таблицы*, а в правой части мы видим список всех таблиц (здесь одна таблица *Учебники*).

После двойного щелчка на таблице она открывается в отдельном окне в режиме редактирования данных (рис. 3.26).

Код	Авторы	Название
1	Чуракова Н.А.	Русский язык
2	Чуракова Н.А. и др.	Русский язык. Ч. 1, 2, 3.
3	Бетенькова Н.М., Горецкий В.Г., Фонин Д.С.	Азбука. Ч. 1, 2.
4	Соловейчик М.С., Кузьменко Н.С.	К тайнам нашего языка. Уче
5	Соловейчик М.С., Кузьменко Н.С.	К тайнам нашего языка. Ч.
6	Соловейчик М.С., Кузьменко Н.С.	К тайнам нашего языка. Ч.
7	Соловейчик М.С., Кузьменко Н.С.	К тайнам нашего языка. Ч.
8	Андрианова Т.М.	Букварь
9	Андрианова Т.М., Илюхина В.А.	Русский язык

Запись 6 из 237 (1)

Рис. 3.26

В левой части расположена область выделения, щелчок в ней выделает **текущую** (активную, рабочую) **запись**, которая обозначается треугольником. Номер текущей записи и общее число записей можно увидеть в нижней строке. Там же находятся кнопки для перехода по записям  и кнопка для вставки новой записи (она всегда добавляется в конец таблицы).

Поиск и сортировка

При нажатии на кнопку (или на клавиши Ctrl+F) открывается окно, в котором можно задать образец и режимы поиска (искать вперёд, назад, в текущем поле или во всех полях и т. д.).

Сортировка — это расстановка записей в определенном порядке. Простейший способ сортировки — по текущему столбцу (в котором стоит курсор). Для этого служат кнопки  и  на главной панели инструментов[1], задающие соответственно алфавитный и обратный алфавитный порядок. Кнопка  позволяет применить несколько уровней сортировки, например книги одного автора отсортировать ещё и по году издания.

Важно понимать, что при сортировке *физическое расположение записей в базе данных* (*в файле на диске*) *не изменяется*, они переставляются в списке только при выводе на экран.

Фильтры

На уроках химии вы использовали фильтрацию, чтобы отделить осадок от жидкости. В базах данных фильтр служит для того, чтобы оставить нужные записи и временно скрыть ненужные.

Фильтр — это условие для отбора записей.

Самый простой вариант — это быстрый фильтр (или «фильтр по выделенному»). Он отбирает все записи, в которых значение текущего поля совпадает с активной ячейкой таблицы. Нужно сделать активной ячейку, содержащую требуемое значение, и щёлкнуть на кнопке  на панели инструментов.

Подчеркнём, *что при фильтрации записи, не удовлетворяющие заданному условию, не удаляются из таблицы, а временно*

[1] В русской версии Microsoft Access на аналогичных кнопках написаны русские буквы А и Я.

скрываются. Чтобы снова показать все записи, нужно отменить фильтр, нажав кнопку **(Отменить фильтр)**. При включенном фильтре эта кнопка выделяется серым фоном (активна).

Сложный фильтр, устанавливающий ограничения на несколько полей, можно задать с помощью кнопки **(Фильтр по умолчанию)**. В диалоговом окне задается сложное условие, в котором можно использовать логические операции «И» (AND) и «ИЛИ» (OR) (рис. 3.27).

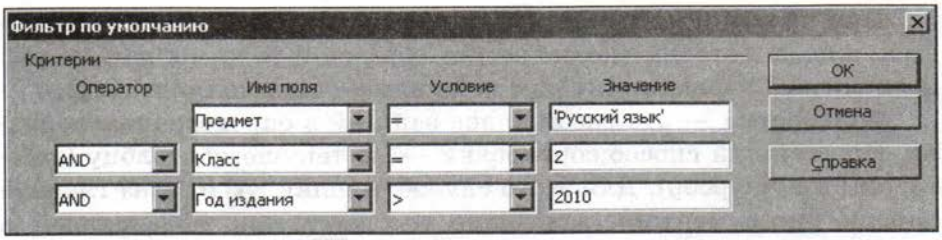


Рис. 3.27

Кнопка удаляет установленный фильтр и отменяет сортировку. Помните, что сначала выполняется операция «И», а потом — операция «ИЛИ».

Фильтры служат для быстрого отбора записей по простым условиям. С каждой таблицей может храниться только один фильтр. Если нужно постоянно использовать несколько разных фильтров или более сложные условия отбора, применяют *запросы*, с которыми мы познакомимся далее.

Вопросы и задания

1. Какие объекты хранятся в файле базы данных OpenOffice.org Base?
2. Как вы думаете, какие достоинства и недостатки имеет идея хранения всех объектов БД в одном файле? Ответ обоснуйте.
3. Почему СУБД (в отличие от табличных процессоров) не разрешает вставлять новую запись в середину таблицы?
4. Что такое сортировка?
5. Изменяется ли при сортировке расположение записей в файле?
6. Что такое многоуровневая сортировка?
7. Что такое фильтр?
8. Какие варианты установки фильтров есть в СУБД OpenOffice.org Base?
9. Можно ли хранить в базе данных несколько разных фильтров для одной таблицы?

Задачи

1. Объясните, есть ли разница между следующими фильтрами?
   а) `Предмет='Математика' AND Класс=2 OR Год издания>2009`
   б) `Предмет='Математика' OR Класс=2 AND Год издания>2009`
   в) `Предмет='Математика' OR Год издания>2009 AND Класс=2`
   г) `Предмет='Математика' AND Год издания>2009 OR Класс=2`
   Есть ли среди этих фильтров два таких, которые отбирают одни и те же записи?

2. Дана таблица с результатами тестирования по различным предметам:

Фамилия	Пол	Математика	Русский язык	Химия	Информатика	Биология
Сомов	м	75	65	70	90	58
Кротов	м	83	75	59	87	60
Белочкина	ж	55	92	64	65	86
Окунев	м	75	68	72	70	56
Судакова	ж	68	70	56	58	60
Щукина	ж	76	58	78	80	85

   Сколько записей будет отобрано с помощью фильтра?
   а) `Пол = 'ж' AND Химия > Биология`
   б) `Пол = 'ж' OR Химия > Биология`
   в) `Пол = 'м' AND Математика > Информатика`
   г) `Пол = 'м' OR Математика > Информатика`
   д) `Пол = 'ж' AND Русский язык > 70 OR Информатика > 80`
   е) `Пол = 'ж' OR Русский язык > 70 AND Информатика > 80`
   ж) `Пол = 'м' AND Информатика > 80 OR Русский язык > 60`
   з) `Пол = 'м' OR Информатика > 80 AND Русский язык > 60`

3. Какой по счёту будет запись с фамилией Белочкина, если отсортировать таблицу из предыдущего задания по полю:
   а) *Фамилия* (по алфавиту);
   б) *Математика* (по убыванию);
   в) *Русский язык* (по убыванию);
   г) *Химия* (по возрастанию);
   д) *Информатика* (по возрастанию);
   е) *Биология* (по убыванию)?

§ 17
Создание однотабличной базы данных

Работать с готовой базой данных вы уже научились, теперь займёмся созданием новой базы «с нуля». Построим однотабличную базу данных *Футбол*, в которую запишем количество побед, ничьих и поражений нескольких футбольных команд за последний сезон, а также среднюю зарплату футболистов (рис. 3.28).

	Команда	Победы	Ничьи	Поражения	Зарплата
	Аметист	10	7	3	13 290 руб.
	Бирюза	5	8	7	12 500 руб.
	Восход	13	5	2	22 000 руб.
	Закат	7	8	5	18 780 руб.
	Коллектор	11	6	3	20 200 руб.
	Кубань	6	12	2	14 000 руб.
	Малахит	12	3	5	17 340 руб.
	Ротор	8	12	0	15 820 руб.
	Статор	9	10	1	19 300 руб.
	Финиш	12	0	8	12 950 руб.

Рис. 3.28

Запустим пакет OpenOffice.org и выберем создание новой базы данных. **Мастер** (программный модуль, который облегчает выполнение стандартных действий) предлагает на выбор три варианта:

- создать новую базу данных;
- открыть существующую базу;
- подключиться к существующей базе, например, к БД Microsoft Access, MySQL или к адресной книге почтовой программы.

Выберем первый пункт. После этого предлагается *зарегистрировать* базу данных — в этом случае вы сможете использовать данные из неё в других программах OpenOffice.org.

В новой базе ещё нет ни одной таблицы. Создать новую таблицу можно двумя способами, которые перечислены в области **Задачи** (рис. 3.29).

Рис. 3.29

Режим дизайна иначе называется **Конструктор**, это означает, что всю работу приходится выполнять вручную. Мастер позволяет быстро построить таблицу на основе одного из готовых шаблонов.

Выбираем режим дизайна. Появляется окно, где нужно в первом столбце ввести названия всех полей новой таблицы, а во втором — выбрать их тип из выпадающего списка (рис. 3.30). Здесь использованы три различных типа данных:

- **Текст [VARCHAR]** — текстовая строка;
- **Целое [INTEGER]** — целое число;
- **Десятичное [DECIMAL]** — десятичное число, которое хранится с заданным числом знаков в дробной части и применяется для работы с денежными суммами.

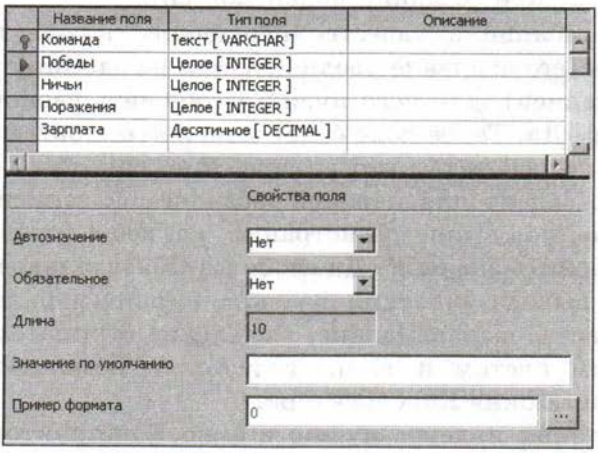


Рис. 3.30

Среди других форматов отметим следующие:

- **Вещественное [REAL]** — вещественное число (может иметь дробную часть);
- **Памятка [LONGVARCHAR]** — текст большого размера;
- **Картинка [LONGVARBINARY]** — двоичные данные большого размера;
- **Логическое [BOOLEAN]** — логическое значение («да»/«нет»);
- **Дата [DATE]**;
- **Время [TIME]**;
- **Дата/Время [TIMESTAMP]**.

Поле *Команда* — это первичный ключ. Чтобы сделать поле ключевым, нужно щёлкнуть правой кнопкой мыши в области выделения (слева от названия поля) и выбрать пункт **Первичный ключ** в контекстном меню. Если выделить несколько полей (щёлкая в области выделения при нажатой клавише Ctrl), можно таким же способом создать *составной ключ* таблицы, состоящий из всех выделенных полей.

В нижней части окна Конструктора настраиваются свойства выделенного поля, например:

- максимальный размер для текста;
- количество знаков в дробной части для десятичного числа;
- значение по умолчанию (которое автоматически вписывается в поле при создании новой записи).

Если в таблице в качестве первичного ключа используется поле-счётчик (его значение увеличивается на единицу при добавлении новой записи), для него нужно установить свойство **Автозначение** равным **Да**. Такое поле будет заполняться автоматически.

Можно потребовать, чтобы значение какого-то поля обязательно было задано для каждой добавляемой записи (вспомните обязательные поля при регистрации на веб-сайтах). Для этого нужно установить значение **Да** свойства **Обязательное**.

Формат вывода значения на экран (например, число знаков в дробной части, выравнивание, выделение отрицательных значений красным цветом и т. п.) задаётся с помощью кнопки  в нижней части окна Конструктора.

Для создания индекса нужно в окне Конструктора щёлкнуть на кнопке  на панели инструментов. Появится окно, в котором можно создавать, удалять и изменять индексы. Индекс по первичному ключу создается автоматически. Любой индекс можно сделать уникальным, в этом случае СУБД не допустит повторения значений индекса.

При закрытии окна Конструктора будет предложено сохранить таблицу и ввести ее название, затем новая таблица появляется в списке таблиц. Для работы с таблицами в окне базы данных служат кнопки верхней панели инструментов:

 — открыть таблицу в режиме редактирования данных;

 — перейти в режим дизайна (в окно Конструктора);

 — удалить таблицу;

 — переименовать таблицу.

Эти же операции можно выполнить с помощью контекстного меню, щёлкнув на имени таблицы правой кнопкой мыши. Для входа в Конструктор нужно выбрать команду **Изменить**.

Вопросы и задания

1. Что такое мастер?
2. Что значит зарегистрировать базу данных?
3. Какие способы создания таблиц вы знаете? Чем они различаются? Приведите примеры.
4. Зачем каждому полю таблицы присваивается некоторый тип данных?
5. Какие типы данных поддерживает OpenOffice.org Base?
6. В чем особенность значений типа **DECIMAL**? Зачем они используются?
7. Как сделать поле первичным ключом? Как изменить ключ таблицы?
8. Как изменить свойства поля?
9. Что обозначают свойства **Автозаполнение** и **Обязательное**?
10. Как изменить формат вывода значений поля?
11. Зачем может понадобиться создавать новый индекс? Как это сделать?
12. Какой индекс в таблице строится автоматически?
13. Как перейти в режим дизайна с помощью контекстного меню?

Задача

Создайте новую базу данных *Футбол* и сохраните её в своей папке. Выполните следующие задания:

а) постройте таблицу, которая рассмотрена в тексте параграфа;

б) определите правильный тип полей, сделайте поле *Команда* первичным ключом;

в) заполните таблицу данными (например, можно использовать данные рис. 3.28);

г) отсортируйте записи по убыванию количества побед;

д) примените фильтр, который отбирает только команды, имеющие более 10 побед и меньше 5 поражений.

§ 18
Запросы

Для пользователя любой информационной системы в первую очередь важно, чтобы он смог выбрать из базы данных ту информацию, которая ему в данный момент нужна. Для этого используются запросы.

Запрос — это обращение к СУБД для отбора записей или выполнения других операций с данными.

Как вы уже знаете, в большинстве современных СУБД для управления данными (т. е. для составления запросов) используется **язык SQL** (англ. *Structured Query Language* — язык структурных запросов). Изучение этого языка выходит за рамки школьного курса, поэтому мы будем использовать в основном визуальные средства составления запросов программы OpenOffice.org Base. В то же время вы сможете посмотреть, как выглядит составленный вами запрос на языке SQL.

Конструктор запросов

Продолжим работу с базой данных *Футбол*, которую мы недавно создали. Перейдём в окне базы данных на вкладку **Запросы** и выберем в области **Задачи** вариант **Создать запрос в режиме дизайна**. Появится окно Конструктора, программа предложит добавить в рабочую область таблицу, из которой будут выбираться данные.

Названия полей, которые нужно включить в запрос, можно перетаскивать мышью в пустые столбцы бланка, расположенного в нижней части окна (рис. 3.31).

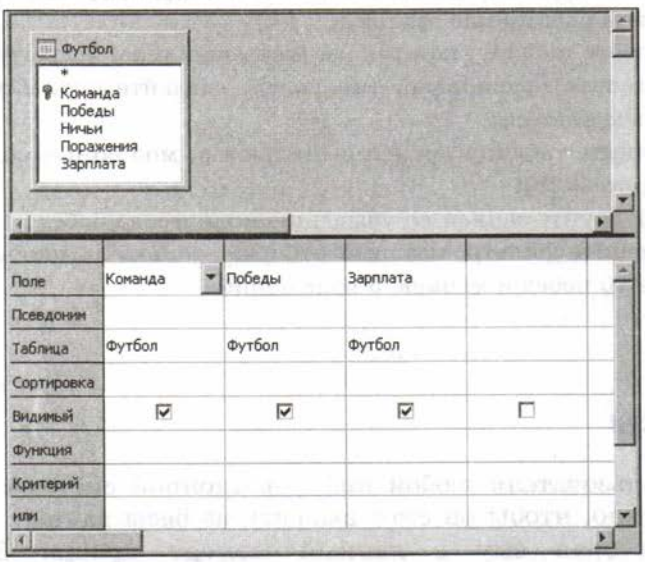


Рис. 3.31

В нашем случае в запрос добавлены поля *Команда*, *Победы* и *Зарплата*.

Чтобы увидеть результаты запроса, нужно нажать клавишу F5 или щёлкнуть на кнопке  на панели инструментов. Результат выполнения запроса — это таблица, которая не хранится в памяти, а строится в момент выполнения запроса. Однако она напрямую связана с данными: если вы измените что-то в результатах запроса, эти изменения будут внесены в базу данных. Такие изменения разрешены только тогда, когда в запросе есть первичный ключ таблицы.

Запрос, который мы построили, программа переводит на язык SQL и отправляет СУБД для выполнения. Чтобы увидеть SQL-запрос, нужно щёлкнуть на кнопке  (повторный щелчок возвращает обратно в режим Конструктора). В данном случае мы увидим такой запрос:

```
SELECT "Команда", "Победы", "Зарплата" FROM "Футбол"
```

В английском языке слово *select* означает «выбрать», а *from* — «из». Таким образом, здесь выбираются три поля из таблицы *Футбол*. В режиме SQL запрос можно редактировать вручную, если вы знаете язык SQL.

Перейдём обратно в Конструктор и сделаем так, чтобы список команд был отсортирован по убыванию числа побед. Для этого в столбце *Победы* нужно установить параметр **Сортировка** равным **по убыванию** (выбрать из выпадающего списка). Теперь можно заново выполнить запрос, нажав F5, и посмотреть на результат.

Если закрыть окно Конструктора, программа предложит сохранить запрос и ввести его имя. После этого запрос появится в списке запросов в основном окне базы данных. Двойной щелчок на имени запроса открывает окно с результатами. Нажав правую кнопку мыши на заголовке столбца таблицы в этом окне, можно настроить формат столбца (тип данных, выравнивание).

Чтобы перейти в Конструктор, нужно выделить название запроса и щёлкнуть на кнопке  на панели инструментов или выбрать пункт **Изменить** из контекстного меню.

Критерии отбора

Теперь отберём только те команды, которые одержали более 10 побед. Для этого в столбце *Победы* зададим критерий отбора *>10* (строка **Критерий**) и проверим запрос.

Добавим второе условие отбора в той же строке **Критерий**: для поля *Зарплата* установим критерий *>15000*. Теперь СУБД отберёт только те записи, для которых одновременно выполняются оба критерия, т. е. условия в одной строке объединяются с помощью логической операции «И» (AND).

Перейдём в режим SQL, найдём в условии слово **AND** и заменим его на **OR** («ИЛИ»). Вернувшись в Конструктор, обнаружим, что условие *>15 000* «переехало» на одну строчку ниже (у этой строчки заголовок **или**). Снова выполним запрос и убедимся, что теперь отбираются команды, для которых выполняется хотя бы одно из двух условий. Таким образом, условия, записанные в одной строке, объединяются с помощью операции «И», а условия в разных строках — с помощью операции «ИЛИ».

Для текстовых данных можно указывать не только точное значение, но и шаблон (вспомните, как строятся маски имён файлов). Например, если в критерий отбора для поля *Команда* ввести

```
LIKE 'К*'
```

то будут отобраны только те команды, название которых начинается с буквы «К». Здесь слово `LIKE` (из языка SQL) обозначает «такой, как...», «похожий на...», а звёздочка — любое количество любых символов. Кроме звёздочки можно использовать знак вопроса, обозначающий один любой символ.

Обратите внимание, что в бланке запроса есть строка **Видимый**, где в каждом столбце расположен флажок (выключатель). Он отключается тогда, когда для данного поля нужно задать условие отбора или сортировку, а выводить его на экран не нужно.

Запросы с параметрами

Создадим запрос, отбирающий команды, в которых зарплата больше заданной, скажем, больше 15 000 рублей. Для этого в критерий отбора для поля *Зарплата* нужно добавить условие *>15 000*.

Если мы захотим изменить это значение, нужно будет изменить запрос. Чаще всего пользователь не имеет права изменять запросы (это делает только администратор базы данных), поэтому возникает проблема: как дать пользователю возможность менять значение в запросе, не меняя сам запрос? Чтобы решить эту задачу, применяют **запросы с параметрами**.

Параметры — это данные, которые пользователь вводит при выполнении запроса.

В конструкторе запроса параметр задаётся с помощью двоеточия, за которым следует имя запроса, например:

```
>=:Минимальная_зарплата.
```

Для соединения двух слов используется знак подчёркивания, потому что в OpenOffice.org Base имя параметра не может включать пробелы[1].

Когда выполняется запрос, на экране появляется окно, в котором пользователь должен ввести значения всех параметров (рис. 3.32).

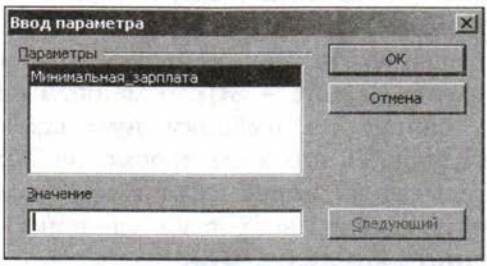


Рис. 3.32

Вычисляемые поля

В теоретической части этой темы мы говорили о том, что в базе данных не нужно хранить значения, которые можно вычислить по другим известным данным. Например, найдём количество очков, которые набрала каждая команда в чемпионате, учитывая, что за победу начисляется 3 очка, а за ничью — одно очко.

В бланке запроса перетащим поле *Зарплата* за его заголовок вправо, освободив 3 столбца. В первые два пустых столбца добавим поля *Ничьи* и *Поражения*, а в третьем вместо имени поля введем нужную нам формулу

```
Ничьи+3*Победы
```

В строке **Псевдоним** можно ввести осмысленный заголовок этого столбца — *Очки*, который и будет появляться в таблице с результатами запроса.

С новым столбцом **(вычисляемым полем)** можно делать всё, что и с обычными столбцами, соответствующими реальным полям

[1] В Microsoft Access этого ограничения нет, там имя параметра заключается в квадратные скобки.

таблицы, — сортировать, устанавливать условия отбора[1]. Например, можно выбрать сортировку по убыванию, чтобы в начале таблицы оказались команды с самым высоким результатом.

Другие типы запросов

Мы рассмотрели только один вид запросов — запросы на выборку данных. Однако язык SQL поддерживает и другие команды, с помощью которых можно полностью управлять базой данных: создавать и удалять таблицы; добавлять, изменять и удалять записи и т. д.

Чтобы использовать эти возможности, в OpenOffice.org Base нужно выбрать пункт **Сервис – SQL** в меню. Кроме того, можно также составить запрос на выборку нужных записей в *Конструкторе*, затем перейти в режим просмотра SQL-запроса (кнопка ), вручную изменить запрос нужным образом (например, из запроса на выборку сделать запрос на удаление) и щёлкнуть на кнопке  (**Выполнить команду SQL**).

Вопросы и задания

1. Что такое запрос? Зачем используются запросы?
2. На каком языке составляются запросы к СУБД?
3. Можно ли строить и изменять запрос, не используя Конструктор?
4. Объясните, зачем нужны строки **Псевдоним**, **Сортировка** и **Критерий** в бланке запроса.
5. Как вы думаете, ограничивает ли структура бланка (одна строка **Критерий** и несколько строк **или**) возможность создания сложных запросов? Ответ обоснуйте (вспомните материал по математической логике).
6. Как увидеть результаты выполнения запроса?
7. Как изменить существующий запрос?
8. Как вы думаете, каков будет результат выполнения следующего SQL-запроса?

```
SELECT "Команда" FROM "Футбол" WHERE "Победы" > 10
```

Проверьте ваш ответ с помощью программы OpenOffice.org Base.

9. Зачем нужны запросы с параметрами?
10. Что такое вычисляемое поле? Как его построить в Конструкторе?
11. Какие операции кроме выборки данных можно выполнить с помощью SQL-запросов?

[1] В OpenOffice.org Base числа в условиях отбора для вычисляемого поля нужно ставить в апострофы: *>'10'*.

Задачи

1. Откройте базу данных *Футбол*, созданную ранее. Постройте следующие запросы и настройте формат вывода данных:
   а) запрос с именем *Запрос85*, который отбирает всю информацию о командах, имеющих более 8 побед и меньше 5 поражений; команды должны быть расставлены по убыванию числа побед;
   б) запрос с именем *ЗапросЗарплата*, который отбирает команды, где зарплата игроков не меньше суммы, введённой пользователем; команды должны быть расставлены по убыванию зарплаты;
   в) запрос с именем *ЗапросОчки*, в котором для каждой команды вычисляется количество набранных очков (за победу — 3 очка, за ничью — 1 очко); команды должны быть расставлены по убыванию количества набранных очков.

   Посмотрите, как эти запросы записываются на языке SQL.
2. Проверьте, какие данные отбирает такой SQL-запрос:
   ```
   SELECT * FROM "Футбол"
   ```
   Посмотрите, как он выглядит в бланке Конструктора. Сделайте выводы.
3. Постройте запрос, отбирающий данные о всех командах, в названии которых есть буква «а» (возможно, в середине). Посмотрите, как он запишется на языке SQL.
4. Постройте запрос, отбирающий данные о всех командах, в названии которых третья буква — «а». Посмотрите, как он запишется на языке SQL.

§ 19
Формы

Пользователи информационных систем для работы с базой данных используют прикладные программы, которые для выполнения любых операций с данными обращаются к СУБД. Некоторые программы, в том числе OpenOffice.org Base и Microsoft Access, могут выполнять функции как СУБД, так и прикладной программы. В них можно построить интерфейс для пользователя, предоставив ему возможность ввода, изменения и удаления записей с помощью диалоговых окон, которые называются **формами**.

Форма создаётся на основе таблицы или запроса. Как и все объекты базы данных, форму можно строить вручную (в **режиме дизайна**) или с помощью **мастера**. На этот раз мы будем использовать второй метод. Перейдите на вкладку **Формы** и выберите (в области **Задачи**) вариант **Использовать мастер для создание формы**.

В первом окне мастера нужно выбрать источник данных — таблицу или запрос (рис. 3.33).

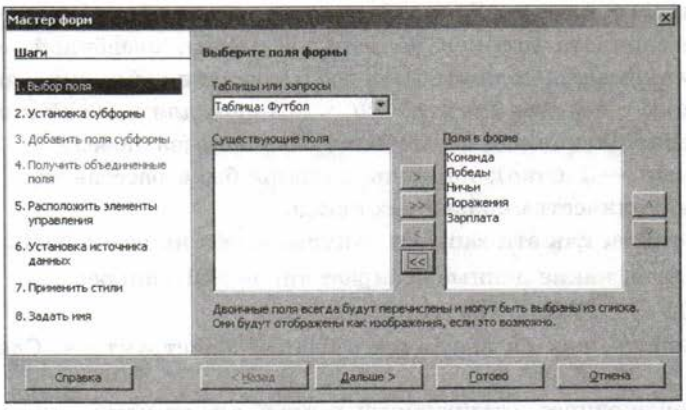


Рис. 3.33

Кнопки со стрелками позволяют добавить поля на форму или отменить выбор. Кнопка с одной стрелкой перемещает только выделенные поля, а кнопка со сдвоенной стрелкой — сразу все поля. Для перехода к следующему шагу мастера нужно нажимать кнопку **Дальше**.

Второй этап (**Установка субформы**, подчинённой формы) мы пропустим, просто нажав на кнопку **Дальше**. Затем нужно выбрать расположение полей с данными, определить стиль оформления и задать имя формы. После нажатия на кнопку **Готово** на последнем шаге мы увидим форму, которую построил мастер (рис. 3.34).

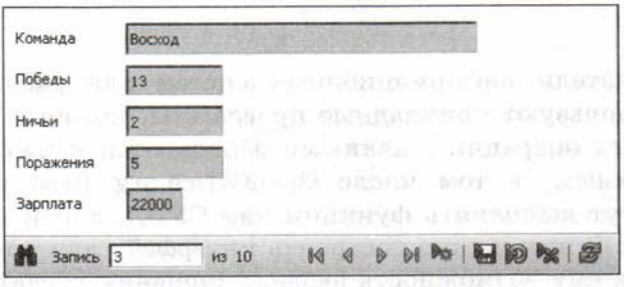


Рис. 3.34

В нижней части окна видны кнопки для поиска, перехода между записями, сохранения и отмены сделанных изменений, удаления записи и др.

Так же как таблицы и запросы, формы можно редактировать в Конструкторе. Для перехода в Конструктор нужно выделить название формы в окне базы данных и щёлкнуть на кнопке на панели инструментов или выбрать пункт **Изменить** из контекстного меню.

На нижней панели инструментов в окне Конструктора расположены кнопки для настройки формы и её элементов. Кнопка позволяет переключаться в режим просмотра данных и обратно. С помощью кнопки можно изменить общие **свойства формы**: название, источник данных, разрешения на добавление, изменение и удаление записей.

Для того чтобы изменить фон формы, нужно щёлкнуть правой кнопкой мыши на свободном месте на форме, выбрать пункт **Страница** из контекстного меню и перейти на вкладку **Фон**. На ней можно выбрать цвет фона или фоновый рисунок.

Чтобы изменить **свойства отдельного элемента**, нужно сначала этот элемент выделить. Вы можете заметить, что щелчок мышью выделяет сразу два элемента: поле и связанную с ним надпись. Чтобы работать с ними по отдельности, нужно при выделении удерживать нажатой клавишу Ctrl. Элементы можно перетаскивать мышью, изменять их размеры за маркеры на рамке.

Для настройки свойств выделенного элемента нужно щёлкнуть на кнопке на нижней панели инструментов или выбрать пункт **Элемент управления** из контекстного меню. На вкладке **Общие** задаётся название элемента, шрифт, цвет фона, размеры, координаты, формат данных, выравнивание и т. п. Вкладка **Данные** определяет источник данных для элемента (название поля).

Кнопка открывает окно **Навигатора форм**. В нём показаны названия всех элементов, щелчком мышью можно выбрать любой из них. С помощью кнопки на форму добавляется новое поле.

Панель **Элементы управления** позволяет добавить на форму флажки (для логических полей), радиокнопки (выбор одного из вариантов), списки, метки и другие элементы (рис. 3.35).

Рис. 3.35

Растровый рисунок можно добавить на форму с помощью команды меню **Вставка – Изображение**. Панель инструментов **Рисование** предназначена для работы с векторной графикой (рис. 3.36).

Рис. 3.36

Таким образом, форма позволяет не только получить доступ к данным, но и оформить их в виде, удобном для пользователя.

Вопросы и задания

1. Что такое форма? Зачем используются формы?
2. Что служит источником данных для формы?
3. Как изменить фон формы?
4. Как изменить источник данных формы?
5. Как выделить поле данных, не выделяя связанную с ним метку?
6. Зачем нужен Навигатор форм?
7. Как можно добавить на форму растровые и векторные рисунки? Попробуйте построить форму, в которой используется графика.

Задача

Постройте форму для просмотра и ввода данных в таблицу *Футбол*. Оформите её так, чтобы работа с данными была как можно проще и понятнее.

§ 20 Отчёты

Отчёт — это документ, предназначенный для вывода данных на печать.

Подготовка отчётов — это задача прикладной программы, а не СУБД. Однако OpenOffice.org Base и Microsoft Access совмещают обе функции, поэтому в них можно оформлять и печатать отчеты. На рисунке 3.37 показан отчёт, построенный на основе запроса к базе данных *Футбол*.

Турнирная таблица

Автор: Василий
Дата: 12.10.11

Команда	Победы	Ничьи	Поражения	Очки	Зарплата
Восход	13	5	2	44	22 000 руб.
Малахит	12	3	5	39	17 340 руб.
Коллектор	11	6	3	39	20 200 руб.
Статор	9	10	1	37	19 300 руб.
Аметист	10	7	3	37	13 290 руб.
Финиш	12	0	8	36	12 950 руб.
Ротор	8	12	0	36	15 820 руб.
Кубань	6	12	2	30	14 000 руб.
Закат	7	8	5	29	18 780 руб.
Бирюза	5	8	7	23	12 500 руб.

Страница 1/1

Рис. 3.37

Для создания отчёта перейдём в окне базы данных на вкладку **Отчеты** и используем мастер (ссылка для запуска мастера находится в области **Задачи**).

Первый шаг напоминает работу мастера создания форм — нужно выбрать источник данных (таблицу или запрос) и определить нужные поля. На следующих шагах задаются метки для обозначения полей (по умолчанию имена полей), группировка (мы пока её не будем использовать), сортировка, стиль оформления. На последнем шаге нужно определить имя отчёта.

Новый отчёт появляется в окне базы данных на вкладке **Отчеты**. Если дважды щёлкнуть на его названии, отчет открывается на экране в готовом виде, его сразу же можно распечатать.

Для того чтобы изменить структуру и оформление отчёта, нужно выделить его название в окне базы данных и щёлкнуть на кнопке  на панели инструментов или выбрать пункт **Изменить** из контекстного меню. Отчет редактируется в текстовом процессоре, вы можете задать название отчёта и имя автора, а также изменить расположение и оформление элементов, например выравнивание, шрифт и т. п.

Вопросы и задания

1. Что такое отчёт?
2. Откуда берутся данные, которые выводятся в отчете?
3. Как изменить расположение и оформление элементов отчета?

Задача

Постройте отчёт вывода на печать данных из запроса *ЗапросОчки*. Оформите его примерно так, как показано на рис. 3.37.

§ 21
Работа с многотабличной базой данных

Таблицы и связи между ними

Вспомним про базу данных кафе, о которой мы говорили в конце § 14 (рис. 3.38).

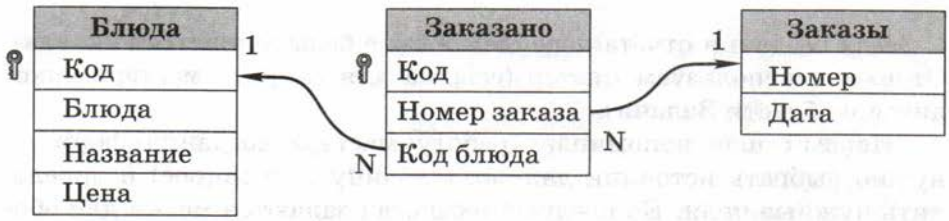


Рис. 3.38

Построим в новой базе данных (назовём её *Кафе*) все необходимые таблицы (пока без связей). Поскольку в этой базе несколько таблиц, далее мы будем использовать общепринятые обозначения типа *Блюда*.[*Название*] — это означает «поле *Название* таблицы *Блюда*».

Не забудьте, что связи устанавливаются только между однотипными полями, т. е. поля *Заказано*.[*Номер заказа*] и *Заказано*.[*Код блюда*] должны быть целого типа (**INTEGER**), чтобы их можно было связать соответственно с номером заказа и кодом блюда. Для поля *Блюда*.[*Цена*] выберите десятичный тип (**DECIMAL**) и денежный формат для вывода на экран.

Чтобы установить связи между таблицами, выберем пункт меню[1] **Сервис – Связи**. С помощью специального окна добавим в рабочую область (она пока пустая) все три таблицы (рис. 3.39).

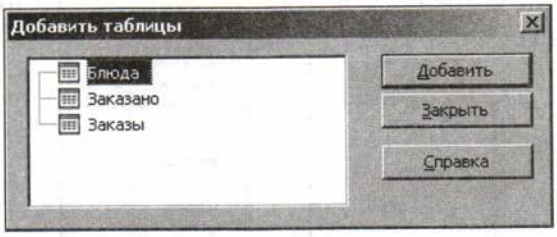


Рис. 3.39

Теперь можно «схватить» мышью название какого-то поля и перетащить его на название поля другой таблицы, с которым нужно установить связь. С помощью этого метода установим все связи, показанные на схеме в начале параграфа. После этого окно можно закрыть, сохранив изменения.

Чтобы изменить или удалить связи, снова зайдите в меню **Сервис – Связи**. Для удаления связи её нужно выделить щелчком мышью и нажать клавишу Delete.

Теперь остаётся заполнить таблицы (можно придумать свои данные или взять их из § 14).

Запрос данных из нескольких таблиц

Мы построим два запроса к базе данных *Кафе* — простой и итоговый. Начнём с простого запроса, в котором будет собрана информация по всем сделанным заказам (рис. 3.40).

Заказ	Дата	Блюдо	Цена
1	11.12.12	борщ	80 руб.
1	11.12.12	гуляш	70 руб.
1	11.12.12	чай	10 руб.
2	12.12.12	борщ	80 руб.
2	12.12.12	бифштекс	110 руб.
2	12.12.12	бифштекс	110 руб.
2	12.12.12	кофе	50 руб.

Рис. 3.40

[1] В программе Microsoft Access нужно щёлкнуть на кнопке **Схема данных** на вкладке **Работа с базами данных**.

Посмотрим, откуда нужно взять данные. Номер заказа (в столбце *Заказ*) и дата хранятся в таблице *Заказы*, а название блюда и цена — в таблице *Блюда*. Перейдём к созданию запроса в режиме дизайна и добавим в рабочую область две названные таблицы (рис. 3.41).

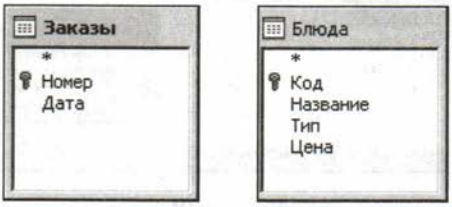


Рис. 3.41

Теперь добавим в столбцы бланка все нужные поля и выполним запрос. Результаты окажутся неожиданными — запрос выдаёт слишком много записей, больше, чем есть на самом деле. Дело в том, что таблицы в запросе оказались несвязанными: мы не добавили таблицу *Заказано*. Из неё не берутся данные, но она служит для связи таблиц в единую систему.

Чтобы исправить ситуацию, выберем пункт меню **Вставка – Добавить таблицу или запрос** и добавим таблицу *Заказано*. После этого запрос отберёт данные правильно. Остаётся только дать столбцам запроса понятные названия, для этого используется строка **Псевдоним** в бланке запроса.

Теперь окно конструктора запросов можно закрыть, при сохранении дайте запросу название *ЗапросЗаказы*. Обратите внимание, что нельзя называть запрос именем, которое совпадает с именем таблицы (например, *Заказы*) или другого запроса.

Двойной щелчок на имени запроса запускает его на выполнение. Щёлкнув правой кнопкой мыши на заголовке столбца, можно изменить его оформление. Установите таким образом денежный формат для поля *Цена* и выравнивание вправо для всех числовых данных.

Итоговый запрос

Теперь построим запрос, который считает общую стоимость каждого заказа (рис. 3.42).

	Заказ	Дата	К оплате
▶	1	11.12.12	160 руб.
	2	12.12.12	350 руб.

Рис. 3.42

Как видно из рис. 3.42, нам нужны номер и дата заказа (из таблицы *Заказы*) и цены блюд (из таблицы *Блюда*), которые нужно как-то сложить. Посмотрев на схему соединения таблиц, легко понять, что для правильного объединения данных нужно добавить в запрос таблицу *Заказано*, хотя ни одно её поле в результате запроса не выводится.

Создадим новый запрос в режиме дизайна и добавим в рабочую область таблицы *Заказы*, *Заказано* и *Блюда*. Перетащим в бланк запроса нужные поля и введём правильные заголовки столбцов (как на образце) в строке *Псевдоним*. Если выполнить этот запрос, мы увидим цены отдельных блюд, а не общую стоимость.

Чтобы подсчитать сумму цен по каждому заказу, в поле *Цена* найдём строку **Функция**[1] и там выберем вариант **Сумма**. Кроме того, надо указать, какие группы записей объединять при суммировании. Для этого в той же строке **Функция** в полях запроса *Номер* и *Дата* выберем вариант **Group** (Группировка). Это значит, что сумма считается для каждой уникальной комбинации «номер запроса – дата». Поскольку у нас номер заказа уникальный, фактически будет рассчитана сумма каждого заказа. Выполним запрос и проверим, что он работает правильно. Теперь можно закрыть окно Конструктора, сохранив запрос под именем *ЗапросКОплате*. Запустим запрос двойным щелчком и настроим формат вывода данных.

Кроме функции **Сумма** в итоговых запросах можно использовать и другие функции, например количество, среднее значение, минимум, максимум.

Формы

Построим форму, показанную на рис. 3.43, в которой для каждого номера заказа выдаётся его дата, состав заказанных блюд с ценами и общая сумма.

Форма создаётся на основе таблицы или запроса. В данном случае мы хотим получить информацию о заказе, поэтому основной источник данных — это таблица — *Заказы*.

Информацию о названиях блюд и ценах можно было бы взять из таблицы *Блюда*,

Рис. 3.43

[1] В Microsoft Access эта строка называется **Групповая операция**. Чтобы она появилась в бланке запроса, нужно щёлкнуть по кнопке Итоги на вкладке Конструктор.

но она напрямую не связана с таблицей *Заказы*, поэтому так сделать не получится. Однако у нас есть запрос *ЗапросЗаказы*, где эти данные объединены, поэтому состав заказа и цены на отдельные блюда мы возьмём из этого запроса.

Как получить общую сумму? Вспомним про *ЗапросКОплате*, где сумма для каждого заказа уже найдена. Таким образом, форма объединяет информацию из таблицы *Заказы* и двух запросов. Связь между ними устанавливается по номеру заказа — это поле есть везде.

Как мы уже упоминали, основной источник данных — это таблица *Заказы*. Щёлкнем на её имени правой кнопкой мыши и выберем пункт **Мастер форм** из контекстного меню. Включим в форму все поля.

На втором шаге мастер предлагает добавить **субформу** (подчинённую форму) — дополнительный источник данных, связанный с главной формой. Отметим флажок **Добавить субформу** (рис. 3.44).

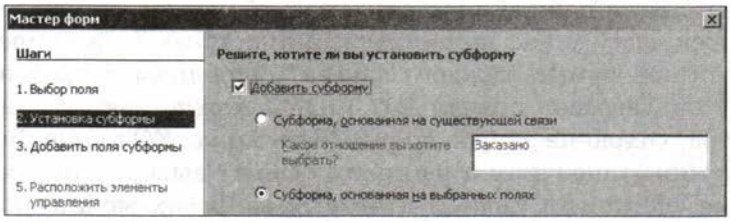


Рис. 3.44

Мастер определил, что для таблицы *Заказы* подчинённой является таблица *Заказано* (они связаны отношением 1:*N*) и предлагает выбрать эту таблицу. Однако нам нужно использовать запрос, поэтому отметим второй вариант: **Субформа, основанная на выбранных полях**. Далее выберем поля *Заказ*, *Блюда* и *Цена* из запроса (рис. 3.45).

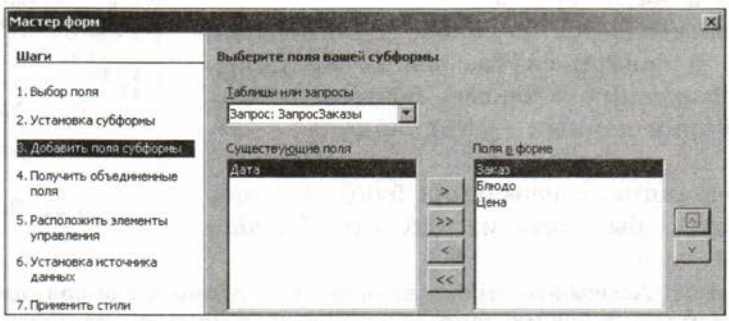


Рис. 3.45

Несмотря на то что поле *Заказ* не будет выводиться на экран, его нужно включить в субформу, потому что через это поле на следующем шаге работы мастера устанавливается её связь с главной таблицей (рис. 3.46).

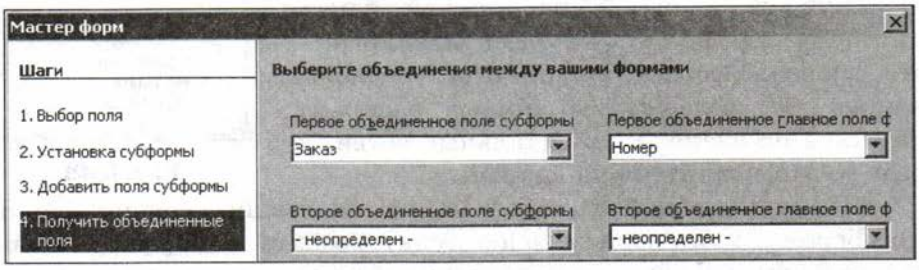


Рис. 3.46

Затем выберем расположение элементов главной и подчинённой форм (рис. 3.47).

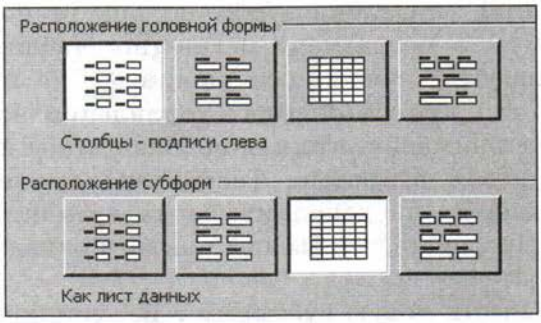


Рис. 3.47

После завершения работы мастера нужно отрегулировать размеры полей и настроить формат вывода. Например, для правильного отображения цены нужно войти в конструктор форм, щёлкнуть правой кнопкой мыши на столбце *Цена* и выбрать пункт **Заменить на – Поле валюты**. Количество десятичных знаков можно изменить с помощью свойства **Точность** в окне свойств столбца (пункт **Столбец** в контекстном меню). Столбец *Заказ* в подчинённой форме нужно просто удалить (пункт **Удалить столбец** в контекстном меню).

Осталось вывести общую стоимость заказа. Напомним, что её нужно взять из другого источника данных — запроса *ЗапросКОплате*. Это значит, что необходимо добавить к форме ещё одну

субформу. Мастер тут не поможет, придётся выполнить эту операцию вручную.

Откроем форму в **режиме дизайна (Конструктора)** и запустим **Навигатор форм**, щёлкнув на кнопке на нижней панели инструментов (рис. 3.48). Здесь **MainForm** — это главная форма, а **SubForm** — подчинённая (её содержимое можно раскрыть, щёлкнув на знаке «+»). Остальные ветви дерева — это метки и поля данных.

Рис. 3.48

Щёлкнем правой кнопкой мыши на имени главной формы и выберем в контекстном меню пункт **Создать – Форма**. В Навигаторе форм появится новая форма с именем *Форма*, но на экране ничего не меняется, ведь мы пока не добавили в эту форму ни одного элемента.

Выделите название новой формы и создайте метку (надпись) с текстом «К оплате». Для этого надо щёлкнуть на кнопке на панели **Элементы управления** (обычно она расположена слева) и выделить область по размеру метки (под таблицей). Свойства метки настраиваются с помощью кнопки на нижней панели инструментов или пункта **Элемент управления** из контекстного меню. Обратите внимание, что в окне Навигатора форм эта метка принадлежит новой субформе. Теперь аналогично добавьте на форму поле валюты из дополнительных элементов управления (кнопка). Проверьте, чтобы оно также принадлежало второй субформе.

Осталось связать новую субформу с источником данных. Для этого выделим субформу в окне Навигатора форм и выберем **Свойства** в контекстном меню. На вкладке **Данные** установим тип содержимого — запрос, в следующей строчке выберем *ЗапросКОплате* (рис. 3.49).

Рис. 3.49

Теперь в окне свойств добавленного поля валюты на вкладке **Данные** надо выбрать поле данных *КОплате*. После этого форма полностью работоспособна.

Отчёты

Используя подстроенный ранее запрос, составим подробный отчёт по всем заказам (рис. 3.50). Обратите внимание, что заказы сгруппированы по датам, кроме того, данные о блюдах, относящихся к одному заказу, тоже расположены вместе. Это **отчёт с группировкой**, причем здесь использованы два уровня группировки — сначала по дате, а потом — по номеру заказа.


Заказы
Исполнитель: Василий Петров
Дата: 15.12.12
Дата 11.12.12
Заказ 1
Блюдо | Цена
гупяш | 40 руб.
борщ | 30 руб.
рассольник | 20 руб.
Дата 12.12.12
Заказ 2
Блюдо | Цена
компот | 10 руб.

Рис. 3.50

Перейдём на вкладку **Запросы**, выделим запрос с именем *ЗапросЗаказы* и выберем пункт **Мастер отчетов** из контекстного меню. Добавим в отчёт все поля запроса, а на третьем шаге установим два уровня группировки — сначала по полю *Дата*, затем — по полю *Заказ* (рис. 3.51).

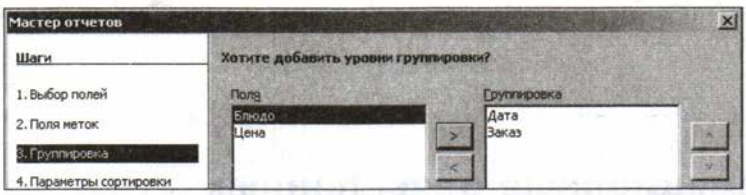


Рис. 3.51

Остальные шаги выполняются так же, как и для простых отчётов, с которыми мы работали раньше.

К сожалению, в настоящей версии OpenOffice.org Base можно строить только самые простые отчёты. Например, невозможно подсчитать общую стоимость каждого заказа или включить в отчёт данные из двух источников (таблиц или запросов), как это мы делали для форм. Нужно отметить, что Microsoft Access обладает значительно бóльшими возможностями для создания профессиональных отчётов.

Вопросы и задания

1. Как установить связи между таблицами?
2. Как вы думаете, по каким признакам программа при установке связи автоматически определяет ее тип (1:1, 1:*N*, *N*:1)?
3. Как строится запрос с выбором данных из нескольких таблиц?

4. Почему иногда в запрос приходится добавлять таблицы, данные из которых не появляются в результатах запроса?
5. Что такое итоговый запрос? Зачем он нужен?
6. Что такое группировка?
7. Что означает группировка по нескольким полям?
8. Объясните, что даёт группировка при построении отчёта.
9. Какие функции можно использовать в итоговых запросах?
10. Как построить форму с подчинённой формой (с субформой)?
11. Как добавить подчинённую форму с помощью Навигатора форм?
12. Как связать новую субформу с главной формой?

Подготовьте сообщение

а) «Работа с базами данных в браузере Firefox»

б) «Работа с базами данных в браузере Chrome»

Задача

Постройте базу данных, запросы, форму и отчёт так, как показано в тексте параграфа.

§ 22
Нереляционные базы данных

Проблемы реляционных БД

Несмотря на то что реляционные (табличные) БД хорошо себя зарекомендовали и составляют подавляющее большинство реально используемых баз данных, они не универсальны и в некоторых задачах их применение приводит к серьёзным проблемам.

Во-первых, для того чтобы построить надёжно работающую реляционную БД, необходимо представить исходные данные как набор взаимосвязанных таблиц. Это требует серьёзных усилий, кроме того, данные становятся менее понятными для человека, потому что он мыслит не таблицами, а объектами, которые имеют определённый набор свойств.

Во-вторых, данные, связанные с одним объектом, разбросаны по разным таблицам, поэтому при поиске их приходится «вытаскивать» с помощью сложных запросов, которые выполняются сравнительно медленно при больших объёмах данных.

В-третьих, во всех реляционных базах данных структура данных чётко определена, причем она задаётся разработчиком при создании базы, и изменить её весьма непросто. Теперь представим себе, что нам нужно хранить документы, которые могут иметь разное количество свойств с разными названиями и типами данных, причём эти свойства заранее не определены и могут меняться со временем. В этом случае использовать реляционные БД, по крайней мере, очень сложно, поскольку они для этой цели не предназначены.

В-четвёртых, объемы данных, которые нужно обрабатывать, всё время возрастают, сейчас базы данных поисковых систем могут достигать нескольких *петабайтов*[1]. Поэтому один компьютер с этим справиться не в силах (система может получать сотни тысяч запросов в секунду). Возникает задача распределить нагрузку на большое количество (иногда десятки тысяч) серверов, связанных между собой через Интернет. Если при этом применять реляционную базу данных, то для выполнения даже простых запросов нужно обращаться ко многим серверам, это недопустимо замедляет поиск. Подобная проблема возникает при «облачных» вычислениях, при которых данные пользователя хранятся на серверах в Интернете. Таким образом, реляционные БД хороши, когда вся база находится на одном компьютере, но они плохо *масштабируются*. Это значит, что при увеличении объема данных и количества запросов не удаётся увеличивать мощность системы, просто добавляя новые сервера. Причина в том, что реляционная модель плохо подходит для распределённых информационных систем.

Базы данных «ключ — значение»

В последние годы появились базы данных нового типа, получившие название «ключ — значение» (англ. *«key-value» database*), которые хорошо показали себя в распределённых системах с большой нагрузкой, в том числе в поисковых системах Интернета.

Базу данных «ключ — значение» можно представить себе как огромную таблицу, в каждой ячейке которой могут храниться произвольные данные («значения»), их структура никак не ограничена. Каждому значению присваивается некоторый код («ключ»), по которому его можно найти. Все данные, относящиеся к конкретном объекту, хранятся в одном месте, поэтому при запросе не нуж-

[1] 1 Пбайт = 2^{10} Тбайт = 2^{50} байт.

но обращаться к разным таблицам, а достаточно просто найти значение по ключу.

СУБД поддерживает только добавление записи, поиск значения по ключу, а также изменение и удаление найденной таким образом записи. Никакие связи между значениями в явном виде не поддерживаются. Хотя объект может содержать ссылки на другие объекты (их ключи), СУБД не проверяет их правильность. Обеспечение надёжности и целостности данных возлагается не на СУБД, а на прикладную программу, которая работает с базой данных.

Ключи — это **хэш-коды** хранящихся данных (значений). Ключи объединяются в группы так, что все данные, связанные с ключами одной группы, хранятся на одном сервере. Таким образом, по ключу можно сразу определить нужный сервер и напрямую получить от него данные. За счет этого обеспечивается масштабируемость — если один сервер не справляется с нагрузкой, нужно добавить ещё один и разделить данную группу ключей на две части.

Многие базы типа «ключ — значение» хранят **документы** — объекты, которые имеют произвольный набор полей-свойств, например:

```
{
  ключ: 123123978623476239476923 7
  автор: "А.С. Пушкин"
  название: "Евгений Онегин"
}
```

Важно, что другие документы могут иметь совершенно другой набор полей. Такие базы данных называют **документоориентированными**.

Итак, базы данных «ключ — значение» обладают **достоинствами**, которые принципиально важны в некоторых задачах:

- *масштабируемость* — возможность наращивания мощности распределенной системы простым добавлением новых серверов;
- *простота* представления данных, близость к человеческому восприятию.

В то же время у них есть и **недостатки**:

- СУБД *не поддерживают связи* между данными, не обеспечивает целостность данных;

- нет стандарта на язык описания и управления данными (для реляционных БД таким стандартом стал язык SQL);
- основной вид запросов — поиск значения по ключу, поэтому *очень сложно*, например, *выполнить сортировку данных*.

В первую очередь, базы данных «ключ — значение» используются при «облачных» вычислениях: в поисковой системе Google (система хранения данных *BigTable*), интернет-магазине *Amazon* (www.amazon.com, база данных *SimpleDB*), энциклопедии *Википедия* (ru.wikipedia.org), социальной сети *Facebook* (www.facebook.com).

Кроме того, есть бесплатные СУБД этого класса, например *MongoDB* (www.mongodb.org) или *CouchDB* (couchdb.apache.org).

Вопросы и задания

1. Назовите недостатки реляционных БД. В каких задачах они проявляются?
2. Что такое распределённые базы данных?
3. Что такое масштабируемость? Почему реляционные БД плохо масштабируются?
4. Как вы понимаете выражение «ключ — значение»?
5. Как обеспечиваются связи между объектами в таких БД? Как обеспечивается целостность данных?
6. Вспомните, что такое хэширование и хэш-коды. Как можно выполнить масштабирование на основе хэш-кодов?
7. Что такое документоориентированные базы данных?
8. Назовите достоинства и недостатки баз данных типа «ключ — значение». В каких задачах их имеет смысл использовать?
9. Что вы думаете о дальнейшей судьбе реляционных БД в связи с появлением новых принципов хранения данных? Проведите исследование.

Подготовьте сообщение

«Нереляционные базы данных: за и против»

Задача

*Попробуйте поработать с какой-нибудь документоориентированной СУБД.

§ 23
Экспертные системы

Эксперт — это человек, который обладает глубокими теоретическими *знаниями* и практическим *опытом работы* в некоторой области. Например, врач-эксперт хорошо ставит диагноз и лечит потому, что имеет медицинское образование и большой опыт лечения пациентов. Он не только знает факты, но и понимает их взаимосвязь, может объяснить причины явлений, сделать прогноз, найти решение в конфликтной ситуации.

Эксперт может ответить на вопросы, на которые нельзя найти ответы в поисковых системах Интернета. Более того, эксперт часто может предложить решение **плохо поставленных задач**, например при нехватке, неточности или противоречивости исходных данных. Для этого он использует свой опыт и интуицию, которые сложно представить в виде формального алгоритма. В то же время *нельзя гарантировать* правильность решения, принятого в таких сложных случаях, хотя иногда эксперт может примерно оценить *вероятность* своей версии. Например, можно получить ответ: «С вероятностью 80% вы поступите в вуз».

Экспертная система — это компьютерная программа, задача которой — заменить человека-эксперта при выработке рекомендаций для принятия решений в сложной ситуации.

Разработка экспертных систем — это одно из направлений развития **искусственного интеллекта**, потому что такие программы пытаются моделировать мышление человека. Результат работы экспертной системы — это не числа, а конкретный совет (рекомендация) в словесной форме.

Экспертные системы применяются в медицине, электронике, геологии, для решения военных и управленческих задач. Первая экспертная система Dendral была создана в Стэнфордском университете в конце 1960-х гг. для определения строения органических молекул по их химическим формулам и свойствам.

В составе экспертной системы выделяют три основные части:

- *базу знаний*;
- *блок получения решения* («решатель»);
- *интерфейс с пользователем*.

База знаний отличается от базы данных тем, что в ней хранятся не только факты, но и правила, по которым из фактов делаются выводы.

Факты — это утверждения, которые считаются истинными, например:

- у окуня есть жабры;
- Иван — отец Марьи;
- Волга впадает в Каспийское море.

Правила обычно формулируются в виде «если..., то», например:

- если x — животное и x дышит жабрами, то x — рыба;
- если x — отец y и y — отец z, то x — дед z;
- если x состоит из атомов углерода и обладает высокой твёрдостью, то x — алмаз.

Таким образом, в отличие от данных, знания представляют собой общие связи предметов, понятий и явления; часто их формулировка содержит переменные[1].

В разработке экспертной системы участвуют эксперты и специально обученный специалист — **инженер по знаниям**, задача которого — представить знания экспертов в такой форме, чтобы они могли быть записаны в базу знаний. Важно, чтобы эту базу знаний можно было постепенно пополнять — добавлять новые правила вывода независимо от предыдущих.

Второй блок экспертной системы — «решатель» — это программа, которая моделирует рассуждения эксперта, используя предоставленные ей исходные данные и информацию из базы знаний. В результате работы она не только выдаёт заключение, но и может подробно показать, как оно было получено (какие правила были использованы). Решатель — это универсальная программа, которая может работать с любой базой знаний понятного ей формата.

Как правило, пользователь работает с экспертной системой в режиме диалога, отвечая на её вопросы. Рассмотрим простейшую экспертную систему для определения класса животных. Предположим, что в базу знаний внесены следующие правила:

- если у животного есть перья, то это птица;
- если животное дышит жабрами, то это рыба;

[1] Вспомните, что в логике утверждение с переменными называется *предикатом*.

- если животное кормит детёнышей молоком, то это млекопитающее;
- если животное — млекопитающее и ест мясо, то это хищник.

Диалог пользователя с экспертной системой может быть, например, таким (ответы пользователя выделены курсивом):

– Это животное кормит детей молоком?
– *Нет.*
– Это животное имеет перья?
– *Нет.*
– Это животное дышит жабрами?
– *Да.*
– Это рыба.

Для того чтобы определить последовательность вопросов, эксперт и инженер по знаниям строят дерево решений, например такое (рис. 3.52).

Рис. 3.52

Конечно, эта экспертная система неполна и в некоторых случаях класс животного определить с её помощью не получится (см. знаки вопроса на схеме).

Простую экспертную систему можно написать на любом языке программирования, в котором есть операторы ввода, вывода и ветвления. Однако для профессиональных разработок в этой области чаще всего применяют специальные языки, например **язык логического программирования Пролог**. Программа на Прологе — это набор фактов и правил вывода. Алгоритм решения задачи писать не нужно, решающая система сама находит ответы на вопросы, заданные в определённой форме.

Итак, экспертная система обладает следующими свойствами:

- применяется в определённой достаточно узкой области;
- использует базу знаний, которая может расширяться;
- может применяться при неточных и противоречивых данных;
- выдаёт ответ в виде рекомендации (совета по принятию решения);
- может показать, как получено решение (какие правила применялись).

В последние годы прогресс в области экспертных систем не очень заметен, что связано с их серьёзными недостатками:

- опыт и интуицию экспертов очень сложно формализовать, свести к чётким правилам;
- отладка и проверка экспертных систем очень сложна, таким образом, трудно гарантировать правильность выводов (это особенно важно в военной области);
- экспертные системы не способны самообучаться, для их поддержки необходима постоянная работа инженера по знаниям.

Вопросы и задания

1. Что такое экспертная система? Из каких элементов она состоит?
2. Чем отличается база знаний от базы данных?
3. Перечислите особенности экспертных систем. В каких областях они применяются?
4. Что входит в задачи инженера по знаниям?
5. Почему развитие экспертных систем в последние годы идёт не очень активно?

Подготовьте сообщение

а) «Что такое база знаний?»

б) «Что делает инженер по знаниям?»

в) «Применение экспертных систем»

г) «Язык программирования Пролог»

Задачи

1. Придумайте какую-нибудь задачу, в которой нужно принять решение. Постройте для неё базу знаний в форме «если..., то...» и дерево решений. Это может быть, например, прогноз погоды, отдых на выходных, поездка за рубеж и т. п.

*2. Постройте для вашей задачи экспертную систему на каком-нибудь языке программирования.

Практические работы

Работа № 13 «Работа с готовой таблицей»
Работа № 14 «Создание однотабличной базы данных»
Работа № 15 «Создание запросов»
Работа № 16 «Создание формы»
Работа № 17 «Оформление отчёта»
Работа № 18 «Язык SQL»
Работа № 19 «Построение таблиц в реляционной БД»
Работа № 20 «Создание формы с подчинённой»
Работа № 21 «Создание запроса к реляционной БД»
Работа № 22 «Создание отчёта с группировкой»
Работа № 23 «Нереляционные БД»
Работа № 24 «Простая экспертная система»

ЭОР к главе 3 на сайте ФЦИОР (http://fcior.edu.ru)

- Проектирование объектов данных
- Ввод данных в БД
- Проектирование экранных форм
- Запросы на выборку данных
- Проектирование отчётов

Самое важное в главе 3

- Информационная система (ИС) в широком смысле — это аппаратные и программные средства, предназначенные для того, чтобы своевременно обеспечить пользователей нужной информацией.
- База данных (БД) — это специальным образом организованная совокупность данных о некоторой предметной области, хранящаяся во внешней памяти компьютера.
- Система управления базой данных (СУБД) — это программные средства, которые позволяют выполнять все необходимые операции с базой данных.

- База данных и система управления базой данных (СУБД) неразрывно связаны: свойства базы данных определяются СУБД, которая ей управляет, и наоборот.
- Пользователь чаще всего работает с СУБД через прикладную программу. Некоторые программы, например, OpenOffice.org Base и MS Access, выполняют функции СУБД и прикладной программы.
- Большинство современных баз данных строятся на основе реляционной модели данных и могут быть представлены в виде набора таблиц. Строки в таблицах называют записями, а столбцы — полями.
- Для управления реляционными базами данных используется язык запросов SQL. С его помощью можно выбирать нужные данные, а также добавлять, изменять и удалять записи и таблицы.
- Для ускорения поиска в базах данных используют индексы — дополнительные таблицы, в которых записи отсортированы по какому-то критерию. Благодаря предварительной сортировке, в индексах можно использовать двоичный поиск, который работает значительно быстрее линейного.
- Чтобы не допустить потери данных при сбоях во время выполнения сложных операций, используют транзакции, каждая из которых должна быть либо выполнена полностью, либо не выполнена вообще.
- Для обработки очень больших массивов данных применяют распределённые БД, в которых данные расположены на многих серверах в сети Интернет. Многие из этих баз данных являются документоориентированными (базы данных «ключ — значение»).

Глава 4
Создание веб-сайтов

§ 24
Веб-сайты и веб-страницы

Введение

Всемирная паутина, или **веб** (англ. **World Wide Web**), — это самая мощная служба Интернета. Она содержит гипертекстовые документы, связанные между собой гиперссылками.

Гипертекст (англ. *hypertext*) — это текст, содержащий гиперссылки.
Гиперссылка (англ. *hyper reference*) — это ссылка на другие объекты (части этого же документа, другие документы, файлы, папки, программы и т. д.).

Например, ссылки на другие статьи в энциклопедиях — это тоже гиперссылки.

Гипертекстовые документы в Интернете называются **веб-страницами**; обычно ссылки на веб-страницах выделяются цветом и подчёркиваются.

Современные веб-документы включают не только текст, но и другие виды информации — рисунки, звук, видео, поэтому для них используется выражение гипермедиа.

Гипермедиа (англ. *hypermedia*) — это гипертекстовый документ, содержащий изображения, звук, видео, причём каждый элемент может быть гиперссылкой.

Веб-страницы размещаются на серверах. Слово «сервер» имеет несколько значений. Сервером называют специальный компьютер, который выделен для обслуживания пользователей (например, для хранения файлов). Другое значение термина «сервер» — программа, которая обеспечивает работу какой-нибудь службы.

Например, для того чтобы пользователи могли обращаться к веб-страницам, на компьютере, где они хранятся, нужно запустить веб-сервер.

Веб-сервер — это программа, которая принимает запросы по протоколу HTTP[1] и отвечает на них — возвращает веб-страницы и дополнительные данные (рисунки, звуковые файлы, видеофайлы).

Для просмотра веб-страниц на экране нужна специальная программа — **браузер** (англ. *browser* — «просмотрщик»). Браузер — это клиентская программа, её задача — послать запрос веб-серверу, получить в ответ веб-страницу и показать её на экране.

Группа веб-страниц, которые объединены общей темой и оформлением, связаны гиперссылками и (чаще всего) расположены на одном сервере, называется **веб-сайтом**. Основа сайта — это его **контент** (англ. *content* — содержание, информационное наполнение), т. е. те материалы, которые могут заинтересовать пользователя. Вторая важная составляющая сайта — **дизайн**, т. е. оформление материала, способ подачи информации. Задача дизайна — сделать использование сайта удобным для читателей.

Статические и динамические веб-страницы

Веб-страницы — это обычные текстовые файлы (в формате «только текст», англ. *plain text*). Для того чтобы определить структуру документа (заголовки, абзацы, списки и др.), используют **язык HTML** (англ. *HyperText Markup Language* — язык разметки гипертекста).

В языке HTML используются команды особого типа — **тэги** (англ. *tag* — метка, ярлык). Существуют тэги для выделения заголовков, абзацев, вставки таблиц. С помощью тэгов в веб-страницы добавляют рисунки, звуки, анимацию, видео, которые хранятся на сервере в виде отдельных файлов. Часто для дополнительных данных на сайте создаются специальные каталоги, например, рисунки могут быть размещены в каталоге images, звуковые и видеофайлы — в каталоге media (рис. 4.1).

[1] *HyperText Transfer Protocol* — протокол передачи гипертекста.

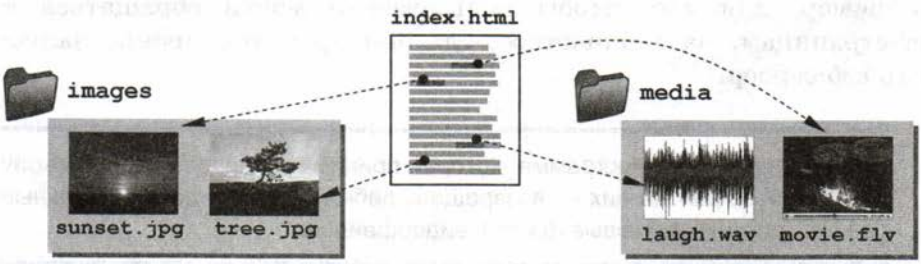


Рис. 4.1

Браузер, получив от сервера запрошенную веб-страницу, обрабатывает её текст и выводит информацию на экран в удобной для человека форме. Встретив команды для вставки дополнительных данных (например, рисунков), браузер запрашивает их с сервера. Таким образом, для полной загрузки веб-страницы может потребоваться несколько обращений в серверу.

Несмотря на существующие стандарты языка HTML, разные браузеры могут по-разному показывать одну и ту же веб-страницу. Поэтому профессиональные разработчики обязательно проверяют, чтобы сайт выглядел по возможности одинаково в разных браузерах (это свойство называют **кроссбраузерностью сайта**).

Веб-страницы можно разделить на два типа:

- статические веб-страницы (они обычно имеют расширения htm или html) хранятся на сервере в готовом виде;
- динамические веб-страницы (с расширениями php, asp, pl) — полностью или частично создаются на сервере в момент запроса.

Статические веб-страницы меньше нагружают сервер и быстрее загружаются, потому что их код полностью готов, серверу остается просто переслать его по сети. Однако они не позволяют работать с изменяющимися данными: выбрать информацию из базы данных, добавить комментарии к фотографиям, построить гостевую книгу и т. п. Кроме того, чтобы поддерживать сайт (вносить изменения в его содержание и дизайн), нужен квалифицированный работник, знающий язык HTML и способный исправлять код страниц. Статические веб-страницы можно использовать на небольших сайтах-визитках, содержимое которых изменяется только автором.

Динамические веб-страницы — это шаблоны, в которых есть программный код на специальных серверных языках — **PHP**, **ASP**, **Perl**. Когда сервер получает запрос на такую страницу, он запускает программу-интерпретатор, которая выполняет этот код. Чаще всего при этом выбирается информация из базы данных, хранящейся на сервере. С помощью программы, встроенной в динамическую страницу, можно добавлять в базу данных информацию, загруженную пользователем (рисунки, видео, комментарии). Практически все крупные сайты состоят из динамических веб-страниц.

Для управления динамическим сайтом часто применяют **систему управления содержимым** (англ. **CMS** — *Content Management System*), с помощью которой сайт могут поддерживать пользователи, не знающие языка HTML. Однако создание динамического веб-сайта — достаточно сложная задача, для решения которой нужно (кроме знания языка HTML) уметь программировать на одном из серверных языков.

Как правило, динамические сайты работают значительно медленнее, чем статические. Это связано с тем, что серверу при получении запроса необходимо обратиться к базе данных, построить запрошенную страницу в памяти и только потом переслать её по сети на компьютер клиента.

Иногда динамическими называют также веб-страницы, которые хранятся на сервере в готовом виде, но содержат программный код на специальных языках программирования (чаще всего — на **JavaScript**). Такой подход часто называют **динамическим HTML** (англ. **DHTML** — *Dynamic HTML*), его основная цель — обеспечить *интерактивность*, т. е. сделать так, чтобы веб-страница «реагировала» на действия пользователя. Код в динамических страницах такого типа выполняет браузер на компьютере-клиенте, поэтому сервер не загружается дополнительной работой.

Программа на языке JavaScript называется сценарием или скриптом.

Скрипт, или **сценарий** (англ. *script*) — это программный код для автоматизации какой-то операции пользователя.

С помощью скрипта можно изменять содержимое и оформление веб-страницы в ответ на действия пользователя:

- заменять текст, оформление, рисунки;
- строить многоуровневые выпадающие меню;

- скрывать и открывать части страницы;
- проверять данные, введённые пользователем;
- выполнять вычисления и т. д.

Вопросы и задания

1. Поясните разницу между Всемирной паутиной и Интернетом.
2. Что такое гипертекст?
3. Существовали ли гипертекстовые документы до изобретения компьютеров? Что нового внесли в эту сферу компьютеры? Подготовьте сообщение.
4. Что такое гипермедиа?
5. Какие значения имеет слово «сервер»?
6. Что такое веб-серверы? Какова их роль в работе Всемирной паутины?
7. Какие задачи решает браузер?
8. Что такое веб-сайт?
9. Что такое контент сайта?
10. Какой язык используется для описания веб-страниц?
11. Какие задачи решают веб-дизайнеры — люди, занимающиеся разработкой дизайна сайтов?
12. Обсудите с учителем и одноклассниками вопрос: «Дизайн сайта — цель или средство?»
13. Что такое тэг? Что можно сделать с помощью тэгов?
14. Как вы думаете, почему для хранения рисунков обычно выделяют отдельный каталог?

*15. Какие операции выполняет браузер при загрузке веб-страницы, адрес которой ввёл пользователь?

16. Как вы думаете, почему разные браузеры могут по-разному показывать одну и ту же веб-страницу?
17. Можно ли просматривать веб-страницу без браузера?
18. Что такое кроссбраузерность?
19. Чем отличаются статические и динамические веб-страницы? Назовите типичные расширения статических и динамических страниц.
20. В чём достоинства и недостатки статических и динамических веб-страниц?
21. Почему крупные сайты практически всегда строятся на динамических страницах?
22. Что такое CMS?
23. Что такое динамический HTML? Чем отличаются страницы этого типа от веб-страниц, написанных на языке PHP?
24. Что такое скрипт, или сценарий? Какие задачи можно решать с помощью скриптов?
25. Какой язык чаще всего применяется для создания интерактивных веб-страниц?

Подготовьте сообщение

а) «Дизайн сайта – цель или средство?»

б) «Что такое кроссбраузерность?»

в) «Современные браузеры»

§ 25 Текстовые веб-страницы

Как создать веб-страницу?

Простейший способ создать текстовую веб-страницу — это набрать её код в каком-нибудь текстовом редакторе, который работает с простым текстом без оформления («только текст», англ. *plain text*) и сохранить в файле с расширением htm или html. В операционной системе для этих расширений, как правило, установлена связь (ассоциация) с браузером, так что при двойном щелчке на имени файла веб-страница открывается в браузере в режиме просмотра.

Для изменения страницы можно использовать любой текстовый редактор. После записи изменений на диск нужно обновить страницу в браузере (чаще всего для этого используется клавиша *F5*).

Кроме того, существуют специальные редакторы, предназначенные для разработки веб-страниц. Многие из них поддерживают режим **WYSIWYG** (англ. *What You See Is What You Get* — что видишь, то и получишь), т. е. при редактировании документ выглядит так же, как при просмотре в браузере. Изменять содержание и оформление веб-страницы можно точно так же, как и в текстовых процессорах (с помощью мыши, меню и командных кнопок), при этом пользователю не нужно знать язык HTML — все необходимые команды (тэги) программа добавляет автоматически. В то же время профессиональные веб-дизайнеры редко используют WYSIWYG-редакторы, потому что для точной вёрстки (размещения материала на странице) практически всегда приходится вручную редактировать HTML-код.

Даже если редактор поддерживает режим WYSIWYG, очень полезно изучать код построенной в нём веб-страницы для того,

чтобы в будущем вы смогли легко перейти к созданию динамических сайтов.

Первая веб-страница

Для разметки веб-страницы используются специальные команды языка HTML — **тэги**. Тэги заключаются в угловые скобки, таким образом, все символы в угловых скобках считаются командами и на экран не выводятся. Неизвестные тэги браузер просто пропускает.

Простейшая веб-страница состоит из двух тэгов: начинается с открывающего тэга `<html>` и заканчивается закрывающим тэгом `</html>`. Такие пары тэгов образуют **контейнеры**: закрывающий тэг ограничивает область действия открывающего. Закрывающий тэг всегда начинается знаком «/» (он также называется «слэш», от англ. *slash*). В этой странице, кроме тэгов, нет никаких данных, поэтому если открыть её в браузере, мы увидим пустое поле:

```
<html>
</html>
```

Разберём теперь более сложный пример. Ниже мы видим код веб-страницы, а на рис. 4.2 — её вид в браузере.

```
<html>
  <head>
    <title>Первая страница</title>
  </head>
  <body>
  Привет!
  </body>
</html>
```

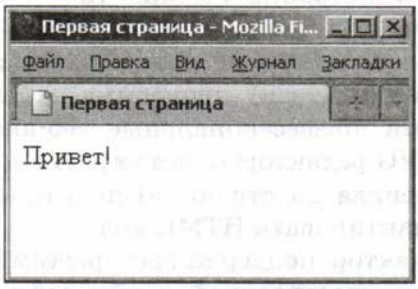


Рис. 4.2

Здесь два блока — надписи «Первая страница» и «Привет!». Посмотрев на код, можно заметить, что он разбит на две части контейнерами <head> (англ. *head* — голова) и <body> (англ. *body* — тело).

Первая часть (контейнер <head>) — это головная часть страницы. Там размещается служебная информация, например, ключевые слова и описание страницы для поисковых систем, кодировка символов и т. п. В нашем примере здесь всего один элемент — <title> (англ. *title* — название), в нём записывают название страницы, которое выводится в заголовке окна браузера (найдите его на рис. 4.2). Содержимое элемента <title> — очень важная информация для поисковых систем, поэтому нужно, чтобы этот текст как можно точнее отражал содержимое веб-страницы.

Вторая (основная) часть страницы расположена внутри контейнера <body>. В нашем случае там находится строка «Привет!», которую мы видим в окне браузера. В примерах, которые будут приводиться далее, мы будем писать только то, что содержится в контейнере <body>.

Заголовки

Для того чтобы текст лучше воспринимался, его нужно *структурировать*, т. е. разбить на разделы и подразделы, в каждом из которых рассматривается отдельный вопрос. В языке HTML есть специальные тэги для выделения заголовков разных уровней:

<h1> — заголовок документа (один на странице);
<h2> — заголовок раздела;
<h3> — заголовок подраздела;
<h4> — заголовок параграфа.

Первая буква *h* в названии тэга связана с английским словом *header* — заголовок. Вот как выглядит текст с заголовками двух уровней (рис. 4.3).

Рис. 4.3

Для этого требуется написать такой код:

```
<h1>Глава 1. Информация</h1>
<h2>1.1 Что такое информация?</h2>
Задачи, связанные с хранением, передачей и обработкой
информации, человеку приходилось решать во все
времена...
```

Обратите внимание, что мы нигде не указали, как именно оформить заголовки (какой шрифт и отступы использовать и т. п.). Браузер применяет то оформление, которое установлено в его настройках *по умолчанию* (т. е. для случая, когда оформление не задано явно). Как видно, заголовки выделяются жирным шрифтом увеличенного размера и выравниваются по левой границе. Если, например, заголовок документа нужно выровнять по центру, используют такую запись:

```
<h1 align="center">Глава 1. Информация</h1>
```

Здесь тэг `<h1>` содержит ещё дополнительное свойство – **атрибут** `align` (англ. *align* – выровнять). Атрибуты тэгов записывают в открывающем тэге внутри угловых скобок. Значение атрибута указывают после знака равенства в кавычках. Для заголовков используют три типа выравнивания:

`left` — влево (по умолчанию);
`center` — по центру;
`right` — вправо.

Абзацы

Чтобы текст (особенно длинный) было удобно читать, его разбивают на **абзацы**, каждый из которых выражает законченную мысль. Для выделения абзацев в текстовых редакторах достаточно просто нажать клавишу Enter в том месте, где заканчивается абзац. Однако для веб-страниц это правило не действует. Если набрать в контейнере `<body>` стихотворение, то мы увидим, что браузер «свернет» всё в один абзац (рис. 4.4):

```
И вечный бой! Покой нам только снится
Сквозь кровь и пыль...
Летит, летит степная кобылица
И мнёт ковыль...
```

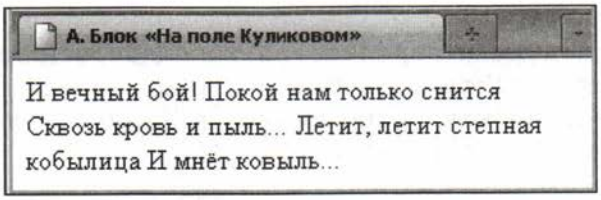


Рис. 4.4

Абзац в языке HTML выделяется контейнером `<p>` (от *англ.* paragraph — абзац) (рис. 4.5):

```
<p>И вечный бой! Покой нам только снится </p>
<p>Сквозь кровь и пыль...</p>
<p>Летит, летит степная кобылица</p>
<p>И мнёт ковыль...</p>
```

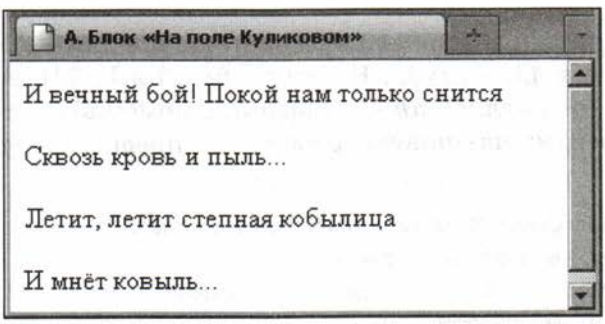


Рис. 4.5

По умолчанию абзацы отделяются друг от друга дополнительным отступом, однако абзацного отступа («красной строки») браузер не делает.

Для выравнивания абзаца в тэг `<p>` можно добавлять атрибут `align`, как для заголовков. Кроме уже упомянутых вариантов выравнивания (`left`, `center`, `right`) для текстовых колонок часто используется значение `justify` — выравнивание по ширине. Однако для узких колонок его использовать нельзя, потому что браузер слишком сильно растягивает интервалы между словами и получаются большие пробелы, затрудняющие чтение (рис. 4.6):

```
<p align="justify">
Молекула воды испарилась из кипящего чайника и,
подлетая к потолку, лоб в лоб столкнулась
с неизвестно как прокравшейся на кухню молекулой
водорода. Кто быстрей отлетел?
</p>
```

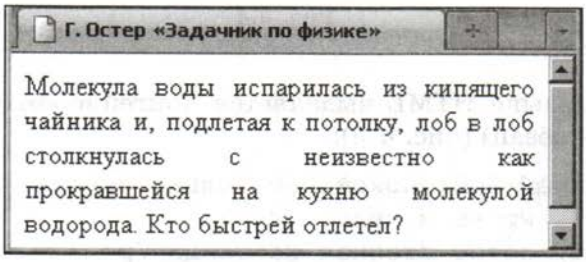


Рис. 4.6

Отступ между абзацами облегчает чтение длинных текстов, но явно не нужен для стихов. На этот случай в HTML есть еще один тэг —
 (от англ. *break* — разрыв), с помощью которого можно просто перейти на новую строку в пределах одного абзаца (рис. 4.7):

```
И вечный бой! Покой нам только снится
<br>Сквозь кровь и пыль...
<br>Летит, летит степная кобылица
<br>И мнёт ковыль...
```

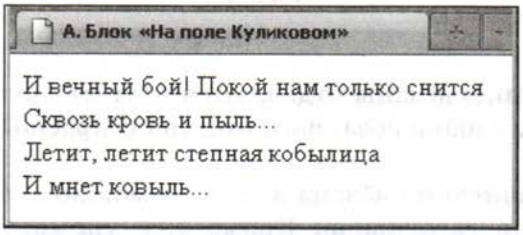


Рис. 4.7

Из кода последнего примера видно, что у тэга
 нет парного закрывающего тэга, он обозначает однократное действие — переход на новую строку.

Специальные символы

Часто нужно вставить на веб-страницу специальные символы, которых нет на клавиатуре. Наиболее интересные из них перечислены в табл. 4.1.

Таблица 4.1

Символ	HTML-код	Название
–	— или —	Тире
	¡ или	Неразрывный пробел
§	§ или §	Параграф
©	© или ©	Символ авторского права
®	® или ®	Зарегистрированная торговая марка
«	« или «	Левая русская кавычка
»	» или »	Правая русская кавычка
°	° или °	Градус
±	± или ±	Плюс-минус
^{2}	² или ²	Квадрат
^{3}	³ или ³	Куб
$^1/_4$	¼ или ¼	Четверть
$^1/_2$	½ или ½	Половина
$^3/_4$	¾ или ¾	Три четверти
×	× или ×	Знак умножения
÷	÷ или ÷	Знак деления
<	< или <	Левая угловая скобка
>	> или >	Правая угловая скобка

Неразрывный пробел используется, например, для того чтобы отделить инициалы от фамилии. Браузер не может разрывать части текста, соединённые неразрывным пробелом (в том числе переносить на новую строку). Вот несколько примеров использования неразрывного пробела и тире:

```
Дом сдали в 2011 году.
А.С. Пушкин &mdash; солнце русской поэзии.
Пёс весил 12 кг.
```

Специальные обозначения для угловых скобок введены для того, чтобы на веб-странице можно было использовать знаки «меньше» и «больше» и браузер не воспринимал бы их как ограничители тегов. Например, неравенство «$X < Y$» кодируется так:

```
X &lt; Y
```

Полный список специальных символов можно найти в Интернете по запросу *HTML entities*.

Списки

В языке HTML можно использовать два вида списков — маркированные (каждый элемент отмечен маркером) и нумерованные (с числовой или буквенной нумерацией).

Маркированный список применяется для перечисления элементов множества, когда порядок неважен. В языке HTML он строится с помощью тэга `<ul>` (от англ. *unordered list* — неупорядоченный список). Каждый элемент списка вложен в контейнер `<li>` (от англ. *list item* — элемент списка) (рис. 4.8).

```
<ul>
<li>Вася</li>
<li>Петя</li>
<li>Коля</li>
</ul>
```

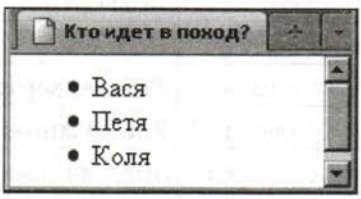


Рис. 4.8

Атрибут `type` (англ. *type* — тип) позволяет выбрать маркер (значок), например:

```
<ul type="circle">
...
</ul>
```

Существуют три стандартных типа маркеров: ● `disk` (диск, установлен по умолчанию), ○ `circle` (окружность) и ■ `square` (квадрат).

Если порядок перечисления элементов важен, применяется **нумерованный список**. В языке HTML для этой цели используют тэг `<ol>` (от англ. *ordered list* — упорядоченный список). В остальном оформление похоже на маркированный список (рис. 4.9).

```
<ol>
<li>детский сад</li>
<li>школа</li>
<li>университет</li>
</ol>
```

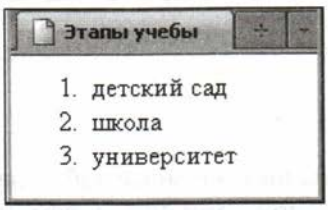


Рис. 4.9

Для настройки списка можно использовать атрибуты `type` и `start` тэга `<ol>`. Атрибут `type` определяет тип нумерации (`1` — арабские цифры, `i` и `I` — римские цифры, `a` и `A` — буквы латинского алфавита). С помощью атрибута `start` (*англ.* start — начало) задаётся начальный номер. Например:

```
<ol type="I" start="3">
...
</ol>
```

Специальных тэгов для создания многоуровневых списков в языке HTML нет, однако эти списки очень легко построить, размещая вложенные списки внутри контейнеров `<li>`, т. е. внутри элементов основного списка (рис. 4.10):

```
<ol>
<li>Россия
  <ul><li>Москва</li>
      <li>Санкт-Петербург</li>
  </ul>
</li>
<li>Украина
  <ul><li>Киев</li>
      <li>Одесса</li>
  </ul>
</li>
</ol>
```

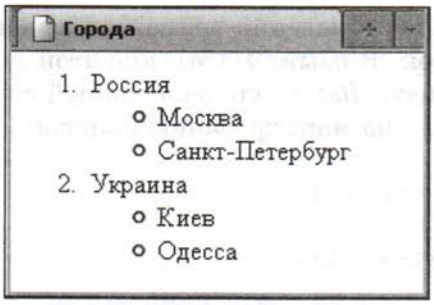


Рис. 4.10

Гиперссылки

Гиперссылка связывает элемент веб-страницы с другим объектом (документом, файлом, меткой в том же документе). Гиперссылки по умолчанию выделяются синим цветом и подчёркиваются. Цвет посещённых гиперссылок изменяется на фиолетовый (чтобы пользователь видел, где он уже был).

Для создания гиперссылки в языке HTML используют тэг `<a>` (от англ. *anchor* — якорь). Его атрибут `href` (от *англ.* hyper reference — гиперссылка) указывает на документ, который должен быть загружен после щелчка на гиперссылке. Например (рис. 4.11):

```
Переход на
<a href="newpage.html">
новую страницу</a>.
```

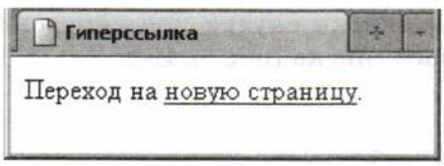


Рис. 4.11

Обратите внимание, что адрес связанного документа (newpage.html) в окне браузера не виден[1], а на экране появляется выделенное содержимое контейнера (текст «новую страницу»).

[1] Его обычно можно увидеть в строке состояния в нижней части окна браузера, если навести указатель мыши на гиперссылку.

Поскольку в атрибуте `href` указано только имя файла, а не его полный адрес, браузер будет искать документ в том же каталоге, где находится страница со ссылкой.

Ссылка на файл `info.htm` в подкаталоге `news` запишется в виде

```
<a href="news/info.htm">Информация</a>.
```

Две точки означают выход из каталога «наверх», в родительский каталог. Например:

```
<a href="../../news/info.htm">Информация</a>.
```

В этом случае для поиска файла info.htm нужно выйти на два уровня вверх по дереву каталогов и затем войти в подкаталог `news`.

Все рассмотренные выше гиперссылки — локальные (местные), т. е. они ведут к файлам, находящимся на том же сервере. Внешняя ссылка представляет собой полный адрес документа — **URL** (англ. *uniform resource locator* — универсальный указатель ресурса), который включает протокол доступа к данным, имя сервера, каталог и имя самого документа. Например:

```
<a href="http://example.net/news/info.htm">
Информация</a>.
```

Если имя каталога и файла не указано, ссылка ведёт на главную страницу сайта:

```
<a href="http://example.net/">Информация</a>.
```

Имя этой страницы зависит от настроек веб-сервера.

Ссылка может быть связана с архивным файлом:

```
<a href="http://example.net/files/game.zip">
Скачать</a>.
```

При щелчке на такой ссылке начинается скачивание файла `game.zip`.

Ссылка, адрес которой начинается с символов `mailto` (от англ. *mail to* — отправить письмо), приводит к запуску программы для отправки электронной почты:

```
<a href="mailto:vasya@mail.ru">Напишите мне!</a>.
```

С помощью гиперссылки можно перейти не только на любой доступный документ, но и в определённое место этого же или дру-

гого документа. Для этого нужная точка в документе должна быть отмечена тэгом <a> с атрибутом name (*англ.* name — имя):

```
<a name="bear"></a>
```

Обратите внимание, что в контейнере <a> ничего нет, поэтому на экране метка совершенно незаметна. Для ссылки на метку перед её именем ставится знак #, например:

```
<a href="#bear">Медведь</a>
```

или так:

```
<a href="animals.html#bear">Медведь</a>
```

Первая ссылка обращается к метке bear в текущем документе, а последняя — в файле animals.html.

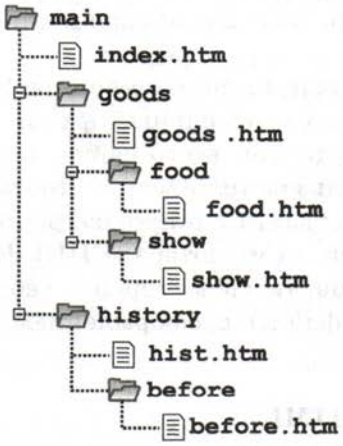
Вопросы и задания

1. Что такое WYSIWYG? Какие преимущества и недостатки имеют редакторы веб-страниц, поддерживающие этот режим?
2. Как вы думаете, почему профессиональные веб-дизайнеры редко используют WYSIWYG-редакторы?
3. Что такое контейнер? Чем различаются открывающий и закрывающий тэги контейнера?
4. Внутри какого контейнера располагается код веб-страницы?
5. Какая информация размещается в контейнерах <head> и <body>?
6. Где будет выведен текст, написанный внутри контейнера <title>? Почему он очень важен для веб-мастера?
7. Какие тэги используются для выделения заголовков?
8. Объясните, что неправильно в такой записи:
   `<h1><h2>Важный момент</h2></h1>`
9. Откуда браузер «знает», как оформлять заголовки?
10. Что такое оформление по умолчанию?
11. Как выровнять заголовок раздела по центру страницы?
12. Что такое атрибут? Где указывается значение атрибута? Надо ли ещё раз записывать атрибут в закрывающем тэге?
13. С помощью какого тэга можно разбить текст на абзацы?
14. Какой тэг можно использовать при наборе стихов для перехода на новую строку?
15. В каком случае можно использовать выравнивание по ширине? Когда оно неуместно и почему?
16. Какие тэги не являются контейнерами? Почему?
17. Что такое неразрывный пробел? Приведите примеры его использования.

*18. Найдите в Интернете правила использования дефиса, тире и знака «минус» на веб-страницах. Подготовьте сообщение.
19. Какие тэги используются для создания списков в HTML?
20. Какие параметры списков можно задать с помощью атрибутов?
21. В каких случаях лучше использовать нумерованные списки, а в каких — маркированные?
22. Как создать многоуровневый список в HTML?
23. Откуда браузер «знает», как оформлять гиперссылки?
24. Как увидеть, куда ведет ссылка, не щёлкая на ней?
25. Как создать гиперссылку на файл в подкаталоге; родительском каталоге?
26. Чем отличаются локальные и внешние гиперссылки?
27. Как создать ссылку, которая приводит к скачиванию zip-архива? Как вы думаете, как браузер определяет, что делать с документом, который указан в ссылке?
28. Как создать ссылку внутри документа?
29. Как создать ссылку на определенное место другого документа?

Задачи

1. На рисунке показана структура сайта, который содержит 6 веб-страниц. Выполните следующие задания:
   а) найдите все веб-страницы, о которых идёт речь;
   б) для каждой из этих страниц запишите HTML-код ссылок на все остальные страницы.



2. Расскажите, что произойдёт после щелчка на этих ссылках:
   а) `<a href="example.html">...</a>`
   б) `<a href="example.html#resume">...</a>`

в) `<a href="example.zip">...</a>`
г) `<a href="mailto:john@yahoo.com">...</a>`

3. Создайте веб-страницу на выбранную тему, включив в неё несколько ссылок на ресурсы Интернета. Разделите текст абзацы, используйте заголовки и списки.

§ 26 Оформление документа

Общий подход

Наверное, вы уже заметили, что в предыдущем параграфе этой главы мы говорили только о том, как структурировать текст: выделять заголовки и абзацы, строить списки, добавлять гиперссылки. Вопрос об оформлении документа (изменении внешнего вида) ещё практически вообще не рассматривался. Дело в том, что в современном веб-дизайне считается хорошим тоном *разделять содержание и оформление*.

В идеале веб-страница (файл с HTML-кодом) должна содержать только логическую разметку — определять смысловые части документа. Например, тэги заголовков и списков — это тэги логической разметки.

Всё оформление (например, размер шрифта и цвет заголовков) должно быть вынесено в отдельный файл. Во-первых, при этом можно легко менять дизайн всего сайта, поправив единственный файл. Во-вторых, пользователь может просматривать веб-страницу на разных устройствах, от широкоэкранного монитора до карманного персонального компьютера (КПК). Вместо того чтобы для каждого из них готовить свою версию веб-сайта, можно просто подключать разные файлы с оформлением для одних и тех же веб-страниц.

Средства языка HTML

Язык HTML предназначен, прежде всего, для логической разметки документа. Для этого используются специальные тэги, некоторые из них приведены в табл. 4.2.

Таблица 4.2

Стиль	Пример
Выделение (англ. *emphasize*)	`<em>Вася</em>`
Сильное выделение (англ. *strong*)	`<strong>Вася</strong>`
Программный код (англ. *code*)	`<code>a := b + c;</code>`
Определение (англ. *definition*)	`<dfn>Информация</dfn> - это...`
Цитата (англ. *citation*)	`<cite>Карету мне, карету! </cite>`
Сокращение (англ. *abbreviation*)	`<abbr>HTML</abbr>`

Именно эти тэги играют важную роль при определении места сайта в выдаче поисковых систем — с их помощью нужно отмечать **ключевые слова**, отражающие содержание страницы.

Обратите внимание, что внешний вид этих блоков никак не задаётся. Браузеры выделяют их некоторым стилем, установленным по умолчанию. Например, контейнер `<strong>` обычно выделяется жирным шрифтом, `<em>` — курсивом, `<code>` — моноширинным[1] шрифтом, а `<abbr>` — подчёркивается штриховой линией. **Веб-мастер** (разработчик сайта) может изменить это оформление с помощью стилевых файлов, о которых пойдёт речь далее.

Для оформления текстов программ используют тэг `<pre>` (англ. *preformatted* — предварительно отформатированный). В этом контейнере сохраняются все пробелы и символы перевода строки; текст обычно оформляется моноширинным шрифтом, чтобы сохранить все отступы (рис. 4.12).

```
<pre>
for i:=2 to n do
  if a[i]<a[iMin] then
    iMin:=i;
</pre>
```

[1] В моноширинном шрифте (например, в шрифте Courier New) все символы имеют одинаковую ширину.

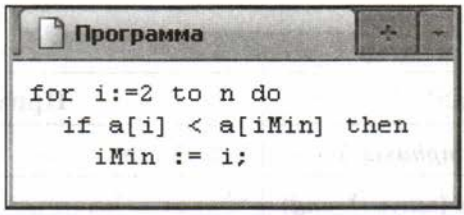


Рис. 4.12

Кроме того, язык HTML содержит так называемые **тэги физической разметки**, которые точно указывают, как должен выглядеть текст. Некоторые из них перечислены в табл. 4.3.

Таблица 4.3

Стиль оформления	Пример	Результат
Курсив (англ. *italic*)	`<i>Вася</i>`	*Вася*
Жирный (англ. *bold*)	`<b>Вася</b>`	**Вася**
Подчёркивание (англ. *underline*)	`<u>Вася</u>`	Вася
Зачёркивание (англ. *strike out*)	`<s>Вася</s>`	~~Вася~~
Верхний индекс (англ. *superscript*)	`Вася<sup>2</sup>`	$Вася^2$
Нижний индекс (англ. *subscript*)	`Вася<sub>2</sub>`	$Вася_2$

Оформление можно изменять также и с помощью атрибутов некоторых тэгов (например, атрибут `align` задаёт выравнивание заголовков и абзацев). Эти средства считаются устаревшими, и использовать их не рекомендуется. Тэги физической разметки оставлены для совместимости с веб-сайтами прошлых лет, где они были единственным средством оформления.

Стилевые файлы

Оформление современных веб-сайтов задаётся с помощью стилевых файлов, в которых в специальном формате описывается внешний вид документа. Такая технология называется **каскадными таблицами стилей** (англ. **CSS** — *Cascading Style Sheets*). Как правило, для всего сайта используется единый стилевой файл. Поэтому изменение дизайна (например, для отображения доку-

мента на карманном персональном компьютере или смартфоне) теоретически сводится к замене этого файла на другой.

Стилевой файл — это простой текстовый файл («только текст», англ. *plain text*), в котором задаются свойства тэгов. Обычно стилевые файлы имеют расширение css.

Для того чтобы изменить цвет фона и текста для всей страницы, нужно в стилевом файле (назовем его `test.css`) записать:

```
body {
  color: white;
  background: #FF6600;
}
```

Сначала указан тэг, оформление которого меняется — это тэг `<body>`. Затем в фигурных скобках определяются все нужные свойства (их называют **селекторами** от англ. *select* — выбирать), в данном случае используются свойства `color` (цвет текста, *англ.* color — цвет) и `background` (цвет фона, англ. *background* — фон). Значение свойства записывают через двоеточие, в конце каждого определения ставится точка с запятой.

Цвет можно задать двумя способами. Во-первых, некоторые «простые» цвета имеют собственные английские названия: `white` — белый, `black` — чёрный, `red` — красный, `green` — зелёный и т. д. Во-вторых, можно указать 24-битный RGB-код цвета, это позволяет закодировать 2^{24} (более 16 миллионов) цветов. Код цвета начинается знаком # и записывается в шестнадцатеричной системе счисления. Первые две цифры — это красная составляющая (R = red), следующие две — зелёная (G = green) и последние две цифры — синяя (B = blue). Таким образом, цвет с кодом `#FF6600` — это оранжевый цвет, для него

$$R = FF_{16} = 255, \quad G = 66_{16} = 102, \quad B = 0.$$

Теперь стилевой файл `test.css` нужно «подключить» к веб-странице с HTML-кодом. Для этого используется специальный тэг `<link>` (англ. *link* — связь), с помощью которого устанавливается связь с другими файлами. Этот тэг нужно расположить внутри контейнера `<head>` (заголовка страницы):

```
<head>
  <title>Страница с файлом стилей</title>
  <link rel="stylesheet" type="text/css"
  href="test.css">
</head>
```

У тэга <link> указаны три атрибута, причем сразу понятно, что href обозначает имя подключаемого стилевого файла.

Атрибут rel — это сокращение от английского relation — отношение; он определяет, какую роль играет файл text.css. В данном случае значение stylesheet говорит о том, что это таблица стилей (англ. *style sheet*). Атрибут type — это тип данных, значение text/css говорит о том, что это файл с каскадной таблицей стилей (CSS).

Если теперь вы откроете веб-страницу с подключённым стилевым файлом, то увидите, что цвета текста и фона изменились.

Стили для элементов

Задав стиль для контейнера <body>, мы определили свойства по умолчанию для вложенных в него элементов. Так в предыдущем примере для всех абзацев (<p>) будет установлен белый цвет символов. Это говорит о том, что свойство color передаётся вложенным элементам, т. е. наследуется.

Наследование означает, что некоторые свойства контейнера-«родителя» передаются всем вложенным элементам.

Это удобно, потому что во многих случаях в стилевом файле достаточно определить только свойства «родителя».

Однако не все свойства наследуются[1]. Например, фон (background) не наследуется, по умолчанию он прозрачный для всех элементов, поэтому фактически мы видим фон «родителя».

Можно задать стиль оформления любого тэга. Например, чтобы сделать рамку (англ. *border*) у всех абзацев, нужно добавить в стилевой файл строки

```
p {
  border: 1px solid blue;
}
```

Здесь для **селектора** (свойства) border задано (через пробел) сразу три «подсвойства»:

[1] В справочниках по CSS можно уточнить, наследуется ли то или другое свойство.

- `1px` — толщина линии (1 пиксель);
- `solid` — тип линии (сплошная);
- `blue` — цвет линии (синий).

Для настройки цвета гиперссылок изменяется стиль для тэга `<a>`:

```
a { color: green; }
```

Стиль посещённых ссылок задаётся с помощью обозначения `a:visited` (*англ.* visited — посещённый):

```
a:visited { color: white; }
```

Здесь `visited` — это так называемый **псевдокласс**, который обозначает состояние объекта. Часто применяется также псевдокласс `hover` (состояние «мышь над объектом»).

В рассмотренных примерах оформление определялось сразу для всех одноимённых тэгов на странице. Часто бывает нужно выделить с помощью специального оформления не все абзацы, а только некоторые. В этом случае используют **классы**. Каждому тэгу можно присвоить свой класс с помощью атрибута `class`, например:

```
<p class="error">Ошибка. Что-то с памятью.</p>
```

Специальное оформление для абзацев класса `error` определяется в стилевом файле:

```
p.error { background: red; }
```

Внешний вид остальных абзацев при этом остаётся без изменений.

Может оказаться, что к какому-то тэгу применимы несколько правил оформления, указанных в таблице стилей. Например, к абзацу класса `error` относятся все стили, определённые для тэга `<p>`, а также все стили для класса `p.error`. Такая ситуация называется **каскадом**, *поэтому таблицы стилей называют каскадными*. В этом случае браузер определяет окончательный вид элемента по специальным (достаточно сложным) правилам. Их смысл состоит в том, что более конкретное указание отменяет более общее. Например, пусть в стилевом файле определён зелёный цвет текста для всех абзацев (общее правило) и красный цвет — для абзацев класса `error` (более конкретное правило):

```
p { color: green; }
p.error { color: red; }
```

Тогда для абзаца класса `error` будет установлен красный цвет, общее правило отменяется.

Можно вообще не указывать тэг, оставив только название класса:

```
.error { color: red; }
```

В этом случае оформление применяется к *любым* тэгам, для которых задан класс `error`. Например, для выделения специальным стилем слова или словосочетания (а не всего абзаца) можно использовать тэг `<span>` (от англ. *span* — интервал, промежуток):

```
<p><span class="error">Ошибка</span>. Что-то
с памятью.</p>
```

В этом примере красным цветом будет выделено только слово «Ошибка» (для него использован класс `error`).

С помощью стилей можно определить оформление тэга, который вложен в другой тэг. Например, пусть мы хотим использовать дополнительное выделение в контейнере `<em>` с помощью такого же вложенного контейнера `<em>`:

```
<em>Тэги <em>логической</em> разметки</em>.
```

Если не изменять стили, то повторное применение тэга `<em>` не меняет результат, и слово «логической» никак не будет выделяться на фоне соседей. Если же добавить в стилевой файл строки

```
em em {
  font-style: normal;
}
```

то слово «логической» будет записано прямым (нормальным, не курсивным) шрифтом, в отличие от окружения, набранного курсивом (так обычно выделяется содержимое контейнера `<em>`). Чтобы определить курсив, нужно задать значение `italic` вместо `normal`. Запись `em em` означает «тэг `<em>` внутри другого тэга `<em>`», а `p em` — «тэг `<em>` внутри тэга `<p>`».

Существует много разных селекторов и ещё больше их возможных значений. Полную информацию вы можете найти в справочной литературе или в Интернете.

Вопросы и задания

1. Как вы понимаете термины «логическая разметка» и «физическая разметка»?
2. Почему считается более грамотным выносить оформление веб-страниц в отдельный файл?
3. Назовите тэги логической и физической разметки языка HTML. Какие из них рекомендуется использовать? Почему?
4. В чем различие тэгов и <b>? В каких случаях используется каждый из них?
5. Какой тэг используется для оформления текстов программ?
6. Что такое CSS?
7. Какое расширение обычно имеют стилевые файлы? В каких программах их можно редактировать?
8. Как подключить стилевой файл к веб-странице?
9. Как изменить оформление стандартных тэгов, например <h1>?
10. Каким способом можно определить цвет элемента на веб-странице?
11. Зачем используют классы?
12. Чем отличается класс от псевдокласса? Какие псевдоклассы вы знаете?
13. Чем различаются два определения?
    а) `p.qq { color:white; }`
    б) `.qq { color:white; }`
14. Зачем нужен тэг ?
15. Что такое наследование свойств в CSS? Чем, по вашему мнению, оно полезно?
16. Почему таблицы стилей называют каскадными?
17. Что означают следующие определения?
    а) `p span { font-style: italic; }`
    б) `p.qq span { font-style: italic; }`
    в) `p span.qq { font-style: italic; }`
    г) `p.qq span.zz { font-style: italic; }`
    д) `p span em { color: green; }`

Подготовьте сообщение

а) «Оформление вложенных элементов в CSS»

б) «Свойства символов и абзацев в CSS»

в) «Классы и псевдоклассы в CSS»

Задачи

1. Выясните экспериментально, в чём различие между тэгами <code> и <pre>.

2. Найдите в Интернете таблицу с названиями цветов, которые понимают браузеры.
3. Переопределите оформление для тэгов <dfn> и <abbr> и составьте небольшой текст, использующий эти тэги.

§ 27 Рисунки

Форматы рисунков

Рисунки хранятся на веб-серверах в виде отдельных файлов и подключаются к веб-странице с помощью тэга <img>. Браузеры умеют работать с тремя **форматами растровых рисунков:** GIF, JPEG и PNG.

Файлы с рисунками в **формате GIF** (англ. *Graphic Interchange Format* — формат для обмена графикой) обычно имеют расширение gif. Этот формат использует один из наиболее эффективных алгоритмов сжатия информации без потерь — **LZW-алгоритм** и позволяет хранить *изображения с палитрой* (*до 256 цветов*), в том числе и анимированные. Некоторые области рисунка можно сделать прозрачными, через них будет виден фон веб-страницы. Рисунки в формате GIF применяются для кодирования чётких изображений (схем, чертежей) или мелких деталей, имеющих не более 256 цветов.

Рисунки в **формате JPEG** (англ. *Joint Photographer Expert Group* — объединённая группа экспертов по фотографии) хранятся в файлах с расширениями jpg или jpeg. Этот формат предусматривает сжатие с потерями, при котором теряются (размываются) некоторые мелкие детали. В файле хранится *полноцветное изображение* (не использующее палитру), глубина кодирования цвета — 24 бита на пиксель, что даёт возможность задавать более 16 млн цветов (режим **True Color** — «истинный цвет»). Формат JPEG применяется для хранения изображений с нечёткими границами, например фотографий.

Формат PNG (англ. *Portable Network Graphic* — переносимая сетевая графика) был создан для улучшения и замены формата GIF, поскольку он, в отличие от GIF, не требует лицензии на использование. Файлы имеют расширение png и могут хранить разные типы изображений:

- *с палитрой* (до 256 цветов);
- *полноцветные* с глубиной цвета 24 или 48 битов на пиксель.

В отличие от других форматов формат PNG поддерживает частичную прозрачность пикселей, что позволяет создавать многослойные изображения на веб-страницах. Для хранения степени прозрачности выделяется отдельный канал (так называемый *альфа-канал*, *англ.* alpha channel) — дополнительно 8 или 16 битов на пиксель. Такой формат называется RGBA: «Red — Green — Blue — Alpha». Главный недостаток формата PNG — низкая эффективность сжатия маленьких рисунков в сравнении с форматом GIF.

Для добавления **векторных рисунков** на веб-страницу применяют **формат SVG** (англ. *Scalable Vector Graphics* — масштабируемая векторная графика), который умеют обрабатывать все современные браузеры[1]. Формат SVG можно использовать для хранения смешанных векторно-растровых изображений, а также анимированных рисунков. Большинство современных векторных редакторов (Inkscape, OpenOffice.org Draw, Adobe Illustrator, CorelDraw) может сохранять изображения в формате SVG.

Рисунки в документе

Для вставки изображения в код веб-страницы используется непарный тэг `<img>`. Его атрибут `src` (от *англ.* source — источник) задаёт имя файла, например:

```
<img src="night.jpg">
```

Этот файл браузер будет искать в том же каталоге, где находится веб-страница. Можно, как и при создании гиперссылок, указывать относительный путь к файлу:

```
<img src="../../images/night.jpg">
```

или даже загружать картинку с другого веб-сервера:

```
<img src="http://example.com/images/night.jpg">
```

По умолчанию рисунок вставляется прямо в текст, как одна «большая буква». Для того чтобы установить обтекание текстом, используют атрибут `align` (англ. *align* — выравнивание):

```
<img src="night.jpg" align="left">
```

Значения `left` и `right` этого атрибута задают соответственно выравнивание влево и вправо (рисунок прижимается к краю текста) (рис. 4.13).

[1] Для Internet Explorer нужно установить специальное дополнение (плагин).

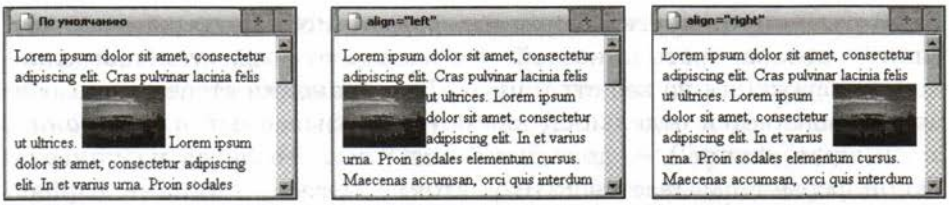


Рис. 4.13

Вариант с выравниванием влево (средний рисунок) смотрится плохо, потому что фотография «стучится» о соседний текст, подходит к нему вплотную. Чтобы этого не происходило, можно увеличить отступы, которые задаются с помощью атрибутов `hspace` (англ. *horizontal space* — горизонтальный отступ) и `vspace` (англ. *vertical space* — вертикальный отступ):

```
<img src="night.jpg" align="left" hspace="10"
vspace="10">
```

Значения отступов указаны в пикселях. Результат показан на рис. 4.14.

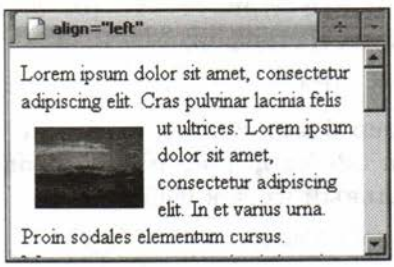


Рис. 4.14

Браузер может определить размеры рисунка только тогда, когда он загрузит его с сервера. Поэтому сначала вместо рисунка на веб-странице появляется небольшой прямоугольник, который потом заменяется изображением. При этом размеры поля, отведённого рисунку, меняются, остальной материал на экране тоже сдвигается, что неудобно для пользователя. Однако есть простое решение этой проблемы — заранее указать браузеру размеры рисунка (атрибуты `width` — ширина и `height` — высота):

```
<img src="night.jpg" width="800" height="600"
alt="Ночь на Ладоге">
```

Атрибут `alt` (англ. *alternative* — альтернативный, замещающий) задаёт текст, который показывается на месте рисунка, пока тот ещё не загружен. Кроме того, значение атрибута `alt` учитывают поисковые системы.

Рисунки очень часто служат гиперссылками. Для этого тэг `<img>` нужно поместить внутрь контейнера-ссылки `<a>`, например:

```
<a href="gallery.htm"><img src="night.jpg"
border="0"></a>
```

Атрибут `border` (англ. *border* — граница) установлен равным нулю для того, чтобы убрать синюю рамку, которая появляется по умолчанию вокруг рисунка-ссылки.

Векторные рисунки вставляются с помощью тэга `<object>`, например:

```
<object type="image/svg+xml" data="test.svg"
        width="48" height="48" align="left"></object>
```

Этот тэг используется для добавления на веб-страницу «нестандартных» данных. Атрибут `type` определяет тип данных (рисунок в формате SVG), атрибут `data` — имя файла, остальные атрибуты такие же, как у тэга `<img>`.

Фоновые рисунки

Для любого элемента веб-страницы можно определить фоновый рисунок. Для этого к свойствам тэга в стилевом файле нужно добавить селектор `background` (фон), значение которого задаётся в виде `url(адрес файла)`. В этом примере определяется фон для всего тела страницы (рис. 4.15).

```
body {
  background: url(grad.jpg);
}
```

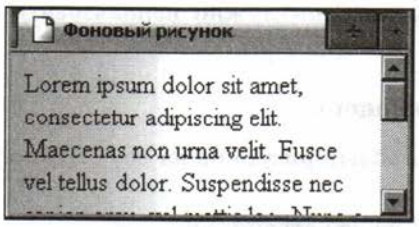


Рис. 4.15

Если рисунок меньше, чем видимая часть элемента, он повторяется. Повторение можно отменить, добавив свойство `no-repeat`:

```
body {
  background: url(grad.jpg) no-repeat;
}
```

Если нужно оставить повторение только по горизонтали, вместо `no-repeat` указывают `repeat-x`, а если задать значение `repeat-y`, то сохраняется только повторение по вертикали.

Фоновые рисунки нужно применять с большой осторожностью. Дело в том, что разнородный фон всегда мешает читать текст, т. е. затрудняет выполнение основной задачи пользователя — получение полезной информации. Поэтому профессиональные дизайнеры используют фоновые рисунки чаще всего там, где нет текста (например, для вставки логотипа).

Вопросы и задания

1. Какие форматы рисунков можно использовать на веб-страницах? Сравните их свойства.
2. Почему изображения с чёткими границами не хранят в формате JPEG?
3. Почему фотографии не хранят в формате GIF?
4. В каких форматах можно хранить изображения с прозрачными и частично прозрачными областями?
5. Что представляет собой формат PNG?
6. Какие атрибуты тэга `<img>` вы знаете? Что они обозначают?
7. Где браузер будет искать этот рисунок?
   `<img src="../../images/night.jpg">`
8. Почему в тэге `<img>` рекомендуется указывать размеры рисунка и атрибут `alt`?
9. Как добавить на веб-страницу рисунок-гиперссылку?
10. Какой формат векторных рисунков можно использовать на веб-страницах? Какой тэг применяется для вставки таких рисунков?
11. Почему фоновые рисунки нужно использовать с большой осторожностью?

Подготовьте сообщение

а) «Форматы растровых рисунков на веб-страницах»

б) «Формат PNG»

в) «SVG-графика на веб-страницах»

§ 28 Мультимедиа

Браузер, как правило, сам не может проигрывать звуковые файлы, видеофайлы, анимацию, т. е. то, что обычно называют **мультимедиа** (от лат. *multum media* — много средств). Для этого нужно подключить к браузеру специальный модуль, который называется **плагин** (от англ. *plug-in* — подключить).

Для загрузки произвольных данных на веб-страницу в языке HTML используется тэг `<embed>`. Например, для подключения звукового файла `myaw.wav` нужно написать

```
<embed src="myaw.wav" autostart="false"></embed>
```

Здесь атрибут `src` указывает на звуковой файл, а атрибут `autostart` (от англ. *auto start* — автозапуск), для которого задано значение `false` (ложь), не даёт файлу проигрываться самостоятельно, без команды пользователя. Если файл найден, на экране появляется небольшое окно проигрывателя:

Пользователь может начать прослушивание файла, если он сам этого захочет. Аналогично можно подключать к веб-странице звуковые файлы в форматах MID и MP3.

При загрузке видеофайлов и флэш-анимации (файлов SWF, выполненных с помощью технологии *Adobe Flash*) указывают ширину (width) и высоту (height) области, которая отводится ролику:

```
<embed src="cube.swf" width="275" height="200">
</embed>
```

В этом примере подключается файл `cube.swf`, который будет запущен в прямоугольном окне размером 275 × 200 пикселей.

Для обмена видеофайлами в Интернете лучше преобразовывать их в формат FLV (англ. *flash video*, файлы flv), для проигрывания которых нужен только проигрыватель Adobe Flash. Вот пример подключения ролика, загруженного на сервис *YouTube* (www.youtube.com):

```
<embed src="http://www.youtube.com/v/YvLNA5OW6xZ"
       width="425" height="350">
</embed>
```

Конечно, здесь приведены только простейшие примеры. Более подробную информацию можно найти в Интернете.

Современный стандарт языка HTML рекомендует для подключения нестандартных данных использовать тег <object> вместо тега <embed>, однако пока не все браузеры умеют с ним работать. Часто рекомендуют вставлять тэг <embed> внутрь контейнера <object>.

В новую версию языка HTML, которая известна под именем **HTML5**, встроены новые стандартные средства для воспроизведения видео- и аудиофайлов — элементы <video> и <audio>. Многие возможности HTML5 уже сейчас поддерживаются современными браузерами (лучше всего — в Google Chrome, Safari, Opera).

Вопросы и задания

1. Что такое мультимедиа?
2. Что такое плагин?
3. Какие тэги используются для подключения звуковых файлов и видео к веб-странице?
4. Почему не следует автоматически запускать звук при загрузке веб-страницы?
5. Какой формат видеофайлов сейчас лучше использовать на веб-страницах? Почему?
6. Какие средства для работы с мультимедиа добавлены в HTML5?

Подготовьте сообщение

а) «Вставка звука на веб-страницу в HTML5»

б) «Вставка видео на веб-страницу в HTML5»

Задача

Подключите к своей странице:

а) найденный в Интернете звуковой файл;

б) флэш-ролик;

в) понравившийся ролик с сервера YouTube (www.youtube.com).

§ 29 Таблицы

Структура

Для вставки таблиц используется тэг <table> (*англ.* table — таблица):

```
<table border="1">
...
</table>
```

Атрибут `border`, равный в этом примере 1, определяет ширину рамки (если его не указать, по умолчанию рамка не показывается вообще).

На месте многоточия нужно определить структуру таблицы и содержание ячеек. Строки описываются сверху вниз, каждая строка заключается в контейнер `<tr>` (от англ. *table row* — строка таблицы), а каждая ячейка внутри строки — в контейнер `<td>` (от англ. *table data* — данные таблицы). Эта таблица состоит из одной строки и двух ячеек (рис. 4.16):

```
<table border="1">
<tr>
  <td>Вася</td>
  <td>Петров</td>
</tr>
</table>
```

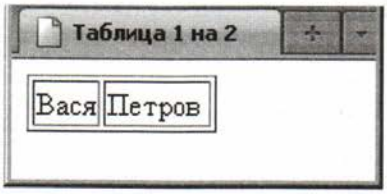


Рис. 4.16

По умолчанию размеры ячейки выбираются так, чтобы в неё поместились все данные, указанные в контейнере `<td>`. В ячейки можно вставлять не только текст, но и рисунки (с помощью тэга `<img>`).

Заголовки строк и столбцов в таблице обычно выделяют с помощью тэга `<th>` (вместо `<td>`). По умолчанию браузеры используют для таких ячеек жирный шрифт и выравнивание по центру (рис. 4.17):

```
<table border="1">
<tr><th>месяц</th></tr>
<tr><td>январь</td></tr>
<tr><td>февраль</td></tr>
<tr><td>март</td></tr>
</table>
```

Рис. 4.17

Довольно часто нужно сделать заголовок таблицы на несколько столбцов. Для этого применяют атрибут `colspan` (от англ. *column span* — охват столбцов) тэга `<th>` (или `<td>`). Значение этого атрибута — количество столбцов, охваченных заголовком (рис. 4.18):

```
<table border="1">
<tr><th colspan="3">месяц</th></tr>
<tr>
  <td>январь</td>
  <td>февраль</td>
  <td>март</td>
</tr>
</table>
```

Рис. 4.18

Для объединения строк используют аналогичный атрибут `rowspan` (от англ. *row span* — охват строк) (рис. 4.19):

```
<table border="1">
<tr>
  <th rowspan="3">Привет,</th>
  <td>Вася!</td>
</tr>
<tr><td>Петя!</td></tr>
<tr><td>Коля!</td></tr>
</table>
```

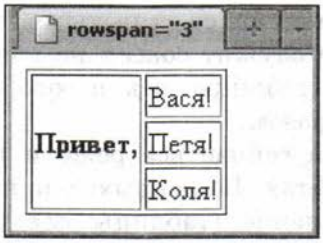


Рис. 4.19

Обратите внимание, что первая строка формально состоит из двух ячеек (включая боковой заголовок), а описание остальных строк содержит только одну ячейку.

Табличная вёрстка

В языке HTML разрешается вкладывать в ячейку одной таблицы другую таблицу. Такой приём веб-дизайнеры раньше часто использовали для **вёрстки** — размещения материала на веб-странице. Вот пример, в котором слева размещена таблица, а справа — текст (рис. 4.20):

```
<table>
<tr><td>
  <table border="1">
  <tr>
    <th>Франция</th>
    <td>Париж</td>
  </tr></table>
</td>
<td>Самая большая страна Западной
Европы.</td>
</tr></table>
```

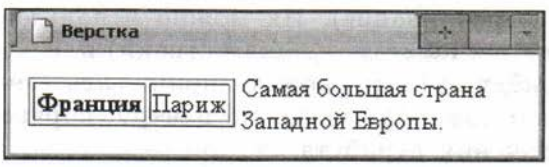


Рис. 4.20

Здесь внешняя таблица со скрытыми границами выполняет «не свою» задачу: она служит совсем не для того, чтобы представить материал в виде таблицы, а для того, чтобы разместить его на экране нужным образом.

Табличная вёрстка сейчас всё реже и реже используется на профессиональных сайтах. Во-первых, она не позволяет разделить содержание и оформление (таблицы следует скорее отнести к оформлению). Во-вторых, при табличной вёрстке страница содержит много «лишнего» (вспомогательного) кода. Вложенные друг в друга таблицы замедляют загрузку страницы. Поэтому часто вместо таблиц используют *блочную вёрстку*, о которой будет рассказано в следующем параграфе.

Тем не менее таблицы остаются самым надёжным средством создания сайтов, на которых информация размещена в несколько колонок. Особенно важно, что при этом несложно обеспечить *кроссбраузерность*, т. е. сделать так, чтобы страница выглядела одинаково в различных браузерах.

Оформление

В языке HTML есть атрибуты, позволяющие задать внешний вид таблицы. Например, в атрибутах тэга `<table>` можно задать ширину таблицы (в пикселях или процентах) и её выравнивание. Таблица, описанная как

```
<table width="50%" align="center">
...
</table>
```

всегда занимает 50% по ширине от свободного места в окне браузера и выровнена по центру. Такой дизайн называют «резиновым», потому что размеры элементов подстраиваются под размеры окна браузера. Ширину (атрибут `width`) и высоту (атрибут `height`) можно задать для каждой ячейки отдельно.

Горизонтальное и вертикальное выравнивание данных в ячейке задается атрибутами `align` и `valign` (от англ. *vertical align* — вертикальное выравнивание). Их можно использовать для тэгов `<tr>` (применяются ко всем ячейкам строки) или `<td>` (для одной ячейки). Атрибут `align` может принимать значения `left`, `center`, `right` и `justify` (влево, по центру, вправо, по ширине). Возможные значения атрибута `valign` — `top`, `middle` и `bottom` (соответственно вверх, посередине и вниз).

Более современный подход — вынести все оформление в стилевой файл и применить каскадные таблицы стилей (CSS). Подробное описание селекторов, задающих внешний вид ячеек, строк и всей таблицы можно найти в справочниках и в Интернете.

Вопросы и задания

1. Какие тэги используются для разметки таблиц?
2. Чем различаются тэги <td> и <th>? В каких случаях они применяются?
3. Можно ли вставлять одну таблицу в ячейку другой? Когда это может понадобиться?
4. Как объединять ячейки в таблицах?
5. Что такое табличная вёрстка? Назовите её достоинства и недостатки.
6. Как можно менять оформление таблиц?
7. Что такое «резиновый» дизайн? Подумайте, когда его имеет смысл использовать, а когда — нет.
8. Подумайте, соответствуют ли таблицы принципу разделения содержания и оформления документа.

Подготовьте сообщение

а) «"Резиновый" дизайн: за и против»

б) «Оформление таблиц с помощью CSS»

Задача

Найдите в Интернете сайты, где используется «резиновый» дизайн и дизайн с фиксированными размерами. Сравните их достоинства и недостатки.

§ 30 Блоки

Что такое блоки?

Блочную вёрстку разработали для того, чтобы заменить таблицы, использовавшиеся для размещения материала, и таким образом более чётко разделить *содержание* веб-страницы (*HTML-код*) и её *оформление* (стилевой *CSS-файл*). При этом, как правило, объем веб-страницы значительно сокращается, хотя бы за счёт того, что один стилевой файл используется для всех страниц сайта.

Блок — это контейнер, который задаётся тэгом `<div>` (от англ. *division* — раздел, подразделение). Он может содержать всё, что угодно — абзацы, ссылки, рисунки, таблицы, другие (вложенные) блоки и т. д.

На экране блок изображается как прямоугольник, который имеет три характерных свойства (рис. 4.21):

- границу или рамку (англ. *border*);
- внешние поля (англ. *margin*) — пространство между блоком и окружающими его элементами;
- внутренние отступы (*англ.* padding) — пространство между границей блока (рамкой) и его содержимым.

Все эти свойства настраиваются в стилевом файле с помощью CSS.

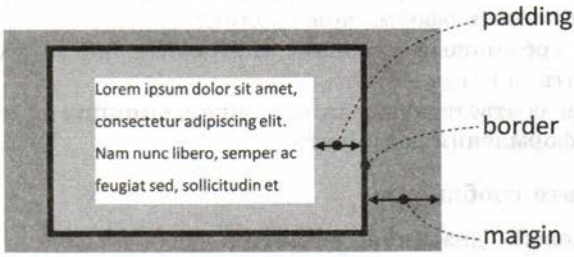


Рис. 4.21

Поскольку на странице может быть много самых разных блоков, для тэга `<div>` обычно задаётся **класс** (атрибут `class`) или **идентификатор** (атрибут `id`). Разница в том, что идентификатор — это уникальный признак (на странице не должно быть двух элементов с одинаковым идентификатором), а класс — это описание общих свойств нескольких элементов. Для блока можно задать одновременно и класс, и идентификатор, например:

```
<div class="info" id="result">
Ответ: 45.
</div>
```

Предположим, что используется такой стилевой файл (рис. 4.22):

```
.info {
  margin: 5px 5px 10px 20px;
  padding: 3px;
}
```

```
#result {
 background: #CCCCFF;
 border: 1px solid blue;
}
```

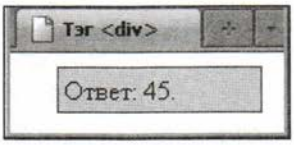


Рис. 4.22

Здесь для всех элементов (в том числе и блоков) класса `info` устанавливаются поля (`margin`) и отступы (`padding`). Поля определены отдельно для каждой стороны в таком порядке: верхнее (5 пикселей), правое (тоже 5 пикселей), нижнее (10 пикселей) и левое (20 пикселей). Отступы одинаковые, со всех сторон по 3 пикселя.

Кроме того, для элемента с идентификатором `result` установлен светло-синий фон и синяя рамка. Мы видим, что к блоку применены как свойства класса, так и свойства идентификатора.

Пример: плавающие блоки

Блоки можно делать «плавающими», они будут обтекаться текстом. В следующем примере мы построим плавающий блок, состоящий из рисунка и подписи к нему. Для него нужно установить обтекание текстом и прижать к левой границе страницы (рис. 4.23):

```
<div class="picture">
  <img src="dog.jpg"
       width="100" height="66">
  <p>На природе</p>
</div>
```

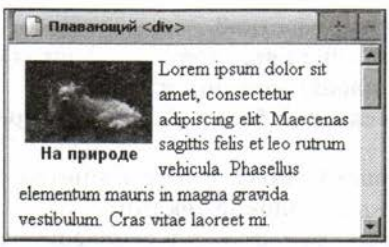


Рис. 4.23

Здесь используется класс, а не идентификатор, потому что на странице может быть несколько иллюстраций. В стилевом файле определяются свойства самого блока (класс `picture`) и свойство абзаца, в котором расположена подпись (`.picture p`):

```
.picture { float:left; margin: 5px; }
.picture p {
  margin: 0;
  text-align: center;
  font-family: sans-serif;
  font-size: 80%;
  font-weight: bold;
}
```

Для элементов класса picture заданы поля 5 пикселей и обтекание текстом: свойству `float` (от англ. *float* — плавать) присвоено значение `left` (прижать влево).

Для абзаца внутри элемента класса `picture` установлены некоторые свойства, с которыми мы раньше не встречались:

- `text-align` — выравнивание текста (`left`, `center`, `right` или `justify`);
- `font-family` — шрифт; здесь можно указать название шрифта (например, Helvetica) или его тип (`sans-serif` — шрифт без засечек; `serif` — шрифт с засечками; `monospace` — моноширинный шрифт);
- `font-size` — размер шрифта (можно задавать по-разному, здесь — 80% от размера шрифта «родительского» элемента, в котором находится блок);
- `font-weight` — насыщенность, здесь установлен жирный шрифт.

Вопросы и задания

1. Зачем нужна блочная вёрстка?
2. Что такое блоки? Какие характерные свойства они имеют?
3. Чем различаются идентификатор и класс?
4. Что происходит, если для блока задать одновременно и идентификатор, и класс?
5. Что такое плавающий блок? Какое свойство (селектор) в стилевом файле задаёт обтекание блоков текстом?
6. Что означает запись `.picture p` в стилевом файле?
7. Какие новые свойства элементов вы узнали?

Подготовьте сообщение

а) «Блочная вёрстка: плюсы и минусы»

б) «Многоколоночная блочная вёрстка»

в) «Что такое адаптивный дизайн?»

Задача

*Используя материалы Интернета, примените блочную вёрстку для размещения содержания страницы в две колонки (в три колонки).

§ 31 XML и XHTML

Что такое XML?

В современных информационных системах (в том числе и в сети Интернет) обмен данными и их обработка очень часто выполняются автоматически, без участия человека.

До недавнего времени для передачи данных использовались в основном *двоичные форматы*. Это значит, что данные представляются в том же виде, в котором они хранятся в памяти компьютера. Вспомним, что любой двоичный код — это просто последовательность битов, и невозможно сказать, что он означает — код программы, текст, рисунок или звук. Поэтому при передаче двоичных данных приёмник должен заранее знать их формат, т. е. такой подход не универсален.

Для обмена гипертекстовой информацией был разработан язык HTML. Поскольку некоторое время назад этот язык был единственным средством разработки веб-страниц, в него были включены тэги физической разметки (определяющие оформление), которые совсем не нужны для машинной обработки. Вместе с тем пользователь не может ввести в язык новые тэги, т. е. HTML непригоден для передачи произвольных данных.

В конце XX века был предложен новый язык описания данных, который во многом снял проблему совместимости различных информационных систем. Он получил название **XML** (от англ. *eXtensible Markup Language* — **расширяемый язык разметки**). Вот пример XML-файла:

```
<?xml version="1.0"?>
<компьютер>
  <процессор частота="2 ГГц">Intel Celeron</процессор>
  <память фирма="Kingston">2048 Мб</память>
  <винчестер фирма="Seagate">320 Гб</винчестер>
  <периферия>
    <монитор>Philips 190C1SB</монитор>
    <клавиатура>Logitech Classic 200</клавиатура>
    <мышь>Genius Navigator 600</мышь>
  </периферия>
</компьютер>
```

Первая строчка говорит о том, что это XML-документ версии 1.0. Оставшаяся часть очень напоминает HTML-код:

- для разметки используются тэги-контейнеры в угловых скобках; закрывающий тэг начинается знаком «/»;
- в один контейнер могут быть вложены другие, это позволяет хранить данные в виде многоуровневой структуры (дерева);
- тэги могут иметь атрибуты, в которых хранятся дополнительные данные.

Фактически XML-файлы — это иерархические базы данных, хранящиеся в текстовом формате. В отличие от HTML здесь можно вводить свои собственные тэги и атрибуты, но никаких средств оформления нет. Для того чтобы вывести данные из XML-файла на экран в нужном виде, используют специальные стилевые таблицы, построенные с помощью языка **XSL** (англ. *eXtensible Stylesheet Language* — **расширяемый язык таблиц стилей**).

XML обладает несомненными *достоинствами* с точки зрения автоматизации обработки данных:

- открытый текстовый формат, не зависящий от типа компьютера и операционной системы;
- построен на строгих правилах, определённых стандартами, поэтому несложно написать программу для обработки XML-файлов и проверки их правильности;
- удобен для представления многоуровневых списков и иерархических баз данных.

В то же время нужно отметить и *недостатки* XML:

- достаточно сложно описать структуры данных, отличающиеся от иерархии, например графы;

- не различаются типы данных (число, текст, дата, время, логическое значение);
- XML-файлы избыточны, т. е. занимают в несколько раз больше места, чем те же самые данные в двоичном виде.

Сейчас язык XML широко используется в программном обеспечении. Например, последние версии офисных пакетов Microsoft Office и OpenOffice.org сохраняют документы как набор XML-файлов, упакованных в zip-архив. На XML основаны форматы RSS (ленты новостей на сайтах и в блогах), MathML (описание математических формул), SVG (векторная графика на веб-страницах).

XHTML

В начале XXI века появилась идея «упорядочить» язык разметки веб-страниц для того, чтобы их было удобно обрабатывать в автоматическом режиме (например, поисковым системам). Используемый повсеместно язык HTML для этого не совсем строг, например в нём можно не закрывать некоторые тэги (<p>, <li> и др.), записывать тэги и заглавными, и строчными буквами и т. п.

Новый язык, построенный на основе XML, получил название **XHTML** (англ. *eXtensible Hypertext Markup Language* — **расширяемый язык разметки гипертекста**). Он во многом похож на HTML, но имеет некоторые существенные отличия, например:

- имена тэгов и атрибутов записываются только строчными буквами;
- все тэги должны быть закрыты; тэги, не имеющие закрывающего тэга (например, <img> или
), должны закрываться слэшем (например,
);
- должна соблюдаться правильная вложенность тэгов; например, этот код неправильный:
  ```
  <p><strong>Вася</p></strong>
  ```
- значения атрибутов должны обязательно заключаться в кавычки, например:
  ```
  <img src="dog.jpg" alt="Пёс Барбос" />.
  ```
- служебные символы должны везде заменяться на их специальные обозначения (например, знаки «<» и «&» заменяются соответственно на < и &).

Как признал в своем блоге создатель Всемирной паутины и языка HTML Тим Бернес-Ли, попытка сразу перейти на XHTML

не удалась. Поскольку этот стандарт поддерживался не всеми браузерами, многие разработчики по-прежнему использовали и используют HTML. В конце 2009 года было объявлено, что все работы, связанные с развитием XHTML, передаются группе, занятой разработкой новой версии языка HTML — HTML5, которая также будет включать XML-вариант (XHTML5).

Вопросы и задания

1. Каково основное предназначение языка XML?
2. Сравните достоинства и недостатки передачи информации с помощью двоичных форматов, HTML и XML.
3. Чем отличается XML от HTML?
4. Назовите недостатки XML. Подумайте, в каких областях этот язык удобен для применения, а в каких — нет.
5. Как определяется оформление элементов при выводе XML-файлов на экран?
6. Какие форматы хранения данных, основанные на XML, сейчас используются?
7. Чем отличается XHTML от HTML?
8. Как вы думаете, почему развитие XHTML сейчас не рассматривается как отдельное перспективное направление?

Подготовьте сообщение

а) «Язык XML: достоинства и недостатки»

б) «HTML и XHTML: сходства и различия»

в) «HTML5 — шаг вперед в развитии веб-сайтов»

§ 32 Динамический HTML

Что это такое?

HTML — это язык статической (неизменной) разметки. С его помощью невозможно, например, сделать на веб-странице выпадающее меню или популярный эффект «ролл-овер» (от англ. *roll-over*), при котором кнопка «реагирует» на наведение курсора мыши — меняет свой вид. Для того чтобы «оживить» веб-страницу, применяют так называемый динамический HTML (DHTML — Dynamic HTML).

Динамический HTML (DHTML — это технология создания интерактивных сайтов, использующая HTML, CSS, язык программирования (чаще всего JavaScript) и объектную модель документа (англ. *DOM — Document Object Model*).

Язык JavaScript — это скриптовый язык, используемый для программирования браузера. Код на JavaScript — это, как правило, набор функций, которые вызываются при наступлении некоторых событий (загрузки страницы, перемещения мыши, щелчка мышью, нажатия на клавишу и т. п.).

Согласно **объектной модели документа (DOM)**, веб-страница — это иерархия (дерево) объектов — рисунков, блоков, абзацев, ссылок и т. п. Корень этого дерева — объект document, который обозначает весь HTML-документ. На рисунке 4.24 показано дерево объектов простого HTML-документа.

```
<html>
  <head>
  <title>DOM</title>
  </head>
  <body>
  <p>Привет, <em>Вася!</em></p>
  </body>
</html>
```

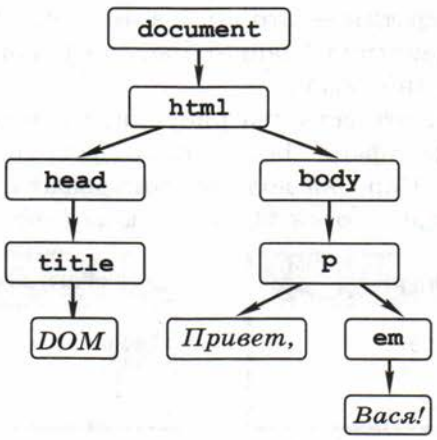


Рис. 4.24

«Сыновья» тэгов — это вложенные тэги, у документа единственный сын — тэг <html>.

С помощью JavaScript можно обратиться к любому элементу и изменить его свойства, например скрыть или показать на экране, добавить или удалить потомков. Самый простой метод поиска элемента — по идентификатору, который записывается как атрибут `id`, например,

```
<div id="menu"> … </div>
```

Есть и другие приёмы. Например, все рисунки хранятся в массиве `images` и нумеруются с нуля. Поэтому к первой картинке в документе можно обратиться как

```
document.images[0]
```

Динамическому HTML посвящены сотни книг и огромное количество веб-сайтов. Здесь мы ограничимся только рядом примеров, которые помогут вам получить представление о DHTML.

«Живой» рисунок

В первом примере мы сделаем **эффект «ролл-овер»** – изменение рисунка в тот момент, когда над ним проходит курсор мыши. Для этого нужно определить реакцию на два события:

- курсор мыши вошёл в область рисунка (*англ.* mouse over — мышь над объектом);
- курсор мыши вышел из области рисунка (*англ.* mouse out — мышь вне объекта).

Обработчик события — это код на JavaScript, который выполняет некоторые действия (меняет рисунок), когда произошло событие, с которым он связан.

Предположим, что есть два рисунка, на которых изображены закрытая коробка (файл `box_closed.gif`) и открытая (файл `box_opened.gif`). Коробка должна «открываться» при наведении мыши и закрываться, когда мышь «ушла» (рис. 4.25).

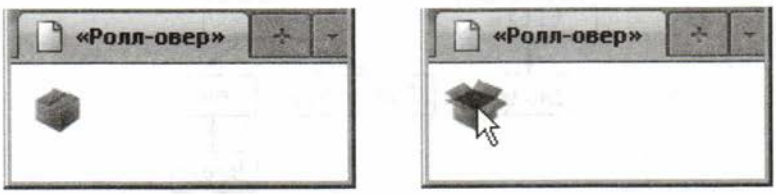


Рис. 4.25

В этом случае нужно использовать такой код вставки рисунка:

```
<img src="box_closed.gif"
     onMouseOver="this.src='box_opened.gif'"
     onMouseOut="this.src='box_closed.gif'">
```

Атрибут `src` содержит адрес рисунка, который появляется при загрузке страницы (закрытая коробка). Атрибут `onMouseOver` задаёт обработчик события «мышь над объектом», при этом выполняется код на JavaScript:

```
this.src='box_opened.gif'
```

Здесь `this` означает «этот элемент», т. е. текущий тэг `<img>`. Тогда `this.src` — это его атрибут `src`, так что приведённая строчка меняет рисунок при наведении мыши.

Обратите внимание, что в JavaScript можно использовать как одинарные, так и двойные кавычки. В данном случае одинарные кавычки (апострофы) использованы внутри двойных: двойные ограничивают весь код на JavaScript, а одинарные — имя файла. Обработчик события «мышь ушла с объекта» (`onMouseOut`) строится аналогично. Если вставить такой рисунок в контейнер `<a>`, он будет служить гиперссылкой.

Скрытый блок

Иногда на веб-странице нужно сделать скрытые элементы с дополнительной информацией, которые должны появляется только после щелчка мышью на какой-нибудь ссылке. На рисунке 4.26 показана такая страница до и после щелчка.

Рис. 4.26

Данные, которые должны быть скрыты, поместим в отдельный блок (кроме текста он может содержать рисунки, таблицы и другие элементы):

```
<div id="details" class="hidden">
Детали &#151; это гайка, шайба, болт, винт, шуруп,
гвоздь и др.
</div>
```

Это блок имеет идентификатор details, по которому мы будем к нему обращаться из программного кода. Кроме того, он относится к классу hidden (*англ.* hidden — скрытый). Заметим, что на странице можно сделать несколько скрытых блоков — для них нужно указать класс hidden, однако все они должны иметь разные идентификаторы.

Теперь нужно сообщить браузеру, что элементы класса hidden должны быть скрытыми. Для этого в стилевом файле (назовем его test.css) установим значение none (англ. *none* — никакой) свойству display (англ. *display* — показывать):

```
.hidden {
 display:none;
}
```

Мы будем с помощью JavaScript менять это свойство после щелчка на ссылке, устанавливая его значение block (блок начинается с новой строки) или inline (в той же строке, без перехода на новую строку).

Как правило, веб-мастера выносят весь код на JavaScript в отдельные файлы, которые обычно имеют расширение js и подключаются на всех страницах сайта с помощь тэга <script> в заголовке страницы. Например, для использования JavaScript-файла test.js нужно записать:

```
<head>
 <script type="text/javascript" src="test.js"></script>
  ...
</head>
```

Атрибут src задает имя файла, а атрибут type — его тип. Многоточие обозначает другие тэги, помещённые внутрь контейнера <head>.

Файл test.js можно создать в любом текстовом редакторе:

```
function show (name)
{
  var elem = document.getElementById(name);
  if (elem)
    elem.style.display = "block";
}
```

Этот файл содержит одну функцию с именем show, которая принимает один параметр name — идентификатор (имя) скрытого

элемента. Тело функции ограничено фигурными скобками, синтаксис языка очень похож на язык Си. В первой строчке создаётся переменная elem, в которую записывается ссылка на нужный элемент (для её получения используется метод getElementById).

Далее с помощью условного оператора проверяем, в самом ли деле такой элемент найден. Если значение elem не равно пустому значению (null), условный оператор срабатывает и меняется стиль элемента (style) — свойству display присваивается значение block.

Осталось оформить гиперссылку так, чтобы при щелчке мышью не происходил переход на другую страницу, а выполнялся код функции show, который показывает блок с идентификатором details. Для этого используем такой код:

```
<a href="#" onClick="show('details');return false;">
Показать детали
</a>
```

Атрибут onClick подключает обработчик события «щелчок мышью» (*англ.* click). В данном случае это код

```
show('details');return false;
```

Сначала вызывается функция show (из файла test.js), которая «открывает» блок с идентификатором details. Оператор return false («вернуть результат — ложное значение») говорит о том, что больше никаких действий выполнять не нужно. Если пользователь отключил выполнение JavaScript в браузере, сработает гиперссылка (href="#") и загрузится та же самая страница (блок, естественно, не появится).

Итак, для работы этого примера нужны три файла: test.css, test.js и основной файл с HTML-кодом, который мы приведём полностью:

```
<html>
  <head>
  <title>Скрытый блок</title>
  <script type="text/javascript"
   src="test.js"></script>
  <link rel="stylesheet" type="text/css"
   href="test.css">
  </head>
```

```
  <body>
  <a href=# onClick="show('details');
   return false;">
  Показать детали</a>
  <div id="details" class="hidden">
  Детали &#151; это гайка, шайба, болт, винт,
   шуруп, гвоздь и др.
  </div>
  </body>
</html>
```

Формы

Форма на веб-странице — это набор элементов для ввода данных пользователем: текстовые поля, переключатели, списки, кнопки и т. д. Как правило, введённые данные отправляются на сервер, где они заносятся в базу данных или как-то иначе обрабатываются. В этом случае необходимо, чтобы на сервере была «принимающая программа», написанная на одном из серверных языков (например, на PHP). Этот материал выходит за рамки школьного курса, поэтому мы покажем простой пример применения формы для тестирования, который можно использовать без сервера.

Пользователь должен ввести ответ на вопрос в поле ввода и щёлкнуть по кнопке *Готово*, после чего он получает сообщение «Правильно!» или «Неправильно!» (рис. 4.27).

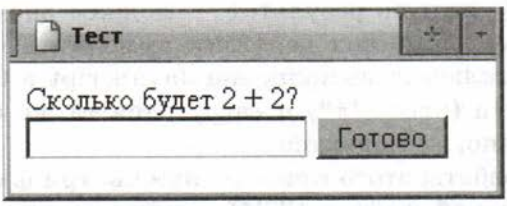


Рис. 4.27

Форма состоит из двух элементов: поля ввода и кнопки, которые помещены в контейнер `<form>` (от англ. *form* – форма):

```
<form name="calc">
  <input name="answer">
  <input type="button" value="Готово"
   onClick="check();">
</form>
```

Для формы задано имя calc (атрибут name), по этому имени можно обращаться к форме и ее элементам. Поле ввода с именем answer задаётся с помощью непарного тэга <input> (от англ. *input* — ввод).

Тот же самый тэг <input> используется и для кнопки, атрибут type задает тип элемента (англ. *button* — кнопка), атрибут value (англ. *value* — значение) — текст на кнопке, а атрибут onClick подключает обработчик события «щелчок мышью». Если вспомнить предыдущий пример, станет понятно, что после щелчка на кнопке вызывается функция check для проверки правильности ввода данных. Эта функция будет расположена в отдельном файле calc.js:

```
function check()
{
if ( calc.answer.value == "4" )
     alert("Правильно!");
else alert("Неправильно!");
}
```

Запись calc.answer.value нужно расшифровывать с конца: «значение элемента answer, который входит в форму calc». Если это значение равно 4, пользователь видит в отдельном окошке надпись «Правильно!», иначе — надпись «Неправильно!». Обратите внимание, что в JavaScript (как и в языке Си) для проверки условия «равно» используются два знака «=». Вывод сообщений здесь выполняет функция alert (англ. *alert* — тревога), которую часто применяют для отладочных сообщений.

Полный код HTML-страницы вы уже можете составить самостоятельно.

Вопросы и задания

1. Чем пользователей не устраивает «чистый» HTML?
2. Что такое динамический HTML? Какие технологии он использует?
3. Чем отличается DHTML от HTML?
4. Что представляет собой JavaScript-код?
5. Что такое DOM? Зачем нужна такая структура?
6. Как можно найти нужный элемент на веб-странице из JavaScript-кода?
7. Что такое «ролл-овер»? Приведите примеры, когда этот эффект действительно полезен.

8. Как вы понимаете слово «событие»? С какими событиями мы работали в приведённых примерах?
9. Что такое обработчик события?
10. Что означает запись `this.src`?
11. Расскажите о правилах использования кавычек в JavaScript-коде.
12. Подумайте, зачем могут понадобиться скрытые блоки на веб-странице.
13. Объясните, зачем скрытому блоку в рассмотренном примере присвоен и класс, и идентификатор.
14. Какое свойство блока нужно изменить, чтобы скрыть или показать его?
15. Как вы думаете, зачем JavaScript-код выносят в отдельные файлы?
16. Как подключаются к веб-странице файлы, содержащие JavaScript-код?
17. Как получить ссылку на элемент, идентификатор которого мы знаем?
18. Зачем в функцию `show` из рассмотренного примера добавлен условный оператор?
19. Как понимать запись `elem.style.display`?
20. Зачем в обработчик гиперссылки в рассмотренном примере добавлена команда `return false`?
21. Что такое форма? Для чего она предназначена?
22. Как можно использовать формы без «принимающей» серверной программы?
23. Зачем нужен атрибут `name` для формы и её элементов?
24. Как браузер отличает кнопку от поля ввода (оба элемента задаются тэгом `<input>`)?
25. Как задать текст на кнопке?
26. Какая функция используется для вывода сообщения пользователю в отдельном окне?

Подготовьте сообщение

а) «Какие задачи решает DHTML?»
б) «Обработка нажатий на клавиши в JavaScript»
в) «Создание тестов с помощью JavaScript»
г) «Создание меню с помощью CSS»
д) «Создание меню с помощью JavaScript»
*е) «Обработка данных форм на сервере»

Задачи

1. Протестируйте все примеры, разобранные в этом параграфе.
2. Измените пример с эффектом «ролл-овер» так, чтобы изменяющийся рисунок был ссылкой на другой документ (или сайт).

3. Проверьте, чем различаются значения `block` и `inline` для свойства `display`.

*4. Измените пример со скрытым блоком так, чтобы при повторном щелчке на той же ссылке блок скрывался.

5. Постройте страницу с несколькими скрытыми блоками, которые появляются при щелчках на разных ссылках.

§ 33 Размещение веб-сайтов

Хранение файлов

Обычно веб-сайты создаются для того, чтобы их можно было просматривать с любого компьютера, имеющего выход в Интернет. Поэтому сайт нужно размещать на компьютере, который подключён к Интернету круглые сутки.

Конечно, можно хранить сайт на диске своего домашнего компьютера, но этот вариант имеет много недостатков:

- нестабильность канала связи с Интернетом, например, при сбоях питания;
- компьютер должен быть постоянно включён;
- придется покупать у провайдера персональный IP-адрес;
- на компьютере нужно установить и настроить веб-сервер — программу, которая принимает запросы браузеров с других компьютеров и возвращает им нужные веб-страницы;
- придётся самостоятельно организовывать защиту сайта от взломщиков, вредоносных программ и сетевых атак.

Поэтому чаще всего сайты находятся на серверах компаний, которые оказывают услуги **хостинга**, т. е. размещают сайты, занимаются их обслуживанием и отвечают за сохранность данных.

Как правило, хостинг — это платная услуга, её стоимость зависит от выбранного тарифного плана. Тарифный план определяет максимально допустимый объём сайта, возможность создания динамических страниц, поддержку работы с базами данных и т. п.

Поскольку каналы связи имеют ограниченную пропускную способность, хостинговые компании вводят для каждого тарифа ограничения на количество **трафика** (англ. *traffic* — поток данных), т. е. на объём передаваемой информации. Например, если на сайт ежедневно заходят 100 пользователей и каждый просмат-

ривает 10 страниц по 100 Кбайт, дневной трафик составит около 100 Мбайт, а трафик за месяц — примерно 3 Гб. Если с сайта будут часто скачивать объёмные файлы (архивы, музыку, фильмы и т. п.), количество трафика значительно возрастёт.

В простейшем случае на одном мощном компьютере-сервере размещаются несколько сайтов (иногда — до 1000) — это **виртуальный хостинг**, подходящий для небольших сайтов.

Для сайта с большой нагрузкой у хостинговой компании обычно арендуется **выделенный сервер** — отдельный компьютер, который поступает в полное распоряжение владельца сайта. С помощью удалённого доступа можно устанавливать на нём любое нужное программное обеспечение.

Промежуточный вариант — **виртуальный частный сервер** (англ. *VPS — virtual private server*). В этом случае ресурсы одного сервера (память, время работы процессора) делятся между несколькими сайтами, но для каждого владельца сайта сервер выглядит как отдельный компьютер, который он полностью контролирует.

Существуют и **бесплатные хостинги** (например, *ucoz.ru*), где «платой» за размещение сайтов служит реклама, которая автоматически встраивается в каждую веб-страницу. Иногда бесплатный хостинг (без рекламы) предоставляется образовательным сайтам и сайтам учебных заведений.

Доменное имя

Чтобы обращаться к сайту, пользователям нужно знать его адрес. В Интернете существуют два типа адресов: числовые **IP-адреса** (например, 94.100.101.202), и символьные **доменные имена** (например, www.mail.ru). Людям неудобно запоминать IP-адреса, поэтому обычно сайты имеют доменные имена.

На некоторых сайтах доменное имя можно получить бесплатно, для этого нужно просто зарегистрироваться. Это будет домен третьего (или более высокого) уровня, например, на сайте ucoz.ru можно зарегистрировать доменное имя типа vasya.ucoz.ru — здесь первая часть (vasya) выбирается пользователем, а домены второго и первого уровней (ucoz.ru) изменить нельзя.

Регистрация доменов второго уровня, например vasya.ru, — платная. На сайтах компаний, оказывающих такие услуги (например, на сайте www.nic.ru), можно проверить, свободно ли нужное доменное имя, и оплатить его регистрацию. Для образова-

тельного сайта доменное имя второго уровня может быть предоставлено бесплатно.

После того как имя зарегистрировано, необходимо связать его с IP-адресом сервера, на котором хранятся файлы. Как вы знаете, для этого используются серверы **DNS** (англ. *domain name system* — **доменная система имён**). Владелец сайта должен сообщить регистратору домена адреса DNS-серверов хостинговой компании, и через несколько часов к сайту можно будет обращаться по доменному имени.

Загрузка файлов

Чаще всего для загрузки файлов на сайт используется **протокол FTP** (*англ.* file transfer protocol — протокол передачи файлов). Владельцу сайта выдаётся имя пользователя (логин) и пароль для входа на FTP-сервер хостинговой компании. С помощью специальных программ — **FTP-клиентов** — можно на удалённом сервере работать с файлами и папками так же, как на своем компьютере: создавать и удалять папки, скачивать файлы на свой компьютер, загружать файлы на сервер, переименовывать и удалять их. Один из популярных FTP-клиентов — кроссплатформенная свободная программа *FileZilla* (filezilla-project.org), версии которой есть для операционных систем Windows, Mac OS, Linux.

Многие HTML-редакторы, например Adobe Dreamweaver (www. adobe.com), поддерживают загрузку файлов на сайт прямо из программы по протоколу FTP.

На некоторых хостингах можно работать с файлами через веб-интерфейс, заходя на специальную веб-страницу. Однако этот способ неудобен, когда сайт состоит из большого количества страниц.

Вопросы и задания

1. Что такое хостинг?
2. Почему не имеет смысла размещать сайт на своём домашнем компьютере?
3. За счёт чего зарабатывают бесплатные хостинги?
4. Что такое тарифный план?
5. Почему вводится ограничение на трафик сайтов?
6. Как зарегистрировать доменное имя второго уровня?
7. Как связать доменное имя с сервером, на котором хранятся файлы?
8. Какими способами можно загружать файлы на сервер?

Подготовьте сообщение

а) «Как зарегистрировать доменное имя?»

б) «Сравнение бесплатных хостингов»

в) «FTP-клиенты»

Задачи

1. Сравните тарифные планы какой-нибудь хостинговой компании.
2. Ожидается, что на сайт будет заходить не более 100 посетителей в день, причем каждый из них просматривает в среднем 8 страниц. Размер одной страницы – около 50 Кбайт. Можно ли при этом выбрать тарифный план с ограничением месячного количества трафика 1 Гб?

*3. Разместите созданный вами сайт на каком-нибудь бесплатном хостинге.

*4. Научитесь пользоваться FTP-клиентом для загрузки файлов на сайт.

Практические работы к главе 4

Работа № 25 «Текстовые веб-страницы»
Работа № 26 «Списки»
Работа № 27 «Гиперссылки»
Работа № 28 «Использование CSS»
Работа № 29 «Вставка рисунков в документ»
Работа № 30 «Вставка звука и видео в документ»
Работа № 31 «Табличная верстка»
Работа № 32 «Блочная верстка»
Работа № 33 «База данных в формате XML»
Работа № 34 «Использование Javascript»
Работа № 35 «Сравнение вариантов хостинга»

ЭОР к главе 4 на сайте ФЦИОР (http://fcior.edu.ru)

- Технология создания Web-сайта
- Размещение сайта в Интернете
- Основные теги HTML

Самое важное в главе 4

- Веб-сайт — это набор связанных между собой гипертекстовых веб-страниц, размещённых на одном сервере.
- Статические веб-страницы кодируются на языке HTML. Динамические страницы содержат программы-скрипты, результат работы которых — HTML- документ, пересылаемый пользователю.
- При грамотном проектировании HTML-документ должен содержать только логическую разметку (заголовки, абзацы, списки и т. п.), а оформление страницы определяется в стилевом файле с помощью каскадных таблиц стилей.
- Для создания интерактивных сайтов применяется технология DHTML, использующая HTML, CSS, язык программирования JavaScript и объектную модель документа.
- Сайты размещаются на серверах, которые имеют собственный IP-адрес и постоянно включены. Обычно для этой цели используются услуги хостинговых компаний, обеспечивающих регулярное резервное копирование и защиту данных.

Для заметок