ФГОС

10



К. Ю. Поляков
Е. А. Еремин

ИНФОРМАТИКА

2

УГЛУБЛЕННЫЙ УРОВЕНЬ

ИЗДАТЕЛЬСТВО БИНОМ

ФГОС

К. Ю. Поляков, Е. А. Еремин

ИНФОРМАТИКА

УГЛУБЛЕННЫЙ УРОВЕНЬ

Учебник для 10 класса

в 2-х частях

Часть 2

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. Углублённый уровень : учебник для 10 класса : в 2 ч. Ч. 2 / К. Ю. Поляков, Е. А. Еремин. — М. : БИНОМ. Лаборатория знаний, 2013. — 304 с. : ил.

ISBN 978-5-9963-1417-1 (Ч. 2)
ISBN 978-5-9963-1152-1

Учебник предназначен для изучения курса информатики на углублённом уровне в 10 классах общеобразовательных учреждений. Содержание учебника опирается на изученный в 7–9 классах курс информатики для основной школы.

Рассматриваются теоретические основы информатики, аппаратное и программное обеспечение компьютера, компьютерные сети, алгоритмизация и программирование, информационная безопасность.

Учебник входит в учебно-методический комплект (УМК), включающий также учебник для 11 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.
УГЛУБЛЁННЫЙ УРОВЕНЬ
Учебник для 10 класса

В двух частях
Часть вторая

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

Подписано в печать 28.02.13. Формат 70×100/16.
Усл. печ. л. 24,70. Тираж 10 000 экз. Заказ 49.

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
Отпечатано в ООО ПФ «Полиграфист»,
160001, г. Вологда, ул. Челюскинцев, 3.
Тел.: 8(817-2) 72-61-75; 8(817-2) 72-60-63.

ISBN 978-5-9963-1417-1 (Ч. 2)
ISBN 978-5-9963-1152-1

© БИНОМ. Лаборатория знаний, 2013

Оглавление

Глава 6. Программное обеспечение 5

§ 38. Что такое программное обеспечение? 5
§ 39. Прикладные программы 7
§ 40. Системное программное обеспечение 21
§ 41. Системы программирования 35
§ 42. Инсталляция программ 43
§ 43. Правовая охрана программ и данных 47

Глава 7. Компьютерные сети 54

§ 44. Основные понятия 54
§ 45. Структура (топология) сети 59
§ 46. Локальные сети 63
§ 47. Сеть Интернет 69
§ 48. Адреса в Интернете 74
§ 49. Всемирная паутина 83
§ 50. Электронная почта 90
§ 51. Другие службы Интернета 93
§ 52. Электронная коммерция 99
§ 53. Право и этика в Интернете 103

Глава 8. Алгоритмизация и программирование 109

§ 54. Алгоритм и его свойства 109
§ 55. Простейшие программы 111
§ 56. Вычисления 118
§ 57. Ветвления 126
§ 58. Циклические алгоритмы 134
§ 59. Процедуры 144

§ 60. Функции 150
§ 61. Рекурсия 155
§ 62. Массивы 165
§ 63. Алгоритмы обработки массивов 170
§ 64. Сортировка 178
§ 65. Двоичный поиск 187
§ 66. Символьные строки 189
§ 67. Матрицы 206
§ 68. Работа с файлами 211

Глава 9. Решение вычислительных задач на компьютере . . . 223

§ 69. Точность вычислений 223
§ 70. Решение уравнений 227
§ 71. Дискретизация 240
§ 72. Оптимизация 246
§ 73. Статистические расчёты 254
§ 74. Обработка результатов эксперимента 261

Глава 10. Информационная безопасность 270

§ 75. Основные понятия 270
§ 76. Вредоносные программы 272
§ 77. Защита от вредоносных программ 279
§ 78. Шифрование 284
§ 79. Хэширование и пароли 288
§ 80. Современные алгоритмы шифрования 290
§ 81. Стеганография 294
§ 82. Безопасность в Интернете 296
Навигационные значки 300

Глава 6
Программное обеспечение

§ 38
Что такое программное обеспечение?

Чтобы компьютер можно было использовать для решения каких-либо задач, на него нужно установить **программное обеспечение** (ПО, англ. *software* — «мягкое оборудование») — программы, выполняющие ввод, обработку и вывод данных. Основное отличие компьютера от простейшего калькулятора состоит именно в том, что компьютер может выполнять введённую в него программу автоматически, без участия человека.

Обычно выделяют три вида программного обеспечения: *прикладные программы*, *системные программы* и *системы программирования* (рис. 6.1).

Всех, кто работает с компьютерами, можно разделить на *пользователей*, *системных администраторов* и *программистов* (рис. 6.2).



Рис. 6.1

Пользователи решают свои задачи с помощью **прикладных программ** (к ним относятся текстовые и графические редакторы, электронные таблицы, системы управления базами данных, программы для прослушивания музыки и просмотра видео, игры и т. п.).

Системные программы обеспечивают согласованную работу всех узлов компьютера, а также удобный *интерфейс* (способ обмена данным) между пользователем и прикладными программами, с одной стороны, и аппаратными средствами компьютера —

с другой. К этой группе относятся *операционные системы*, *драйверы* (программы для управления внешними устройствами) и *утилиты* (служебные программы). Задача **системных администраторов** — настроить системное и прикладное ПО так, чтобы пользователи смогли нормально работать.

Программисты создают новые программы с помощью **систем программирования** (инструментальных средств).

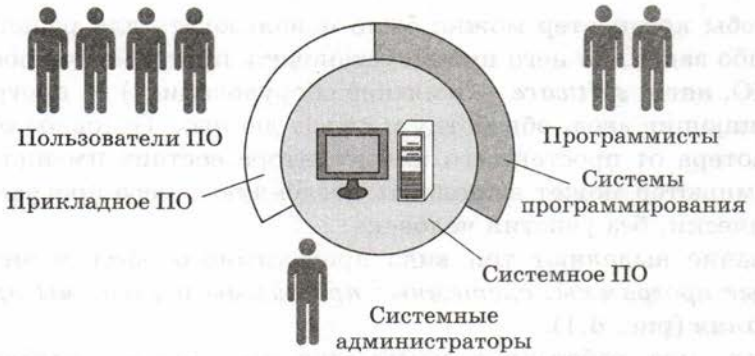


Рис. 6.2

До недавнего времени программное обеспечение было «привязано» к определённой операционной системе. Например, некоторые программы работают только под управлением Windows, а другие — только под управлением Linux. В последние годы разработано много кроссплатформенных программ, у которых есть версии для разных операционных систем.

Кроссплатформенная программа — это программа, у которой есть версии для различных операционных систем (например, Windows и Linux).

Часто термин «программное обеспечение» понимают в широком смысле как целую отрасль, включающую все этапы разработки программ, в том числе тестирование (проверку программ, поиск ошибок) и разработку документации.

Контрольные вопросы

1. Назовите три типа программного обеспечения. Чем они различаются?
2. Какие задачи решают пользователи, программисты, системные администраторы?
3. Что означает слово «интерфейс»?
4. Что такое драйверы, утилиты?
5. Что обозначают английские термины *hardware* и *software*?
6. Какое ПО называется кроссплатформенным?

§ 39
Прикладные программы

Текстовые редакторы

Многие пользователи используют компьютер, прежде всего, для работы с текстами. Обычно различают **редактирование текста** (*изменение содержания* текста: замена, вставка и удаление символов и слов; разбивка на абзацы) и **форматирование текста** (*изменение внешнего вида* текста — выбор шрифта, изменение размера, цвета и т. п.).

Простейшие программы этого класса — **текстовые редакторы** — умеют только редактировать текст. Они работают с файлами в формате «только текст» (англ. *plain text*), в которых хранятся коды символов без оформления. Современные редакторы умеют сохранять текст в разных кодировках, но чаще всего используются кодировки семейства **UNICODE**: UTF-16 (2 байта на символ для большинства символов) или UTF-8 (с переменным числом байтов на символ). Примеры текстовых редакторов:

- *Блокнот* в операционной системе Windows;
- *nano*, *gedit*, *KWrite* и *Kate* в операционной системе Linux.

На рисунке 6.3 показано окно текстового редактора KWrite.

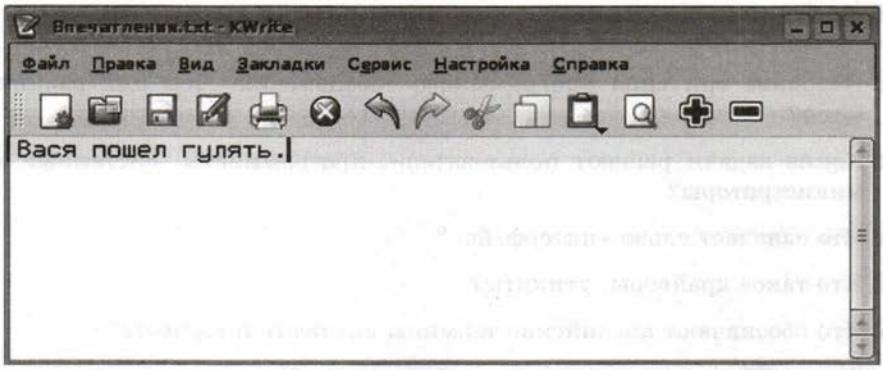


Рис. 6.3

Основные **возможности** современных **текстовых редакторов**:

- ввод и редактирование текста;
- создание, открытие, сохранение и печать документов типа «только текст»;
- работа с буфером обмена (копирование, вырезание, вставка);
- отмена последних операций;
- поиск и замена фрагментов текста;
- подсветка ключевых слов языков программирования (Си, Паскаль и др.) и языков разметки текста (XML, HTML, LaTeX);
- проверка орфографии.

Текстовые редакторы часто используются системными администраторами для редактирования файлов с настройками программ (файлов конфигурации). Тексты программ тоже хранятся в формате «только текст», поэтому программисты набирают и редактируют их в текстовых редакторах.

Офисные пакеты

Для подготовки офисных документов возможностей текстовых редакторов недостаточно. Часто нужно применять в одном документе различные шрифты, выделять фрагменты (курсивом, подчёркиванием, маркером), добавлять таблицы, графики, рисунки и т. п. Кроме того, возникают и другие задачи: выполнение табличных расчётов, подготовка презентаций для выступлений и докладов, работа с базами данных и т. п. Набор программ для подготовки электронных документов называют «офисным пакетом». В **офисный пакет** обычно включаются:

- *текстовый процессор*, который позволяет не только редактировать текст, но и оформлять его по стандартам современного делопроизводства;
- *табличный процессор* — программа для выполнения расчётов с табличными данными;
- *программа для подготовки презентаций*;
- *программа для работы с базами данных*.

Самые известные офисные пакеты — *Microsoft Office* (www.microsoft.com), *OpenOffice.org* (openoffice.org) и *WordPerfect Office* (www.corel.com). Пакеты Microsoft Office и WordPerfect Office — коммерческие, а OpenOffice.org (а также его новую версию — LibreOffice) можно установить и использовать бесплатно. Кроме того, пакет OpenOffice.org — это *кроссплатформенное ПО*: существуют его версии для операционных систем Windows и Linux.

Текстовые процессоры — это следующий шаг в развитии текстовых редакторов. На рисунке 6.4 показано окно текстового процессора *OpenOffice.org Writer*.

В состав пакета Microsoft Office входит текстовый процессор *Microsoft Word*, который считается стандартным средством для оформления офисных документов. Все современные тексто-

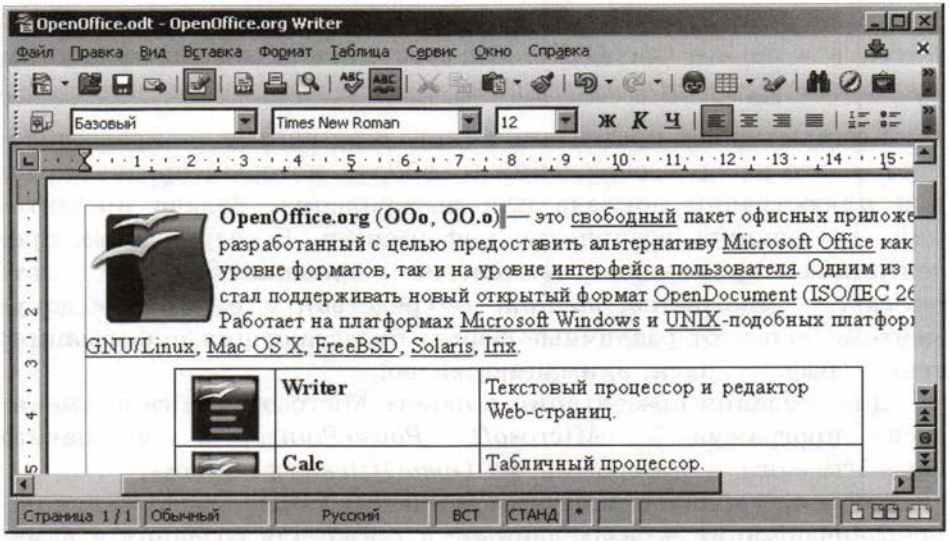


Рис. 6.4

вые процессоры позволяют сохранять документы в форматах, совместимых с Word (DOC и DOCX).

С помощью текстовых процессоров можно не только редактировать, но и форматировать текст (изменять его оформление). Кроме того, они позволяют:

- создавать составные документы, включающие списки, рисунки, таблицы, диаграммы;
- использовать стили оформления (например, заголовки разного уровня);
- использовать шаблоны (заранее оформленные заготовки) документов;
- выполнять несложные вычисления в таблицах;
- сохранять документ в разных форматах, в том числе в *HTML* (как веб-страницу) и *PDF* (англ. *Portable Document Format* — переносимый формат документов).

Табличные процессоры (электронные таблицы, англ. *spreadsheet*) — это программы для обработки табличных данных. В отличие от текстовых процессоров они не только хранят данные, но и позволяют выполнять с ними достаточно сложные вычисления, строить диаграммы, проводить анализ, делать прогнозы. Сейчас электронные таблицы — незаменимый рабочий инструмент экономистов, бухгалтеров, менеджеров. В состав пакета Microsoft Office включен табличный процессор *Microsoft Excel*, а в пакете OpenOffice.org есть близкая по возможностям программа *OpenOffice.org Calc*.

Компьютерная презентация (лат. *praesentatio* — представление) — это набор изображений (*слайдов*), который предназначен для иллюстрации доклада или выступления. Задача презентации — улучшить восприятие информации. В современных презентациях применяют технологии *мультимедиа* (от лат. *multum* — множество, *medium* — средство), т. е. в одном документе используют различные формы представления информации: текст, графику, звук, анимацию, видео.

Для создания презентаций в пакете Microsoft Office применяется программа *Microsoft PowerPoint*, а в пакете OpenOffice.org — программа *OpenOffice.org Impress*.

Система управления базами данных (СУБД) — это ПО для поиска информации в базах данных, а также для создания и изменения баз данных. В пакет Microsoft Office входит СУБД

Microsoft Access, а в пакет OpenOffice.org — СУБД *OpenOffice.org Base*.

Онлайн-офис

Бурное развитие Интернета привело к появлению **онлайн-офисов** (англ. *online* — «на линии») — специальных сайтов (интернет-сервисов), которые предоставляют основные возможности офисных пакетов: текстовый редактор, электронные таблицы, средства для создания презентаций. Для использования такой службы необходим компьютер с доступом в Интернет, причём не имеет значения, какая операционная система на нём установлена. Документы пользователей хранятся на сервере, для доступа к ним нужно зайти на сайт под своей учётной записью, которая защищена паролем. Самый известный онлайн-офис — *Google Docs* (docs.google.com).

Одно из достоинств онлайн-офисов — возможность совместной работы над документами через Интернет. Другим пользователям можно также открыть доступ к отдельным документам для просмотра и/или изменения. Любой документ может быть *экспортирован* (сохранён) в файл на диске компьютера.

Онлайн-офисы используют технологию, известную под названием **«облачные вычисления»** (англ. *cloud computing*). Ее суть в том, что пользователь размещает свои данные на серверах Интернета и не должен заботиться о способе их хранения, операционной системе и программном обеспечении. Слово «облако» — это метафора, образ достаточно сложной системы, детали работы которой знать не обязательно. Несмотря на удобства «облачных» сервисов, существуют опасения, что пользователь может потерять контроль над своими данными, и это чревато серьёзными проблемами. Например, иногда не удаётся полностью удалить данные, которые человек сам же разместил. Кроме того, возможна потеря данных и утечка информации. Поэтому документы ограниченного доступа не следует размещать на «облачных» сервисах.

Графические редакторы

Графические редакторы — это программы для создания и редактирования изображений. Изображения, хранящиеся в компьютере, делятся на растровые и векторные (см. § 16). С ними нужно работать по-разному, поэтому существуют отдельные программы для редактирования растровой и векторной графики, которые часто называют растровыми и векторными графическими редакторами.

Растровые редакторы предназначены для:

- обработки фотографий;
- подготовки изображений к печати;
- создания и редактирования изображений для веб-сайтов.

Лучшим профессиональным растровым редактором считается программа *Adobe Photoshop* (www.adobe.com). Существуют её версии для операционных систем Windows и Mac OS (для компьютеров фирмы Apple). Стандартным приложением Windows является растровый редактор *Paint*, но для сложной обработки (например, для цветокоррекции фотографий) его возможности недостаточны.

Бесплатная программа *Gimp* (gimp.org) — кроссплатформенная, она работает как в Windows, так и в Linux. На рисунке 6.5 показано окно программы Gimp.

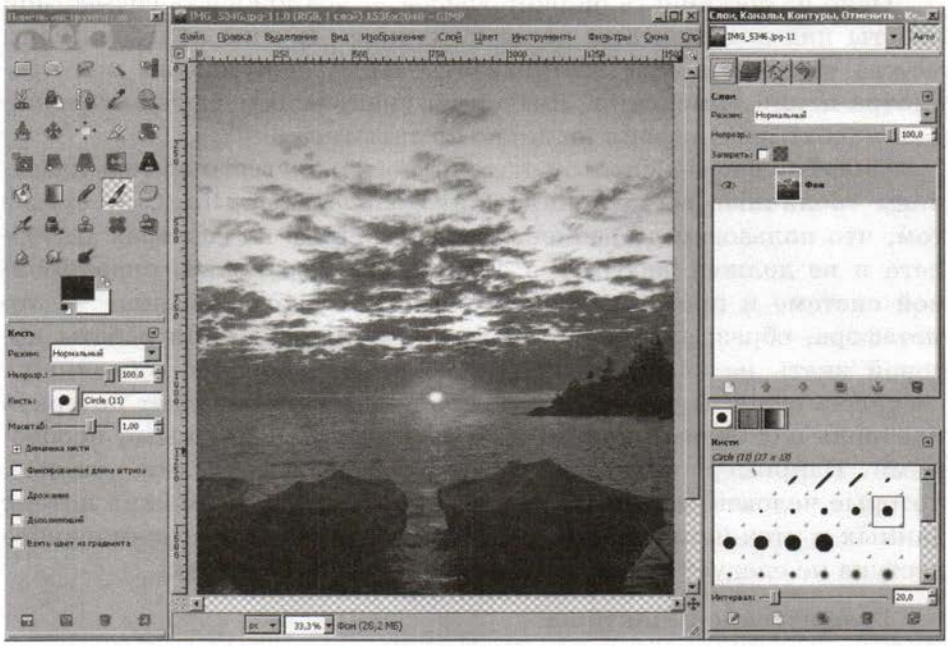


Рис. 6.5

Среди бесплатных растровых редакторов широкими возможностями обладает *Paint.NET* (www.getpaint.net), но он пока устойчиво работает только в среде Windows.

В последнее время были созданы бесплатные «онлайновые» редакторы (например, www.pixlr.com), которые позволяют обрабатывать изображения на специальной веб-странице в Интернете, без установки дополнительного ПО на компьютер пользователя.

Векторные редакторы используются для подготовки:

- художественных иллюстраций;
- технических иллюстраций (схем, графиков);
- логотипов, визиток, плакатов;
- изображений для веб-сайтов (иконок, кнопок).

Среди профессиональных векторных редакторов можно назвать двух лидеров — программы *Adobe Illustrator* (www.adobe.com) и *CorelDraw* (www.corel.com). В свободно распространяемый пакет OpenOffice.org входит векторный редактор *OpenOffice.org Draw*. Бесплатный редактор *Inkscape* (www.inkscape.org) — ещё одна кроссплатформенная программа, работающая как в Windows, так и в Linux. На рисунке 6.6 показано окно редактора Inkscape.

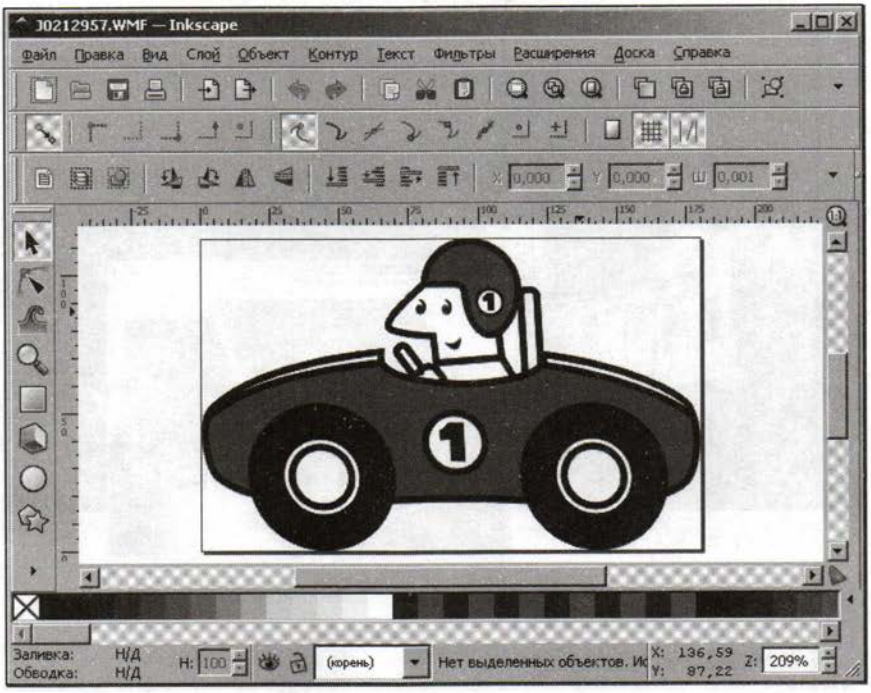


Рис. 6.6

Во многих графических редакторах, например в Adobe Photoshop и Corel Draw, можно создавать документы, содержащие как растровую, так и векторную графику. Текстовые процессоры (Microsoft Word, OpenOffice.org Writer) позволяют вставлять в документ растровые и векторные рисунки.

Во многих областях деятельности двумерных рисунков недостаточно, и необходимо представить объект в *трёхмерном пространстве*. Такие задачи возникают, прежде всего, в архитектуре, кино, телевидении и компьютерных играх. С помощью трёхмерной графики (англ. **3D** — *3 dimensions* — «3 измерения») создаются многие современные мультфильмы.

Для работы с трёхмерными объектами используют программы специального класса — **программы 3D-моделирования** (рис. 6.7), которые позволяют:

- определить форму (геометрию) объектов сцены;
- задать материалы для объектов;
- установить источники света;
- определить точки наблюдения (виртуальные камеры);
- создать анимацию с трёхмерными объектами;

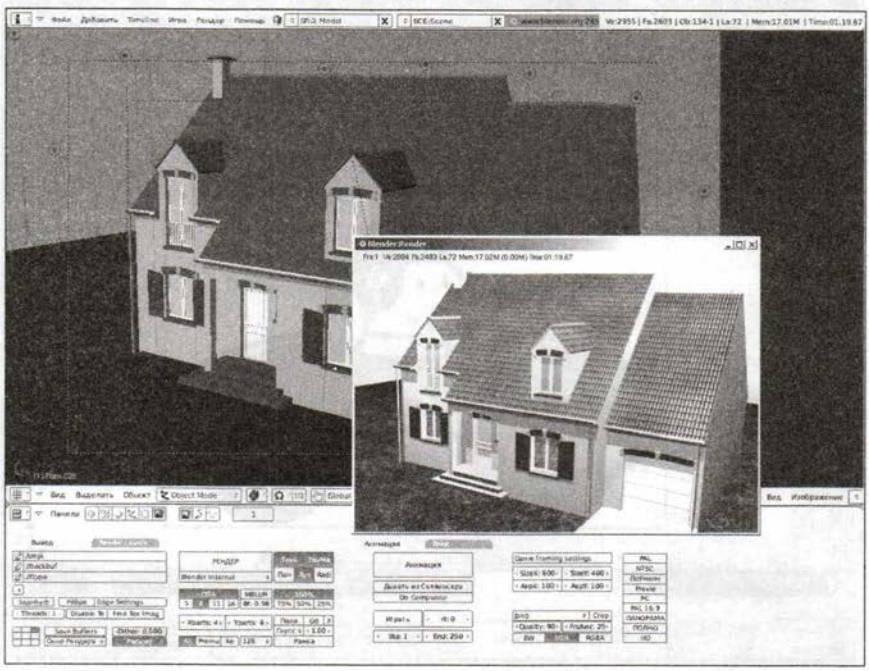


Рис. 6.7

- выполнить *рендеринг*, т. е. построить изображение трёхмерных объектов и сцены на плоскости или последовательность кадров анимации с учётом свойств объектов и источников света.

Среди программ 3D-моделирования, работающих в системе Windows, наиболее популярна *3D Studio MAX* (usa.autodesk.com). В области кино и телевидения стандартом считается кроссплатформенная программа *Maya* (www.autodesk.com/maya), у которой существуют версии для операционных систем Windows, Linux и Mac OS. Среди свободно распространяемых программ наиболее известна кроссплатформенная программа *Blender* (www.blender.org).

Алгоритмы работы с трёхмерной графикой достаточно сложны и требуют большого объёма вычислений. Поэтому для нормальной работы программ 3D-моделирования, особенно для выполнения рендеринга, требуется быстродействующий процессор и много оперативной памяти.

Настольные издательские системы

В любом современном издательстве для подготовки к печати книг и журналов используется специальное программное обеспечение — **настольные издательские системы** (англ. **DTP** — *DeskTop Publishing* — «настольное издательство»).

Основное отличие этих программ от текстовых процессоров состоит в том, что в них можно выполнять *вёрстку* — точно задавать расположение текста, рисунков, таблиц и другого материала на странице в соответствии с типографскими правилами. В настольной издательской системе готовят **оригинал-макет** (изображение, точно совпадающее с будущим отпечатком) и отправляют его в типографию.

Долгое время для подготовки оригинал-макетов использовались настольные издательские системы *QuarkXPress* (www.quark.com), Corel Ventura и Adobe PageMaker; сейчас конкуренцию им составляют *Adobe InDesign* (www.adobe.com) и бесплатная программа *Scribus* (www.scribus.net), окно которой показано на рис. 6.8.

В состав пакета Microsoft Office входит программа вёрстки *Microsoft Publisher* с несколько меньшими возможностями. Вёрстку несложных изданий (например, визиток, буклетов) можно выполнить в программе *CorelDraw*, но профессионалы не рекомендуют её использовать.

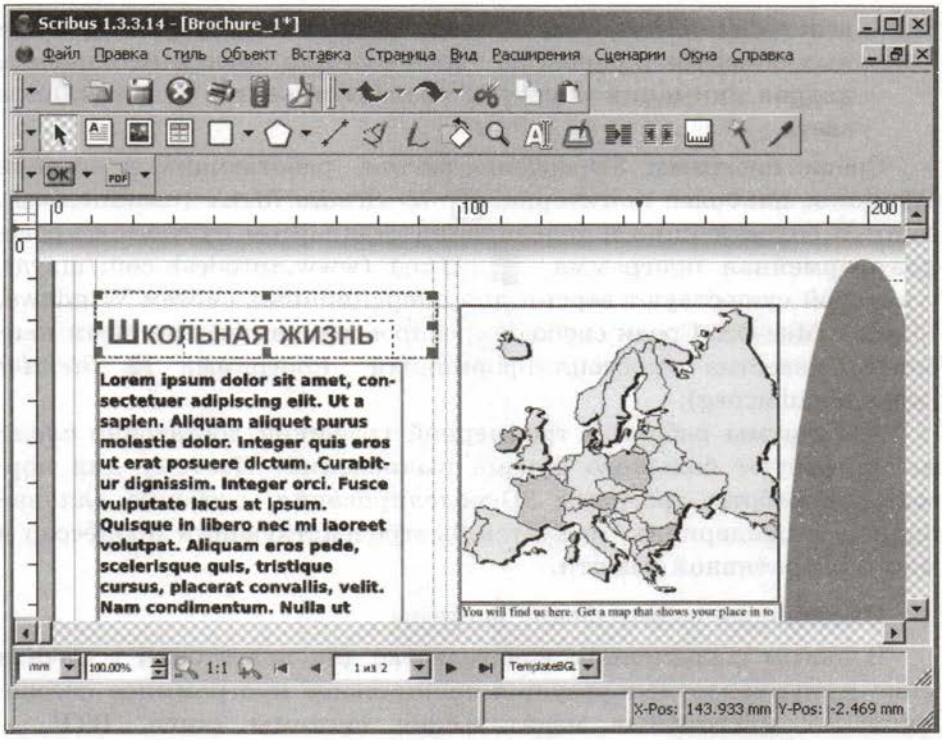


Рис. 6.8

Лучшая система подготовки математических текстов называется TEX (www.ctan.org). Её разработал известный американский математик и специалист по теоретической информатике *Д. Кнут*. Система TEX — бесплатная и кроссплатформенная, она принята в качестве стандарта во многих российских и зарубежных издательствах, выпускающих математическую литературу.

Редакторы звука и видео

Аудиоредакторы — это программы для редактирования звуковых файлов. С их помощью можно:

- загружать, редактировать и сохранять звуковые файлы разных форматов;
- записывать звук с микрофона или другого источника;
- вырезать фрагменты из файла;
- соединять звуковые фрагменты в один файл;

- изменять громкость и темп звука;
- удалять шумы.

Самые известные аудиоредакторы — это *Adobe Audition* (www.adobe.com), *Sound Forge* (www.sonycreativesoftware.com), *Audacity* (audacity.sourceforge.net). В отличие от первых двух программ Audacity — бесплатный продукт, существующий для многих операционных систем (Windows, Linux, Mac OS). На рисунке 6.9 показано окно программы Audacity.

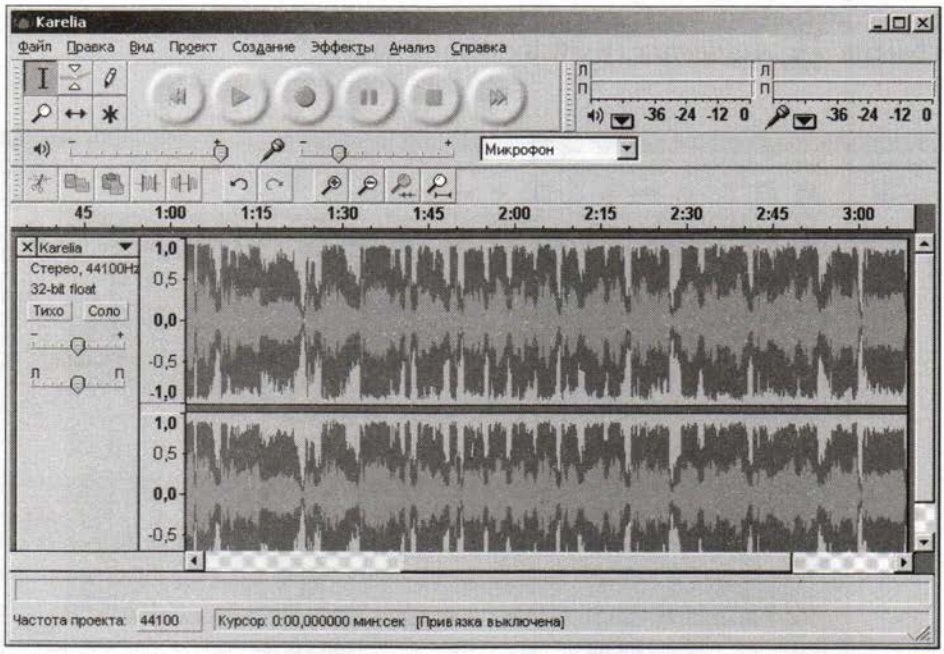


Рис. 6.9

Видеоредакторы предназначены для создания и редактирования цифрового видео. Их основные возможности:

- ввод данных с видеокамеры;
- коррекция цвета;
- добавление, перестановка, удаление фрагментов фильма;
- добавление звука и титров;
- сохранение фильма в различных цифровых видеоформатах;
- создание DVD-дисков.

Среди коммерческих программ этого типа наиболее популярны *Adobe Premier* (www.adobe.com), *Pinnacle Studio* (www.pinnaclesys.com), *VideoStudio Pro* (www.corel.com), *Sony Vegas Pro* (www.sonycreativesoftware.com).

На компьютерах фирмы Apple используется видеоредактор *iMovie* (www.apple.com).

Существуют и бесплатные видеоредакторы, например программа *Kino* для операционной системы Linux (kinodv.org), окно которой показано на рис. 6.10, программа *VirtualDub* (www.virtualdub.org) для Windows и кроссплатформенная программа *Avidemux* (www.avidemux.org).



Рис. 6.10

ПО для работы в Интернете

Для просмотра веб-страниц в Интернете нужна специальная программа, которую называют **браузером** (англ. *browser* — обозре-

ватель). Пользователи Интернета чаще всего используют следующие браузеры:

 *Internet Explorer* (стандартное приложение операционной системы Windows);

 *Firefox* (www.mozilla-russia.org);

 *Chrome* (www.google.com/chrome);

 *Safari* (www.apple.com/safari);

 *Opera* (www.opera.com).

Большинство браузеров бесплатные, многие из них кроссплатформенные, существуют их версии для большинства современных операционных систем. В состав Linux входит браузер  *Konqueror*, но пользователи чаще всего предпочитают Firefox.

Кроме браузера большинство пользователей используют **почтовые программы (почтовые клиенты)**, предназначенные для работы с **электронной почтой**. Их основные возможности:

- создание, отправка и приём сообщений;
- автоматическая проверка почты через заданный интервал времени;
- сортировка сообщений по папкам;
- ведение адресной книги (списка контактов).

В состав стандартных приложений современных версий операционной системы Windows входит почтовая программа  *Почта Windows*. Несколько бóльшими возможностями обладают профессиональные программы  *Microsoft Outlook* (входящая в пакет Microsoft Office) и  *TheBat* (www.ritlabs.com). На компьютерах фирмы Apple устанавливается почтовый клиент  *Apple Mail* (www.apple.com). В состав браузера Opera входит встроенный почтовый клиент *Opera Mail*, возможностей которого вполне достаточно для простых задач.

Самая известная из бесплатных почтовых программ —  *Mozilla Thunderbird* (www.mozilla-russia.org). Она является кроссплатформенной: существуют её версии для операционных систем Windows, Linux, Mac OS. На рисунке 6.11 показано окно программы Mozilla Thunderbird.

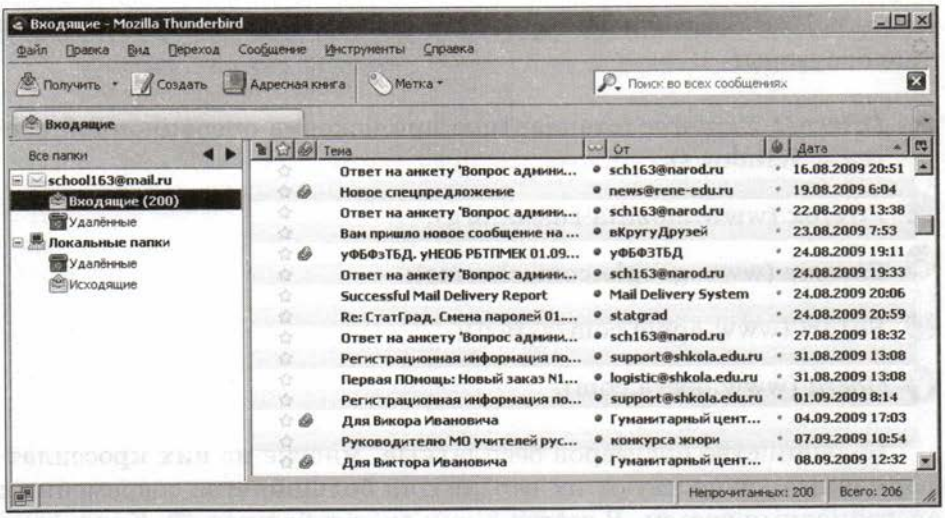


Рис. 6.11

Для **общения в реальном времени** (это значит, что собеседники одновременно находятся за компьютерами) используют программы для обмена мгновенными сообщениями (**мессенджеры**). Самые известные мессенджеры — *ICQ* (www.icq.com), *Mail.ru Агент* (www.mail.ru), *Kopete* (для Linux), *iChat* (для компьютеров фирмы Apple). В последнее время всё бóльшую популярность приобретает система мгновенного обмена сообщениями *Jabber* (www.jabber.org), также известная под названием *XMPP*. Она бесплатная, обеспечивает надёжную защиту передаваемых данных, позволяет устанавливать связь с пользователями других систем, например ICQ.

С помощью программы *Skype* (www.skype.com) можно установить через Интернет голосовую и видеосвязь между компьютерами. Для этого на каждом из компьютеров должен быть микрофон и веб-камера.

Вопросы и задания

1. Какое ПО называют прикладным?
2. Чем различаются текстовые редакторы и текстовые процессоры?
3. Что означает формат «только текст»? В каких случаях он используется?
4. Какие программы обычно входят в офисный пакет?

5. Что такое кроссплатформенное ПО?
6. Что такое СУБД? Приведите примеры известных вам СУБД.
7. Что такое «онлайн-офис»? В чём его достоинства и недостатки?
8. Что такое «облачные вычисления»?
9. Перечислите возможности растровых редакторов.
10. Для каких целей используются векторные редакторы?
11. Что такое настольная издательская система? Чем она отличается от текстового процессора? Приведите примеры настольных издательских систем.
12. Что такое оригинал-макет?
13. Какая система лучше всего подходит для набора математических текстов?
14. Какими возможностями обладают аудиоредакторы?
15. Перечислите возможности редакторов видео.
16. Что такое браузер?
17. Перечислите возможности почтовых программ.
18. Что такое мессенджер?

Подготовьте сообщение

а) «Система подготовки математических текстов LaTeX»
б) «Кроссплатформенное ПО»
в) «Онлайн-офис»
г) «Сервисы SAAS: за и против»
д) «Облачные вычисления»

§ 40
Системное программное обеспечение

Что такое операционная система?

Команды, которые может выполнять процессор, представляют собой числовые коды. Чтобы он выполнил программу, нужно эту программу загрузить в память и передать процессору адрес первой команды. В принципе это можно делать вручную, с помощью переключателей (1/0) или перфокарт, как и было на первых компьютерах. Однако в этом случае ввод программы будет занимать значительно больше времени, чем её выполнение, поэтому процессор будет простаивать. Кроме того, для ввода и вывода данных нужно программировать внешние устройства, каждое из которых имеет собственный набор команд. В таких условиях с компьютером могут работать только специально подготовленные программисты, и эта работа очень трудоёмкая. Ситуация ещё более

усложняется, если требуется записать данные на жёсткий диск или обеспечить одновременную работу нескольких программ.

Для решения всех этих проблем программисты разработали вспомогательные программы (точнее, программные системы, состоящие из многих программ), которые называются операционными системами.

Операционная система (ОС) — это комплекс программ, обеспечивающих согласованную работу всех узлов компьютера, а также удобный *интерфейс* (способ обмена данными) между пользователем и прикладными программами, с одной стороны, и аппаратными средствами компьютера — с другой.

Операционная система обеспечивает:

- взаимодействие пользователя и аппаратных средств;
- обмен данными между прикладными программами и устройствами компьютера;
- работу файловой системы (хранение данных в виде файлов и папок);
- запуск и выполнение прикладных программ;
- обработку ошибок, контроль за работой оборудования;
- распределение ресурсов компьютера между несколькими одновременно работающими программами (время работы процессора, память, внешние устройства).

Операционные системы бывают **однозадачные** (на компьютере в любой момент выполняется только одна программа) и **многозадачные** (пользователь может запустить несколько программ, которые будут выполняться одновременно).

Первые операционные системы появились на компьютерах *второго поколения* и были *однозадачными*. Нередко получалось так, что большую часть времени занимали не вычисления, а операции ввода и вывода данных, тогда как процессор в это время простаивал. Чтобы полностью использовать мощность компьютера, разработали **пакетный режим**: в разные области памяти загружали несколько программ. Когда одна программа выполняла операции ввода/вывода, процессор переходил к выполнению следующей, и таким образом мог быть загружен практически на полную мощность.

На компьютерах *третьего поколения* часто применялся **многопользовательский режим (режим разделения времени)**, при котором с большим компьютером (мэйнфреймом) было связано не-

сколько **терминалов** (так называли рабочие места с клавиатурой и монитором). С каждого терминала можно было отправить задание на выполнение, таким образом, с компьютером одновременно работало несколько программистов.

Операционные системы *первых персональных компьютеров* были *однозадачными*. Самая популярная ОС в 1980-х годах — *MS DOS* (англ. *Microsoft Disk Operating System* — дисковая операционная система фирмы Microsoft). Сейчас иногда на недорогие ноутбуки устанавливается её бесплатный аналог — *FreeDOS* (www.freedos.org).

Все *современные ОС многозадачные*. ОС распределяет время работы процессора между запущенными на выполнение программами, выделяя каждой *кванты* (небольшие интервалы) времени, так что создается впечатление, что программы работают одновременно, даже если на компьютере установлен один процессор.

В **состав операционной системы** обычно входят:

- *начальный загрузчик* — небольшая программа, расположенная в самом первом секторе загрузочного диска; его задача — организовать загрузку в память ядра (основной части) ОС и передать ему управление;
- *система управления памятью*;
- *система управления задачами*, которая обеспечивает загрузку в память и выполнение программ, а также распределение ресурсов между ними;
- *система ввода/вывода*, которая управляет внешними устройствами и файлами; она использует программы, предназначенные для обмена данными с дисководами, клавиатурой, монитором и принтером, записанные в постоянном запоминающем устройстве (ПЗУ) микросхемы BIOS, расположенной на материнской плате (см. § 35);
- *командный процессор* — программа, которая выполняет команды пользователя, введенные в командной строке, и командные файлы — текстовые файлы, содержащие списки команд и даже программы на специальном языке программирования;
- *утилиты* (лат. *utilitas* — польза) — служебные програмы для проверки и настройки компьютера.

Может ли компьютер работать без операционной системы? Да, в том случае, если он работает по одной-единственной программе, которая хранится в ПЗУ или на диске, и автоматически запуска-

ется при включении питания. Например, микрокомпьютеры, встроенные в бытовые устройства, могут обходиться без операционной системы. Однако такой компьютер очень сложно программировать (нужно обращаться напрямую к аппаратуре) и невозможно настраивать, поэтому во многих более сложных устройствах (игровых приставках, банковских терминалах и т. д.) используют операционные системы.

Современные операционные системы

Самые популярные современные **операционные системы для персональных компьютеров** — Windows, Mac OS и Linux. Все они используют графический интерфейс с пользователем: окна программ, управление с помощью мыши, кнопки, переключатели и т. п.

Система **Windows** разработана фирмой Microsoft (www.microsoft.com) и распространяется на коммерческой основе. Под управлением Windows работает более 90% персональных компьютеров, имеющих доступ в Интернет.

Примерно 5% пользователей используют операционную систему **Mac OS**. Она устанавливается на компьютеры фирмы Apple, которые часто используют профессионалы в области дизайна, компьютерной графики, полиграфии, видеомонтажа.

Около 1% компьютеров работают под управлением ОС **Linux**. Ее начал разрабатывать в 1991 г. финский студент *Линус Торвальдс* в качестве хобби. Сейчас в развитии Linux принимают участие сотни разработчиков во всём мире. В современном ядре Linux насчитывается более 11 млн строк кода. Система Linux распространяется бесплатно вместе с исходными кодами, так что каждый (при желании и умении) может её усовершенствовать.

На основе ядра Linux построено много различных *дистрибутивов* (распространяемых сборок), самые известные из них — Ubuntu (www.ubuntu.com), Mandriva (www.mandriva.ru), OpenSUSE (www.opensuse.org), Slackware (www.slackware.com), Gentoo (www.gentoo.org). В дистрибутивы входит не только сама операционная система, но и программное обеспечение, состав которого зависит от конкретной сборки. Существуют дистрибутивы с улучшенной поддержкой русского языка, например *ALT Linux* (www.altlinux.org).

Достоинства Linux:

- бесплатное распространение ОС и многих программ для нее;
- высокий уровень безопасности и защиты от вирусов;

- невысокие требования к аппаратным средствам;
- возможность гибкой настройки.

Основные сферы применения Linux:

- личные компьютеры (не нужно платить за ПО);
- портативные компьютеры, которые закупаются организациями в большом количестве;
- серверы в локальных сетях и в Интернете (до 50% всех серверов), важно быстродействие;
- суперкомпьютеры (до 80% всех суперкомпьютеров), важна возможность настройки для работы на нестандартном оборудовании;
- встроенные компьютеры в банкоматах, терминалах оплаты, стиральных машинах и даже беспилотных военных аппаратах; важна бесплатность и возможности настройки.

Среди *недостатков этой ОС* обычно отмечают:

- сложность настройки для неквалифицированного пользователя (для выполнения многих операций необходимо вводить команды в режиме командной строки);
- отсутствие драйверов для некоторых устройств и сложность их установки;
- отсутствие версий популярных профессиональных программ, например Adobe Photoshop;
- отсутствие поддержки современных игр.

Появление карманных персональных компьютеров (КПК), смартфонов и коммуникаторов привело к развитию специальных **операционных систем для мобильных устройств**, которые могут работать на маломощном оборудовании. Представители ОС этого типа — *Google Android* (на основе ядра Linux), *Symbian*, *Windows Phone*, *BlackBerry*. Портативные компьютеры фирмы Apple (iPhone, iPad) работают под управлением операционной системы *iOS*.

Новая операционная система компании Google для персональных компьютеров, названная *Chrome OS*, строится на ядре Linux. Она нетребовательна к аппаратным ресурсам компьютера, основная роль отводится **веб-браузеру и «облачным вычислениям»**. Данные пользователя хранятся на серверах Интернета, для их обработки используются веб-службы, при этом на компьютер не нужно устанавливать дополнительное программное обеспечение. Недостаток этой ОС — низкая безопасность. Также она не подойдёт тем, кому нужно выполнять сложную обработку графики и видео.

Существует ещё один класс операционных систем, от которых требуется не просто решать задачи, а делать это за определённый промежуток времени. Такие ОС называются **операционными системами реального времени.** Они применяются в тех случаях, когда задержка может привести к аварии, катастрофе или финансовым потерям: в системах аварийной защиты, системах управления роботами и самолётами, в военных приборах. Например, робот, снимающий деталь с конвейера, должен сделать это за маленький промежуток времени. Наиболее известные системы реального времени — *QNX* (www.qnx.com), *Windows CE* (www.microsoft.com), *VxWorks* (www.windriver.com) и *LynxOS* (www.lynuxworks.com/rtos).

Многие современные операционные системы, включая Linux, Mac OS, QNX, VxWorks, LynxOS, относятся к классу **UNIX-подобных ОС**. Это значит, что они используют общие идеи и принципы, заложенные в 1970-х годах при разработке системы UNIX:

- для настройки и управления системой используются простые текстовые файлы;
- программы часто используют текстовый ввод данных и вывод результатов;
- широко применяются утилиты, запускаемые в командной строке;
- каждая утилита выполняет одну задачу; её режимы работы можно задавать с помощью параметров командной строки;
- утилиты можно объединять в «конвейер», направляя результаты работы одной утилиты на вход следующей;
- все устройства (жёсткие диски, флэш-диски, принтеры, сканеры) рассматриваются как файлы.

Сейчас **система UNIX** используется в основном для управления серверами. Все UNIX-подобные системы считаются очень надёжными с точки зрения безопасности. Достаточно сказать, что для них практически неактуальна проблема компьютерных вирусов.

Драйверы устройств

Драйверы (англ. *driver* — водитель) — это программы специального типа, которые находятся в оперативной памяти и обеспечивают обмен данными между ядром ОС и внешними устройствами компьютера (принтером, сканером и др.), а также контроллерами (звуковой картой, видеокартой, сетевой картой и т. п.). Драйверы обычно включают в подсистему ввода/вывода.

Драйвер представляет собой набор процедур, которые вызываются ядром ОС при необходимости передать данные устройству или принять от него данные. Задача драйвера — преобразовать команды ввода/вывода в команды конкретного устройства (рис. 6.12). Драйверы загружаются в память и фактически становятся частью ОС. Такая схема позволяет устанавливать и использовать устройства, которые были разработаны уже после выпуска операционной системы.



Рис. 6.12

Если драйвер не установлен, устройство работать не будет, потому что неизвестно, как к нему обращаться. Драйверы наиболее популярных устройств обычно включаются в дистрибутив (установочный пакет) операционной системы. Когда ОС обнаруживает новое устройство, она пытается найти подходящий драйвер в своей базе данных. Если такого драйвера нет, его можно установить вручную с диска, который прилагается к устройству. Кроме того, любой драйвер можно бесплатно скачать из Интернета с сайта производителя.

Утилиты

Утилиты решают вспомогательные задачи, расширяя возможности ОС. К утилитам относятся:

- программы для проверки дисков (*chkdsk* в Windows, *fsck* в Linux);
- программы для разбивки жёстких дисков, с помощью которых можно сделать несколько разделов на одном диске (**Управление дисками** в Windows; **GNU Parted** в Linux);
- файловые менеджеры — программы для работы с файлами; самые известные файловые менеджеры для Windows: **Проводник** (входит в состав ОС), **Total Commander** (www.ghisler.com), **Free Commander** (www.freecommander.com), **Far Manager** (farmanager.com); в Mac OS использу-

ется программа *Finder*, а в операционной системе Linux — файловые менеджеры *Konqueror*, *Midnight Commander* и др.;

- антивирусные программы: *AVP* (www.kaspersky.ru), *DrWeb* (www.drweb.com), *Nod32* (www.eset.com), *McAfee* (home.mcafee.com) и др.; бесплатны для домашнего использования антивирусы *AVG* (freeavg.com), *Avast* (avast.com), *Avira* (www.avira.de), *Panda* (www.pandasecurity.com);
- архиваторы и программы для сжатия данных; в ОС Windows наиболее популярны: *WinRAR* (www.rarlab.com) и *WinZip* (www.winzip.com); в Linux — *Ark* (utils.kde.org) и *File Roller* (fileroller.sf.net); **архиватор** 7ZIP (www.7-zip.org) распространяется бесплатно с исходными кодами для различных операционных систем;
- программы для шифрования данных, например, *PGP* и её версии для разных операционных систем (www.pgpru.com);
- редакторы, позволяющие менять данные на диске и в оперативной памяти; например, программы *HxD* (mh-nexus.de/en/hxd) и *WinHex* (www.winhex.com) для ОС Windows или *hexedit* (rigaux.org/hexedit.html) для Linux;
- сетевые утилиты для проверки связи в локальной и глобальной сетях; например, утилиты *ping*, *tracert* (*traceroute*), *nslookup* в Windows и Linux.

Часто к утилитам относят также:

- программы для записи CD и DVD-дисков; в системе Windows наиболее известны программы *Nero Burning ROM* (www.nero.com), *CDBurnerXP* (cdburnersp.se) и *DeepBurner* (www.deepburner.com); в Linux для этой цели используют утилиту *K3b* (k3b.org);
- программы для сканирования и распознавания текста; широко применяются коммерческая программа *ABBYY FineReader* (www.abbyy.ru) и бесплатная *CuneiForm* (www.cuneiform.ru).

Файловые системы

Вы знаете, что данные можно хранить в виде файлов на жёстких дисках, CD- и DVD-дисках, флэш-дисках. Однако

жёсткий диск и флэш-диск «ничего не знают» о том, что записанные на них данные в самом деле объединены в файлы и папки.

Вместе с тем когда программа сохраняет файл на диске, она «ничего не знает» о том, в какое место диска эта информация будет записана, указывается только имя файла и каталог. Поэтому между программой и носителем информации необходим «посредник», который определяет, в какое именно место диска будут записаны биты переданных данных. Эту роль выполняет **драйвер файловой системы** (рис. 6.13).



Рис. 6.13

Файловая система — это порядок размещения, хранения и именования данных на носителе информации.

Файловые системы решают несколько **задач**:

- определяют правила построения имён файлов и каталогов;
- определяют, как именно размещаются файлы на диске;
- предоставляют программам функции для работы с файлами;
- обеспечивают защиту данных в случае сбоев и ошибок;
- обеспечивают установку прав доступа к данным для каждого пользователя;
- обеспечивают совместную работу с файлами (когда один пользователь открыл файл, для остальных устанавливается режим «только чтение»).

С точки зрения файловой системы, диск делится на **кластеры** (блоки) размером от 512 байтов до 64 Кбайт (кластер — это один или несколько секторов диска). Каждому файлу выделяется целое число кластеров. Файлу размером 1 байт выделяется целый кластер, остальное место считается занятым, но фактически не используется. Поэтому при большом размере кластера хранить мелкие файлы невыгодно, значительная часть места пропадает впус-

тую. Вместе с тем при увеличении размера кластера скорость чтения и записи больших файлов повышается, кроме того, увеличивается и максимальный объём диска, который поддерживает файловая система.

В операционной системе Linux применяются файловые системы *ext3* и *ext4*. Они поддерживают **журналирование**, помогающее сохранить данные в случае сбоев. Его суть в том, что перед выполнением операции с файлами ОС записывает «план действий» в специальный журнал. Когда операция полностью закончена, эта запись из журнала удаляется. Если во время операции произошел сбой (например, отключение питания), по записям в журнале можно сразу определить, какие файлы могли быть затронуты. Таким образом, журналируемая файловая система устойчива к сбоям.

В системе Windows применяют файловые системы *NTFS* и *FAT32*. Хотя FAT32 в некоторых случаях работает быстрее и требует меньше памяти, она считается устаревшей. В отличие от FAT32 файловая система NTFS:

- обеспечивает защиту от сбоев с помощью журналирования (в FAT32 журналирования нет);
- позволяет назначить права доступа к файлам и папкам (в FAT32 каждый пользователь может просматривать и изменять данные всех остальных);
- позволяет задать квоту (ограничение) на использование диска каждому пользователю;
- позволяет использовать сжатие файлов и папок без дополнительных программ.

В операционной системе Mac OS применяется файловая система *HFS* (англ. *Hierarchical File System* — иерархическая файловая система).

Первые файловые системы были **одноуровневыми**, т. е. все файлы хранились в одном каталоге (на дискете). С увеличением ёмкости дисков (и количества файлов!) это стало неудобно, поэтому разработали **иерархические** (многоуровневые) **файловые системы**, где файлы группируются в каталоги, а каталоги вложены друг в друга. Такая структура называется **деревом каталогов**.

В операционной системе Linux существует один корневой каталог (обозначаемый как «/»), остальные файлы и каталоги вложены в него. Любое устройство (включая жёсткие диски,

принтеры, сканеры и т. п.) в Linux рассматривается как файл, т. е. входит в состав единой иерархической файловой системы (рис. 6.14).

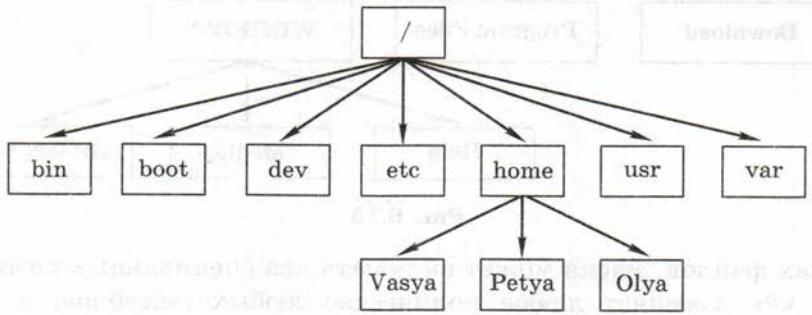


Рис. 6.14

Вот что хранится в каталогах, показанных на схеме на рис. 6.14:

- bin — команды операционной системы;
- boot — ядро ОС и данные для загрузки;
- dev — файлы устройств, подключённых к ОС; устройства присоединяются к файловой системе (монтируются) с помощью специальной команды;
- etc — файлы с настройками ОС и некоторых программ;
- home — домашние каталоги пользователей;
- usr — установленные пакеты программ;
- var — часто меняющиеся данные, например журналы ОС.

Чтобы указать путь к файлу или каталогу, перечисляют (начиная от корня) все каталоги, в которых он находится, разделяя их символом «/» («слэш»). Например, адрес домашнего каталога пользователя petya запишется как /home/petya, а адрес файла qq.txt в этом каталоге — как /home/petya/qq.txt.

Дерево каталогов в операционной системе Windows строится отдельно для каждого диска (рис. 6.15).

В качестве разделителя при записи адреса файла или каталога (папки) используют обратный слэш «\», например: C:\WINDOWS\System32\shell32.dll.

Для работы с группами файлов применяются **маски** или **шаблоны** (англ. *wildcards*). Кроме символов, которые допустимы в

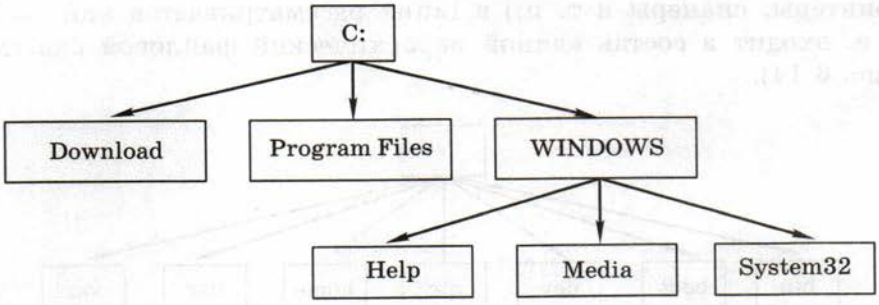


Рис. 6.15

именах файлов, маска может включать два специальных символа: знак «*» заменяет любое количество любых символов, а знак «?» — один любой символ. Приведём несколько примеров:

.	все файлы;
*.bmp	все файлы с расширением bmp;
a*.?	файлы, имя которых начинается с буквы «a», а расширение состоит из одного символа;
x.??*	файлы, в имени которых есть буква «x», а расширение содержит не менее двух символов;
*z.a?	файлы, имя которых заканчивается на букву «z», а расширение начинается с буквы «a» и состоит из двух символов.

Маски чаще всего применяют для поиска файла по известной части имени или по расширению. Например, для того чтобы найти все документы Microsoft Word, имя которых содержит слово «отчет», можно использовать маску *отчет*.doc*. При этом будут найдены, например, такие файлы:

отчет2011.doc
Самый_важный_отчет.docx
Новый_отчет_март_2011.docx

В операционной системе Windows заглавные и строчные буквы в названиях файлов и каталогов не различаются, т. е. к файлу с именем Вася.txt можно обращаться как вася.txt, вАсЯ.txt, ВаСя.txt или ВАСЯ.txt. В Unix-подобных ОС (Linux, Mac OS) это не так, все перечисленные имена файлов — разные, и такие файлы могут быть созданы в одном каталоге.

Нужно отметить, что файловая система не обязательно напрямую связана с жёстким диском или другим физическим носителем информации. Существуют, например, **сетевые файловые системы**, которые являются просто способом обращения к файлам на удалённых компьютерах.

Вопросы и задания



1. Зачем нужны операционные системы? Можно ли обойтись без них?
2. Что такое операционная система?
3. Какие задачи выполняет ОС?
4. Чем отличаются многозадачные ОС от однозадачных?
5. Как обеспечивается многозадачность на компьютерах с одним процессором?
6. Что такое пакетный режим работы?
7. Что такое многопользовательский режим?
8. Перечислите составные части ОС.
9. Что такое начальный загрузчик?
10. Что такое драйвер?
11. Какие программы относятся к утилитам?
12. Какими достоинствами и недостатками обладает система Linux?
13. Как ОС обменивается данными с внешними устройствами? В чём достоинства такой схемы?
14. Что происходит, когда ОС обнаруживает новое устройство?
15. Что такое файловая система? Зачем она нужна?
16. Какие задачи решает файловая система?
17. Что такое кластер?
18. Почему невыгодно хранить мелкие файлы в файловой системе с большим размером кластера?
19. Что улучшается при увеличении размера кластера?
20. Какие файловые системы используются в ОС Linux, Windows и Mac OS?
21. Почему файловая система FAT32 считается устаревшей?
22. Что такое одноуровневая и многоуровневая файловые системы?
23. Что такое корневой каталог?
24. В чём различие файловых систем в ОС Linux и Windows?
25. Где расположены каталоги пользователей в ОС Linux?
26. Какие символы используются как разделители при записи адреса файла в ОС Linux и Windows?
27. Что такое сетевая файловая система?

Подготовьте сообщение:

а) «Журналирование в файловых системах»
б) «Операционные системы для персональных компьютеров»
в) «Операционные системы для мобильных устройств»
г) «Операционные системы реального времени»
д) «UNIX-подобные операционные системы»

Задачи

1. Пользователь vasya работает в ОС Linux. Перемещаясь из одного каталога в другой, он последовательно посетил каталоги math, lections, professor и оказался в своем домашнем каталоге. Каково полное имя каталога, из которого начал перемещение пользователь?
2. Пользователь ОС Windows, перемещаясь из одного каталога в другой, последовательно посетил каталоги Математика, Задания, c:\, Классы, 10-А. Каково полное имя каталога, из которого начал перемещение пользователь?
3. Определите, какие из указанных имён файлов удовлетворяют маске:

   ?hel*lo.c?*

   а) hello.c
   б) hello.cpp
   в) hhelolo.cpp
   г) hhelolo.c
   д) hello.cc
   е) ahello.cpp
   ж) ahelolo.c
   з) ahelolo.cp

4. Определите, какие из указанных имен файлов удовлетворяют маске:

   d?cf*.jp*g

   а) dscf34.jpeg
   б) d1cfab.jpg
   в) dccf6754.jpeg
   г) dcsf1111.jpeg
   д) d2cf34.jpeg
   е) dcf1234.jpg
   ж) dsscf6754.jpg
   з) dscf.jpg

5. Определите, по какой из масок будет выбрана следующая группа файлов:

   abcd.txt, bc.tar, bcd.txt, bc.tgz

   а) *bc*.?t*
   б) ?bc?.t??
   в) ?bc?.t*
   г) *bc*.t??

6. Определите, по какой из масок будет выбрана следующая группа файлов:

0qqq.txt, qq0q.ppt, 0qq0.txt, aqqb.ppt

а) ?0*.???
б) ?qq*.*
в) *qq*.?t
г) ?q??.???

7. Определите, какой из перечисленных файлов подойдет под все предложенные маски:

12abc.xls, xabx.xml, abc.xls, aba.xml

а) *ab*.?x*
б) ?ab?.x??
в) ?ab*.x*
г) *ab?.x??

8. В каталоге находятся следующие файлы:

fort.docx, ford.docx, lord.doc, orsk.dat, port.doc

Определите, по какой из масок будут выбраны все эти файлы, кроме orsk.dat

а) *o?*.d?*
б) ?o*?.d*
в) *or*.doc?
г) ?or?.doc?

§ 41 Системы программирования

Зачем нужны системы программирования?

Процессор, выполняющий всю обработку данных, понимает только машинные команды (числовые коды). Чаще всего их записывают в шестнадцатеричном коде, например, так:

```
B82301052500
```

Чтобы понять, что делает этот код, нужно взять таблицу команд процессора и посмотреть, что означает каждая пара шестнадцатеричных цифр (байт).

Программы для первых компьютеров составляли именно в **машинных кодах**. Программирование было доступно только специалистам, отладка программы занимала очень много времени.

Человек плохо воспринимает коды, поэтому в дальнейшем для каждой машинной команды придумали символические обозначения. Например, приведённая выше программа — это две машинные команды для процессоров фирмы Intel, их можно записать так:

```
MOV AX,0123h
ADD AX,25h
```

Здесь AX — это имя регистра (ячейки памяти) процессора, команда MOV записывает в регистр новое значение, а команда ADD добавляет число к содержимому ячейки. Буква «h» после числа означает, что оно записано в шестнадцатеричной системе счисления.

Вспомним, что процессор может выполнить только программу, написанную в машинных кодах. Поэтому возникает задача: перевести программу с такого языка в машинные коды. Для этого используют **программы-ассебмлеры** (англ. *assembler* — рабочий-сборщик), а сам язык называется **языком ассемблера**. Этот язык **машинно-ориентированный**, потому что он определяется набором команд конкретного процессора (ориентирован на машину).

Очевидно, что программировать на языке ассемблера тоже не очень удобно — нужно хорошо знать команды процессора, организацию памяти и т. п. Кроме того, каждый процессор имеет свою систему команд и свой язык ассемблера. Это значит, что программы на языке ассемблера *непереносимы* — программа, написанная для одного процессора, не будет работать на другом.

Людям хочется (в идеале) разговаривать с компьютером на естественном языке, не думая о том, какой процессор в нём установлен. К сожалению, пока это невозможно. Сейчас для программирования чаще всего используют компромиссный вариант — **языки программирования высокого уровня**, или **алгоритмические языки**. Это **формальные языки**, созданные специально для разработки программ. Команды строятся из слов естественного (чаще всего, английского) языка, каждая команда воспринимается однозначно в соответствии с установленными правилами.

Для перевода программы, написанной на языке высокого уровня, в машинные коды, применяют специальные программы — **трансляторы** (англ. *translator* — переводчик). Кроме трансляторов в системы программирования входят и другие программы, о которых будет рассказано далее.

Системы программирования — это программные средства для создания и отладки новых программ.

Языки программирования

К 2010 году в мире было разработано более 8500 языков программирования. Первой программисткой в мире считается *Ада Лавлейс*, которая в 1843 г. написала программу для Аналитической машины Чарльза Бэббиджа. В 1979 г. в США был разработан язык программирования **Ада**, названный в её честь. Один из первых алгоритмических языков — **Фортран** — создан в 1957 г., однако он и сейчас применяется для научных вычислений.

Как вы уже знаете, языки программирования можно разделить на **языки низкого уровня** (машинно-ориентированнные, языки ассемблера) и **языки высокого уровня** (алгоритмические языки). По области применения языков программирования выделяют:

- профессиональные языки общего назначения: **Java, C, C++, C#, Visual Basic, Delphi;**
- языки для программирования интернет-сайтов: **PHP, JavaScript, Perl, ASP, Python;**
- языки для решения задач искусственного интеллекта: **Лисп, Пролог;**
- языки для обучения программированию: **Бейсик, Паскаль, Лого, Python.**

Трансляторы

Основа любой системы программирования — транслятор.

Транслятор — это программа, которая переводит в машинные коды тексты программ, написанных на языке высокого уровня.

Существуют два типа трансляторов: интерпретаторы и компиляторы.

Интерпретатор анализирует текст программы по частям. Разобрав очередной фрагмент, он немедленно выполняет описанные в нем действия и переходит к обработке следующего фрагмента.

Достоинства интерпретаторов:

- программы переносимы (программа будет работать в любой системе, где установлена программа-интерпретатор);
- удобно отлаживать программу.

Есть и существенные *недостатки*:

- программу невозможно выполнить, если не установлен интерпретатор;
- программы выполняются медленно (в цикле из 100 шагов каждая строчка 100 раз «разбирается» интерпретатором);
- в тех частях программы, которые не выполнялись во время отладки, могут оставаться синтаксические ошибки.

Второй тип трансляторов — **компиляторы**. Они, в отличие от интерпретаторов, сразу переводят всю программу в машинный код и строят исполняемый файл, готовый к запуску.

Достоинства компиляторов:

- чтобы запустить программу, не нужно устанавливать транслятор;
- поскольку программа уже переведена в машинные коды, она выполняется значительно быстрее, чем при использовании интерпретатора.

Недостатки тоже есть:

- при любом изменении нужно ждать окончания компиляции (перевода в коды); это несколько затрудняет отладку;
- готовая программа будет выполняться только в той операционной системе, для которой она была создана[1].

Чтобы как-то совместить достоинства интерпретаторов и компиляторов, была предложена идея компиляции программы в некоторый промежуточный исполняемый код (**псевдокод**, **Р-код**), а не сразу в команды конкретного процессора. Для выполнения такого псевдокода нужна специальная среда — **виртуальная машина**, которую в принципе можно разработать для любого процессора и любой операционной системы.

[1] Многие программы, разработанные для ОС Windows, могут быть запущены в Linux с помощью программы-оболочки Wine (www.winehq.org).

Программа сначала обрабатывается компилятором, который строит псевдокод, а потом этот псевдокод выполняется интерпретатором. Таким образом,

- при компиляции в псевдокод проверяются все синтаксические ошибки, поэтому при выполнении такую проверку делать не нужно; это значительно ускоряет работу программ в сравнении с интерпретацией;
- обеспечивается переносимость программ — можно выполнять программу (псевдокод) на любом компьютере, где есть виртуальная машина.

Байт-код — это разновидность псевдокода, в котором команда занимает 1 байт, а далее следуют её аргументы (или их адреса). Современные версии интерпретируемых языков Perl, PHP, Python используют компиляцию в байт-код для ускорения выполнения программы.

Готовые программы на **Java** распространяются в виде байт-кода, поэтому для их выполнения необходимо установить **виртуальную Java-машину**. При этом для ускорения работы часто используется **JIT-компиляция** (англ. **JIT** — *just-in-time* — в это самое время), при которой байт-код «на лету» преобразуется в команды конкретного процессора. Тогда при повторном выполнении команды трансляция уже не нужна.

Аналогичный подход применяется в **среде .NET**, которую разработала фирма Microsoft. Одна из основных идей среды .NET — объединение программ, написанных на разных языках. В частности, разные части программы могут быть написаны на **C#**, **J#**, **VB.NET**, **Delphi.NET**, все они в конечном счёте транслируются в байт-код на промежуточном языке **IL** (англ. *Intermediate Language*), который потом выполняется виртуальной машиной.

Состав системы программирования

В **состав системы программирования** обычно входят:

- **транслятор**;
- **компоновщик** (редактор связей, сборщик, англ. *linker*) — программа, которая собирает разные части (модули) создаваемой программы и функции из стандартных библиотек в единый исполняемый файл. На рисунке 6.16 показано, как собирается программа на языке Си, состоящая из двух модулей (исходные файлы qq.c и qq1.c).

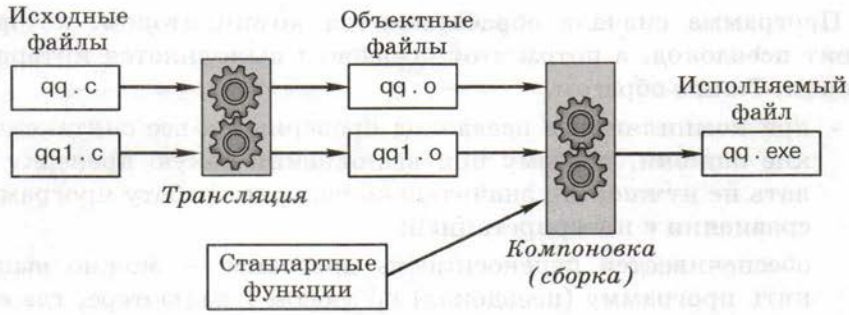


Рис. 6.16

- **отладчик** (англ. *debugger*[1]) — программа для поиска ошибок в других программах, позволяющая:
  - выполнять программу в пошаговом режиме (по одной строке);
  - выполнять программу до строки, где установлен курсор;
  - устанавливать точки останова (англ. *breakpoints*);
  - просматривать и изменять значения переменных в памяти;
- **профилировщик** (англ. *profiler*) — программа, позволяющая оценить время работы каждой процедуры и функции («профиль» времени выполнения программы); используется для того, чтобы выяснить, какую именно процедуру нужно оптимизировать в первую очередь.

Любая система программирования включает **библиотеки стандартных подпрограмм**. Это набор готовых процедур и функций, которые можно вызывать из своей программы. Например, в большинстве языков программирования есть стандартные функции для вычисления синуса и косинуса. Они подключаются к программе на этапе сборки, это делает компоновщик.

Многие программы используют одни и те же достаточно сложные системные функции (например, операции с окнами в графической среде). Если включать эти функции в код каждой программы, размеры исполняемых файлов намного увеличатся, из-за этого жёсткий диск и память будут расходоваться неэффективно.

[1] Согласно одной из версий, это название связано с жучком, который попал между контактов реле компьютера Mark II в 1947 г. Дословно: debug — «удаление жучков».

Поэтому библиотеки таких функций хранятся на диске в виде отдельных файлов — **динамически подключаемых библиотек**, в системе Linux они имеют расширение so (от англ. *shared objects* — разделяемые объекты), а в Windows — расширение dll (от англ. *dynamic-link library* — динамически подключаемая библиотека). Когда программа вызывает функцию из такой библиотеки, библиотека загружается в память, и управление передаётся вызванной функции. Несколько программ могут обращаться к одной и той же копии библиотеки в памяти.

Набор стандартных структур данных и функций операционной системы, которые программисты могут использовать в прикладных программах, называется **интерфейсом прикладного программирования** (англ. **API** — *Application Programming Interface*). В ОС Windows применяется **Windows API**, а в Unix-подобных операционных системах — стандарт **POSIX** (англ. *Portable Operating System Interface for Unix* — переносимый интерфейс операционных систем Unix).

Сейчас для разработки программ чаще всего используют **интегрированные среды** (англ. **IDE** — *Integrated Development Environment*). В такую оболочку обычно входит текстовый редактор для набора текста программ, транслятор, компоновщик, отладчик и профилировщик.

Многие современные интегрированные среды позволяют строить интерфейс программы (расположение элементов в окне) с помощью мыши. Они называются **средами быстрой разработки приложений** (*англ.* RAD — Rapid Application Development) или **средами визуального программирования**. На рисунке 6.17 показано окно RAD-среды *Lazarus* для программирования на объектной версии языка Паскаль.

Среди профессиональных RAD-сред нужно в первую очередь назвать *Microsoft Visual Studio* (msdn.microsoft.com/vstudio). Её профессиональная версия — коммерческая, но все желающие могут бесплатно скачать и использовать ограниченную версию **(Express)** для учебных целей.

Большой популярностью пользуются также среды *Dev-C++* (wxdsgn.sourceforge.net) и *Delphi* (embarcadero.com). Кроссплатформенная среда *Code::Blocks* (www.codeblocks.org) распространяется бесплатно, существуют версии для Windows, Mac OS и Linux.

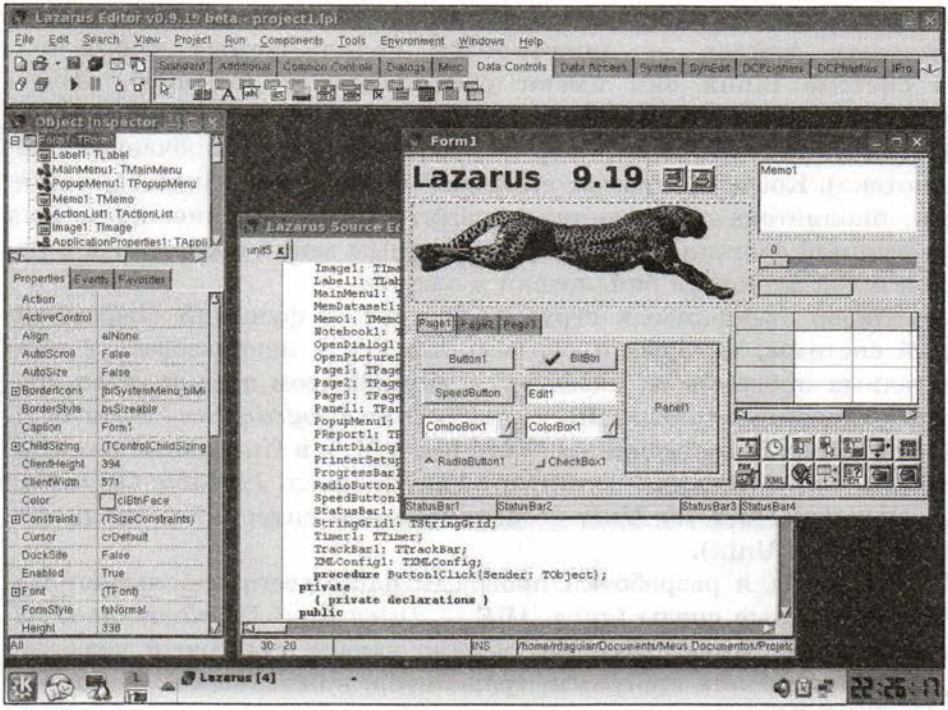


Рис. 6.17

Вопросы и задания

1. Что такое машинный код?
2. Зачем нужны системы программирования? Можно ли обходиться без них?
3. Что такое язык ассемблера? Почему он называется машинно-ориентированным?
4. Что такое язык программирования высокого уровня?
5. Как можно разделить языки программирования по области применения?
6. Зачем нужен транслятор?
7. Какие два типа трансляторов вы знаете? В чём их достоинства и недостатки?
8. Какие программы входят в системы программирования?
9. Зачем нужен компоновщик?
10. Что такое отладчик? Перечислите возможности отладчиков.
11. Что такое профилировщик? Зачем он нужен?
12. Что такое интегрированная среда разработки?

Подготовьте сообщение

а) «Классификация языков программирования»
б) «Среды для быстрой разработки программ (RAD)»
в) «Как выполняются программы на Java?»
г) «Платформа Microsoft .NET»
д) «Средства отладки программ»
е) «Динамически подключаемые библиотеки»

§ 42 Инсталляция программ

Большинство современных программ требуется устанавливать (**инсталлировать**, от англ. *install* — установить). Это связано с тем, что:

- необходимо проверить, соответствует ли компьютер требованиям (к процессору, оперативной памяти, операционной системе и т. д.), которые обязательны для работы программы;
- программы содержат множество файлов, которые должны быть записаны на диск определённым образом;
- у пользователя должна быть возможность выбора нужных ему компонентов программы (остальные не устанавливаются);
- необходимо записать некоторые файлы в каталоги операционной системы (например, при установке драйверов устройств);
- необходимо настроить режимы работы программы с учётом особенностей компьютера;
- при установке коммерческих программ необходимо вводить ключ (серийный номер копии программы).

Инсталляция — это установка и настройка программы на компьютере пользователя.

Пользователь получает программу в виде **дистрибутива** (установочного пакета, от англ. *distribute* — распространять). Дистри-

бутив — это несколько файлов на CD- или DVD-диске или один файл (который часто можно загрузить из Интернета). В таком виде программу невозможно использовать по прямому назначению. Данные в дистрибутиве обычно сжаты, распаковка происходит во время установки. Чтобы установить или удалить ПО, пользователь должен иметь права администратора компьютера.

Иногда программное обеспечение может поставляться в виде исходного кода. Чтобы преобразовать его в готовую для выполнения программу, нужно использовать одну из систем программирования.

Обычно установка включает несколько этапов (некоторые из них могут отсутствовать):

- просмотр лицензионного соглашения (договора о возможности использования программы);
- ввод ключа (серийного номера) программы;
- выбор компонентов программы, которые пользователь хочет установить;
- определение каталога, в котором нужно разместить файлы программы;
- распаковка и копирование файлов на жёсткий диск компьютера;
- настройка программы с помощью файлов конфигурации (или запись настроек в системный реестр в ОС Windows);
- создание ярлыков для запуска программы в меню и/или на Рабочем столе.

В операционной системе Linux программы чаще всего распространяются в виде **пакетов** (файлы с расширениями rpm или deb, в зависимости от сборки) или **в исходных кодах**. Для установки пакетов используются утилиты *apt-rpm* и *apt-get*. Они позволяют проверить зависимости пакетов: устанавливается не только указанный пакет, но и другие пакеты, необходимые ему для работы.

Начинающим пользователям Linux проще всего использовать графические программы для работы с пакетами (**менеджеры пакетов**), например *Aptitude* или *Synaptic* (рис. 6.18). В них можно мышью отметить пакеты, которые требуется установить, обновить или удалить.

В ОС Windows для установки программ используется служба *Windows Installer*, которая работает с установочными пакета-

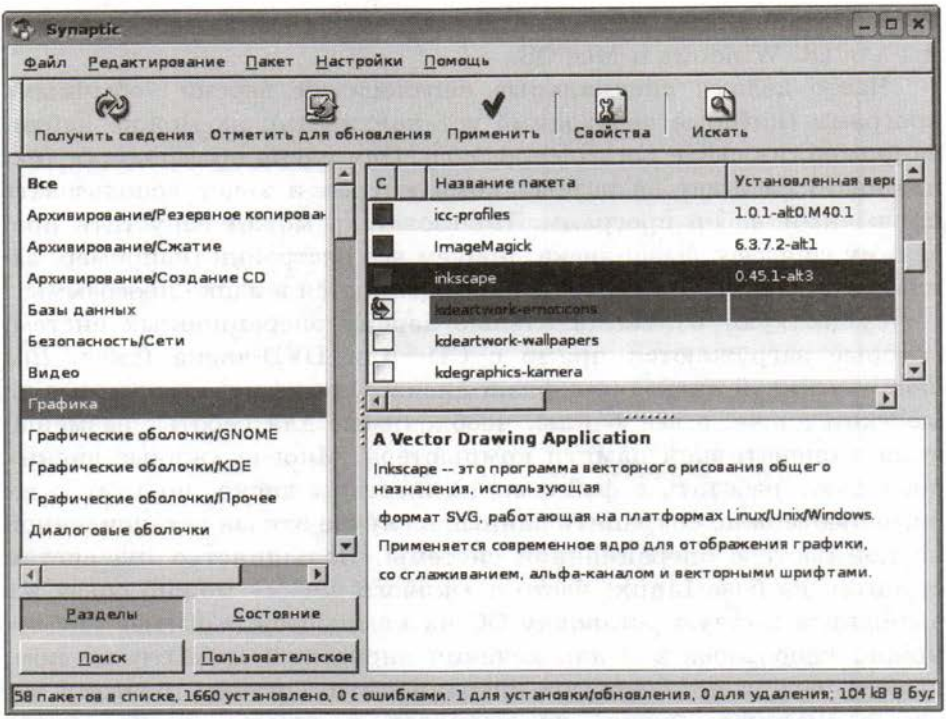


Рис. 6.18

ми — файлами в формате msi. Дистрибутив также может представлять собой программу (файл с расширением exe), которая содержит все необходимые данные и при запуске «ведёт» пользователя через все этапы установки.

Программы для операционной системы Mac OS распространяются в виде пакетов (файлов с расширением pkg). Для работы с ними в ОС включена программа *Installer*.

Системные администраторы, которым приходится устанавливать ПО на большое количество компьютеров, нередко используют автоматическую установку (без участия человека) или удалённую установку (через сеть).

Установка ПО — это дополнительное неудобство для пользователей. Поэтому особой популярностью пользуются **переносимые программы** (англ. *portable applications*). Их не нужно устанавливать, они могут быть просто скопированы на жёсткий диск компьютера или запущены прямо с CD-, DVD- или флэш-диска.

К этой группе относятся многие небольшие бесплатные программы для ОС Windows и Mac OS.

Часто делают специальные переносимые версии «обычных» программ (которые необходимо устанавливать); их можно найти, например, на сайте portableapps.com. Они очень полезны для тех, кто часто работает на разных компьютерах и хочет использовать привычный набор программ. Пользователь может запустить программу со своего флэш-диска, причем все настройки (например, закладки в браузере Opera Portable) сохраняются в папке программы.

Существуют **ознакомительные версии** операционных систем, которые загружаются прямо с CD- или DVD-диска (англ. *live disc* — «живой диск») или флэш-диска. Они не меняют данные на жёстком диске, а все файлы, необходимые для работы, размещаются в оперативной памяти компьютера. Многие «живые диски» позволяют работать с файлами на жёстком диске, поэтому с их помощью можно сохранить данные в случае отказа установленной на компьютере операционной системы. Большинство live-систем строится на базе Linux; часто с «живого диска» можно сразу же выполнить полную установку ОС на компьютер. «Живые диски» можно использовать и для лечения вирусов: в этом случае компьютер загружает «чистую» операционную систему, поэтому вирус с жёсткого диска не попадает в память и не может блокировать работу антивируса.

Вопросы и задания

1. Что такое инсталляция? Почему она необходима для многих современных программ?
2. Что происходит во время инсталляции?
3. Что такое дистрибутив? В чём его отличие от установленной программы?
4. Что такое менеджер пакетов?
5. Как устанавливаются программы в разных операционных системах?
6. Какие методы используются для массовой установки ПО?
7. Что такое переносимая программа?
8. Как можно познакомиться с операционной системой, не устанавливая её на жёсткий диск?

Подготовьте сообщение

а) «Зачем нужны инсталляторы?»
б) «Инсталляция в Windows и Linux»
в) «Распространение программ в виде пакетов»
г) «"Живые диски" (Live-CD)»

§ 43
Правовая охрана программ и данных

Авторские права

По законам большинства стран компьютерные программы и данные охраняются **авторским правом**. Это значит, что автор (или правообладатель, например, фирма, в которой работает автор) может ограничивать распространение и использование программы.

В Конституции Российской Федерации записано, что «интеллектуальная собственность охраняется законом» (ст. 41 ч. 1). Интеллектуальная собственность — это права на результаты творческой деятельности человека. Эти права детально определены в Гражданском кодексе РФ (часть IV «Права на результаты интеллектуальной деятельности и средства индивидуализации»).

Авторские права распространяются на:

- программы для компьютеров;
- базы данных (массивы данных, специально организованные для поиска и обработки с помощью компьютеров).

Не охраняются авторским правом:

- алгоритмы и языки программирования;
- идеи и принципы, лежащие в основе программ, баз данных, интерфейса;
- официальные документы.

Важно понять, что охраняется форма, а не содержание. Это значит, что авторские права получает не тот, кто придумал метод решения задачи, а тот, кто написал программу, которая решает задачу на основе предложенного алгоритма.

Согласно российским законам об авторском праве, автор — это физическое лицо (не организация). Авторское право:

- возникает «в силу создания» продукта и не требует формальной регистрации, хотя при желании автор может зарегистрировать программу в государственных органах;
- обозначается знаком ©, после которого записывается фамилия автора и год первого выпуска программы, например: © Иванов, 2008;
- действует в течение жизни и 70 лет после смерти автора;
- передаётся по наследству.

Автор получает **личные права**:

- право авторства (право считаться автором);
- право на имя (право выпускать программу под своим именем, псевдонимом или анонимно);
- право на неприкосновенность программы и её названия;

и **имущественные права** — осуществлять или разрешать:

- выпуск программы в свет;
- копирование в любой форме;
- распространение;
- изменение (в том числе перевод на другой язык).

Серьёзные нарушения авторских прав могут попасть под действие Уголовного кодекса РФ (ст. 146 «Нарушение авторских и смежных прав»). Уголовная ответственность наступает при крупном ущербе (более 50 000 руб.). Присвоение авторства (плагиат) наказывается лишением свободы на срок до 6 месяцев. В случаях незаконного использования, а также приобретения и хранения объектов авторского права (например, дисков с нелицензионными программами) в целях сбыта срок лишения свободы может достигать 5 лет (при особо крупном ущербе).

Типы лицензий на использование ПО

Право на использование программы даёт документ (договор), который называют **лицензией** (лат. *litentia*) или **лицензионным соглашением**. Это соглашение между правообладателем и пользователем, где чётко определены права и обязанности сторон. Как правило, в соответствии с лицензией пользователь без дополнительного разрешения автора может:

- установить программу на один компьютер (или так, как указано в договоре);
- вносить изменения, необходимые для работы программы на компьютере пользователя; исправлять явные ошибки;
- изготовить копию, чтобы можно было восстановить программу в случае сбоя;
- передать программу другому лицу вместе с лицензией.

Программы, которые получены и используются в соответствии с законом, называют **лицензионными**. Если же при создании копии были нарушены авторские права, её называют контрафактной или пиратской.

По **типу лицензий** можно разделить ПО на 4 типа:

- коммерческое;
- условно-бесплатное (англ. *shareware*);

- бесплатное (англ. *freeware*);
- свободное ПО (англ. *open source* — ПО с открытым кодом).

Значительная часть ПО является **коммерческой**, поскольку создаётся с целью получения прибыли путём продажи экземпляров. За каждую копию коммерческого ПО нужно платить (покупать лицензию); исходный код, как правило, не распространяется. Обычно фирмы предусматривают скидки при закупке большого количества лицензий (лицензии на организацию) и скидки для образовательных учреждений. Зарегистрированные пользователи программ имеют право на бесплатную техническую поддержку — консультации по телефону или электронной почте. Типичный пример коммерческого ПО — операционная система Windows.

Часто разработчики дают возможность бесплатно скачать пробную (англ. *trial*) версию программы из Интернета и попробовать, как она работает (англ. *try before you buy* — попробуй, прежде чем купить). Такие программы называют **условно-бесплатными** (англ. *shareware*). Пробные версии всегда имеют какие-то ограничения, например:

- ограниченный срок работы (обычно 30 дней);
- ограниченное количество запусков;
- ограничение функций (например, невозможно сохранить результаты на диске);
- встроенный рекламный блок;
- всплывающие сообщения с призывом заплатить автору деньги за программу.

Обычно в лицензионном соглашении указывается, что пробная версия не может быть использована для коммерческих целей и профессиональной работы.

К условно-бесплатным можно отнести многие популярные программы, у которых есть пробные версии. Например, на ноутбуки часто устанавливается пробная версия пакета Microsoft Office, которую можно бесплатно использовать 60 дней. В Интернете можно скачать пробные версии векторного редактора Corel Draw, почтовой программы TheBat, всех программ фирмы Adobe.

Для решения большинства задач можно найти **бесплатные** программы (англ. *freeware*). Это значит, что лицензионное соглашение не требует никаких выплат автору. Бесплатные программы можно свободно скачать из Интернета и использовать, однако чаще всего коммерческое использование и изменение запрещают-

ся, исходные коды не распространяются. Иногда фирмы бесплатно распространяют ограниченные версии коммерческих программ. К бесплатным программам относятся браузеры Opera, Chrome и Safari, программа для записи CD и DVD-дисков CDBurner XP, множество небольших утилит.

Наибольшие возможности предоставляет **свободное ПО** (англ. *open source* — открытый исходный код). Авторы свободных программ передают пользователю не только исполняемую программу, но и её исходный код, и предоставляют право:

- использовать программу в любых целях;
- изучать исходный код и изменять его для своих целей;
- свободно распространять программу;
- улучшать программу и распространять измененные версии на тех же условиях.

Программное обеспечение, которое не удовлетворяет этим критериям, называется собственническим или **проприетарным** (англ. *proprietary* — частное).

В первые годы на свободное ПО никакие документы не оформлялись, но возникла необходимость защищать права авторов юридически. Поэтому сейчас свободное ПО чаще всего распространяется под **лицензией GPL** (англ. *General Public Licence* — «генеральная общественная лицензия»). В ней перечислены приведённые выше права и установлено одно ограничение: разрешается распространять изменённую версию только как свободное ПО (запрещается делать его несвободным).

К свободному программному обеспечению относится операционная система Linux, браузер Mozilla Firefox, почтовая программа Mozilla Thunderbird, графические редакторы Gimp и Inkscape, архиватор 7ZIP (www.7-zip.org), среда для быстрой разработки программ Code::Blocks и многие другие программы.

Как ни странно, свободное ПО может приносить прибыль. Например, некоторые фирмы оказывают платную техническую поддержку по развёртыванию и настройке системы Linux. Второй вариант — предоставление коммерческой лицензии в том случае, если открытый исходный код используется в коммерческих программах.

Вопросы и задания



1. Что обозначает термин «авторские права»?
2. Что такое интеллектуальная собственность?
3. Какие законы Российской Федерации регулируют вопросы, связанные с авторскими правами?
4. На какие творческие результаты распространяются авторские права?
5. Что не охраняется авторским правом?
6. Что означает положение «охраняется форма, а не содержание»?
7. Нужно ли регистрировать авторское право?
8. Кто может быть правообладателем согласно российским законам?
9. Как обозначается авторское право в документе?
10. Как действует авторское право после смерти автора?
11. Какие личные и имущественные права имеет автор?
12. Какие наказания предусмотрены за нарушение авторских прав?
13. Программист разработал программу SuperPuper в 2010 г. Как он должен правильно обозначить в тексте программы своё авторское право?
14. Что такое лицензионное соглашение?
15. Что обычно может делать пользователь программы, не спрашивая дополнительного разрешения автора?
16. Что такое лицензионная программа?
17. Что такое контрафактная программа?
18. Какие типы лицензий на ПО сейчас существуют?
19. Что такое условно-бесплатная программа? Какие ограничения она может иметь?
20. Какие программы относятся к бесплатным (freeware)?
21. Что такое свободное ПО? Почему оно распространяется по лицензии?
22. Какие свободы предоставляет пользователю лицензия GPL?
23. Какие ограничения предусматривает лицензия GPL?
24. Что такое собственническое (проприетарное) ПО? Чем оно отличается от коммерческого?
25. Как можно сделать разработку свободного ПО коммерчески выгодным?
26. Какие типы ПО можно законно загружать из Интернета?
27. Можно ли, не спрашивая автора (правообладателя):
    а) скопировать картинку с веб-страницы на свой компьютер;
    б) послать скопированную картинку другу;
    в) разместить на своем сайте отсканированную книгу;
    г) привести на сайте цитату из книги с указанием источника;
    д) разместить на своем сайте картинку с другого сайта?
28. Можно ли размещать в Интернете, не спрашивая авторов:
    а) произведения А. С. Пушкина;
    б) звукозаписи популярных исполнителей;
    в) документы, принятые Государственной думой;
    г) описание алгоритма решения квадратного уравнения;
    д) базу данных мобильных телефонов?

Подготовьте сообщение

а) «Лицензия GPL»
б) «Свободное ПО: за и против»
в) «Авторское право в России и за рубежом»
г) «Как зарегистрировать программу?»
д) «Как доказать авторское право?»

Практические работы к главе 6

Работа № 14 «Использование возможностей текстовых процессоров»
Работа № 15 «Использование возможностей текстовых процессоров»
Работа № 16 «Оформление рефератов»
Работа № 17 «Оформление математических текстов»
Работа № 18 «Знакомство с настольной издательской системой Scribus»
Работа № 19 «Знакомство с аудиоредактором Audacity»
Работа № 20 «Знакомство с видеоредактором»
Работа № 21 «Сканирование и распознавание текста»
Работа № 22 «Инсталляция программ»

ЭОР к главе 6 на сайте ФЦИОР (http//fcior.edu.ru)

- Классификация ПО
- Векторный редактор
- Пакеты прикладных программ. Характеристика классов пакетов прикладных программ
- Основные функции и состав операционной системы
- Основные элементы интерфейса и управления
- Классификация языков программирования. Компиляторы и интерпретаторы
- Развитие языков программирования
- Установка на диск прикладных программ
- Установка на диск прикладных программ. Обслуживание дисков
- Законодательство РФ «Об информации, информационных технологиях и о защите информации»

Самое важное в главе 6

- Программное обеспечение необходимо для работы современного компьютера.
- Пользователи решают свои задачи с помощью прикладных программ.
- Программисты пишут программы с помощью систем программирования.
- Операционная система — это комплекс программ, который обеспечивает взаимодействие пользователя и прикладных программ с аппаратными средствами компьютера.
- Драйвер — это программа, которая обеспечивает работу внешнего устройства или контроллера.
- По типу лицензии различают коммерческое, условно-бесплатное, бесплатное и свободное программное обеспечение.

Глава 7
Компьютерные сети

§ 44
Основные понятия

Что такое компьютерная сеть?

Компьютерная сеть — это группа компьютеров, соединённых линиями связи.

Для передачи данных между компьютерами могут использоваться:

- телефонная линия;
- специальные электрические кабели;
- оптоволокно (нить из стекла или пластика, по которой идёт свет);
- радиоволны (в беспроводных сетях).

Объединяя компьютеры в сеть, мы получаем следующие *преимущества*:

- быстрый обмен данными между компьютерами без использования сменных носителей (CD- и DVD-дисков, флэш-дисков);
- совместное использование ресурсов:
  - — общих данных, которые могут быть размещены на одном компьютере;
  - — программ, которые могут запускаться с другого компьютера;
  - — внешних устройств (например, все компьютеры в сети могут использовать один принтер);
- электронную почту и другие способы сетевого общения (чаты, форумы и т. п.).

В то же время существуют и *недостатки*:

- необходимы денежные затраты на сетевое оборудование (кабели, вспомогательные устройства) и программное обеспечение (например, операционную систему специального типа);

- снижается безопасность данных, поэтому компьютеры, на которых ведутся секретные разработки, не должны быть подключены к сети;
- для настройки сети и обеспечения её работы необходим высококвалифицированный специалист — системный администратор.

Системный администратор обычно решает следующие задачи:

- устанавливает и настраивает программное обеспечение;
- устанавливает права доступа пользователей к ресурсам сети;
- обеспечивает защиту информации;
- предотвращает потерю данных в случае сбоя электропитания;
- периодически делает резервные копии данных на DVD-дисках, съёмных жёстких дисках;
- устраняет неисправности в сети.

Какие бывают сети?

По «радиусу охвата» обычно выделяют следующие типы компьютерных сетей:

- **персональные сети** (англ. **PAN** — *Personal Area Network*), объединяющие устройства одного человека (сотовые телефоны, карманные компьютеры, смартфоны, ноутбук и т. п.) в радиусе не более 30 м; самый известный стандарт таких сетей — **Bluetooth**;
- **локальные сети** (англ. **LAN** — *Local Area Network*), объединяющие, как правило, компьютеры в пределах одного или нескольких соседних зданий;
- **корпоративные сети** (англ. *Corporate network*) — сети компьютеров одной организации (возможно, находящиеся в разных районах города или даже в разных городах);
- **городские сети** (англ. **MAN** — *Metropolitan Area Network*), объединяющие компьютеры в пределах города;
- **глобальные сети** (англ. **WAN** — *Wide Area Network*), объединяющие компьютеры в разных странах; типичный пример глобальной сети — **Интернет**.

Серверы и клиенты

В любой сети одни компьютеры используют ресурсы других. Для описания роли компьютеров в обмене данными вводят два термина: сервер и клиент.

Сервер — это компьютер, предоставляющий свои ресурсы (файлы, программы, внешние устройства и т. д.) в общее использование.
Клиент — это компьютер, использующий ресурсы сервера.

Обычно серверы — это специально выделенные мощные компьютеры, которые используются только для обработки запросов большого числа клиентских компьютеров (**рабочих станций**) и, как правило, включены постоянно. Чаще всего они находятся в отдельных помещениях, куда пользователи не имеют доступа; это повышает защищённость данных.

В крупных локальных сетях используют несколько серверов, каждый из которых решает свою задачу:

- *файловый сервер* хранит данные и обеспечивает доступ к ним;
- *сервер печати* обеспечивает доступ к общему принтеру;
- *почтовый сервер* управляет электронной почтой;
- *серверы приложений* (например, серверы баз данных) выполняют обработку информации по запросам клиента.

Сервер получает запросы от клиентов, ставит их в очередь и после выполнения посылает каждому клиенту ответ с результатами выполнения запроса. Задача клиента — послать серверу запрос в определённом формате и после получения ответа вывести ответ на монитор пользователя. Такая технология называется **«клиент — сервер»**. Её используют, например, все веб-сайты в Интернете: программа-браузер (клиент) посылает запрос веб-серверу и выводит его ответ (веб-страницу) на экран.

Часто понятия «сервер» и «клиент» относятся не к компьютерам, а к программам. Например, на одном и том же компьютере может работать веб-сервер (программа, которая отправляет веб-страницы по запросу пользователей) и почтовый клиент (программа, которая обращается к почтовому серверу на другом компьютере для отправки и получения сообщений электронной почты).

Обмен данными

Для того чтобы люди могли полноценно общаться, нужно, чтобы они говорили на одном языке. Эта аналогия действует и для компьютерных систем, где вместо слова «язык» используется термин «протокол».

Протокол — это набор правил и соглашений, определяющих порядок обмена данными в сети.

Можно объединить в одну сеть устройства, которые используют разные протоколы обмена данными. Для этого требуется устройство-«переводчик», которое называют **шлюзом**. Задача шлюза — «перевести» принятые данные в формат другого протокола. Шлюзы часто используются для связи между промышленными сетями (измерительной аппаратурой, датчиками) и сетями персональных компьютеров.

В современных сетях пересылаемые данные делятся на **пакеты**. Дело в том, что чаще всего одна линия связи используется для обмена данными между несколькими узлами. Если передавать большие файлы целиком, то получится, что сеть будет заблокирована, пока не закончится передача файла. Кроме того, в этом случае при сбое весь файл нужно передавать заново, это увеличивает нагрузку на сеть.

Если передавать отдельные пакеты, время ожидания сокращается до времени передачи одного пакета (это доли секунды), нагрузка на линию связи становится более равномерной. По сети одновременно передаются пакеты, принадлежащие нескольким файлам. На рисунке 7.1 по одной линии связи (между узлами 3 и 4) одновременно выполняется передача данных от узла 2 к узлу 5

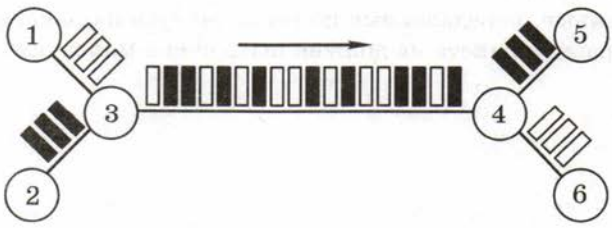


Рис. 7.1

(пакеты обозначены чёрными прямоугольниками) и от узла 1 к узлу 6 (белые прямоугольники).

Вместе с каждым пакетом передаётся его **контрольная сумма** — число, найденное по специальному алгоритму и зависящее от всех данных пакета. Узел-приёмник рассчитывает контрольную сумму полученного блока данных и, если она не сходится с контрольной суммой, указанной в пакете, фиксируется ошибка и этот пакет (а не весь файл!) передаётся, как правило, ещё раз.

Казалось бы, чем меньше размер пакета, тем лучше. Однако это не так, потому что любой пакет кроме «полезных» данных содержит служебную информацию: адреса отправителя и получателя, контрольную сумму. Поэтому в каждом случае есть некоторый оптимальный размер пакета, который зависит от многих условий (например, от уровня помех, количества компьютеров в сети, передаваемых данных и т. д.). Чаще всего для обмена данными в локальных сетях и в Интернете используются пакеты размером не более 1,5 Кбайт.

Вопросы и задания

1. Что такое компьютерная сеть?
2. Какие каналы связи могут использоваться в сетях?
3. Какие преимущества даёт объединение компьютеров в сеть? Что при этом ухудшается?
4. Что входит в обязанности системного администратора?
5. Как разделяются сети по области действия?
6. Что такое персональные сети?
7. Что такое сервер и клиент?
8. Может ли один компьютер выполнять роли сервера и клиента?
9. Что такое протокол? Зачем нужны протоколы?
10. Что такое шлюз?
11. Зачем данные, передаваемые по сети, делятся на пакеты?
12. Почему размер пакета не должен быть очень маленьким?

§ 45
Структура (топология) сети

Для обмена данными в сети очень важно, как именно связаны компьютеры линиями связи. В этом параграфе мы кратко рассмотрим три основные структуры (**топологии**) сетей с разными схемами соединений между компьютерами: общую шину, звезду и кольцо. Каждая из них обладает своими достоинствами и недостатками.

Общая шина

Шина — это линия связи, которую несколько устройств используют для обмена данными. В схеме **«общая шина»** (рис. 7.2) компьютеры (рабочие станции) подключены к одному кабелю с помощью специальных разъёмов.

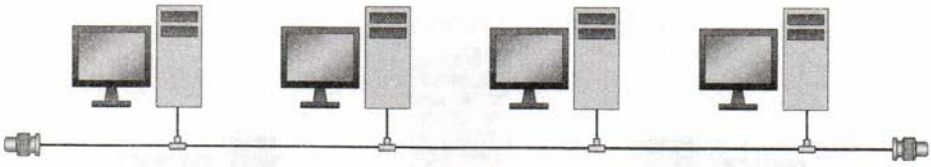


Рис. 7.2

Чтобы сигнал не отражался от концов кабеля (и не шёл в обратную сторону), их закрывают заглушками (**терминаторами**).

Так как существует всего одна линия связи, компьютеры передают данные по очереди. Сигнал, который идёт по шине, получают все компьютеры, но каждый из них обрабатывает только те данные, которые ему предназначены.

Достоинства схемы «общая шина»:

- самая простая и дешёвая схема;
- небольшой расход кабеля;
- легко подключать новые рабочие станции;
- при выходе из строя любого компьютера сеть продолжает работать.

Недостатки:

- при разрыве кабеля или выходе из строя терминатора вся сеть не работает;

- низкий уровень безопасности (каждая рабочая станция имеет доступ ко всем данным, которые идут по сети);
- один канал связи на всех (при увеличении числа компьютеров падает скорость передачи; возможны конфликты, когда две рабочие станции хотят передать данные одновременно);
- сложно обнаруживать неисправности (неясно, где проблема);
- ограничение размера (не более 185 м, при большей длине нужны усилители сигнала).

Звезда

В схеме **«звезда»** (рис. 7.3) есть центральное устройство, через которое идёт весь обмен данными. На практике чаще всего в центре находится **коммутатор** (его часто называют «свитч», от англ. *switch* — переключать). Коммутатор передает принятый пакет только адресату, а не всем компьютерам в сети.

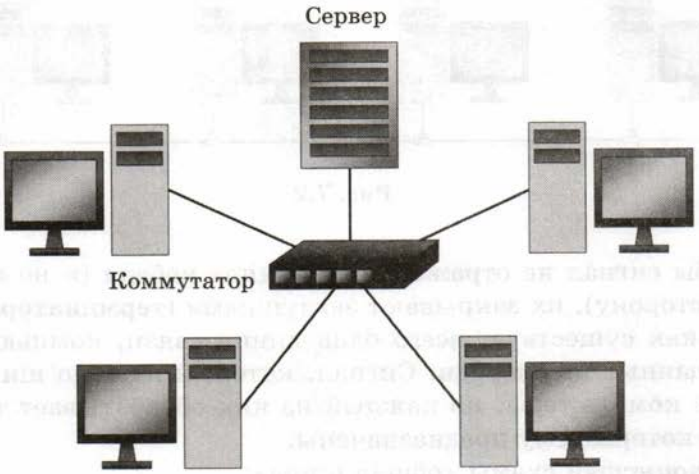


Рис. 7.3

Достоинства схемы «звезда»:

- при выходе из строя любой рабочей станции сеть остаётся работоспособной;
- высокий уровень безопасности (каждая рабочая станция получает только «свои» данные, а не все, что передаются по сети);

- простой поиск неисправностей и обрывов (сразу ясно, с каким компьютером нет связи).

Недостатки:

- большой расход кабеля, высокая стоимость;
- при выходе из строя коммутатора вся сеть не работает;
- количество рабочих станций ограничено количеством портов коммутатора.

Многоуровневая схема «звезда» (рис. 7.4) представляет собой иерархическую структуру (дерево). Она нередко применяется в крупных сетях, которые состоят из сотен компьютеров.

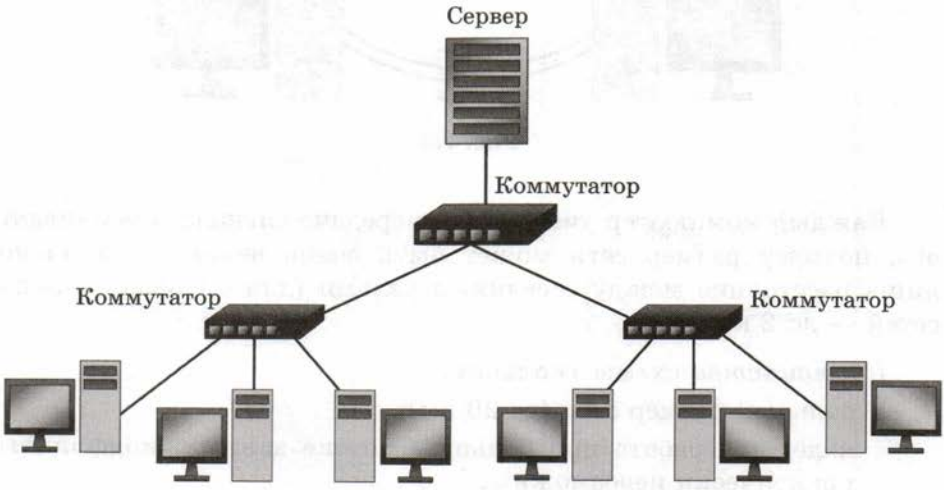


Рис. 7.4

Кольцо

В схеме **«кольцо»** (рис. 7.5) каждый компьютер соединён с двумя соседними, причём от одного он только получает данные, а другому только передаёт. Таким образом, пакеты движутся по кольцу в одном направлении. Для повышения надёжности обычно используют «двойное кольцо», в котором каждая линия связи дублируется. По второму кольцу данные могут передаваться в обратном направлении.

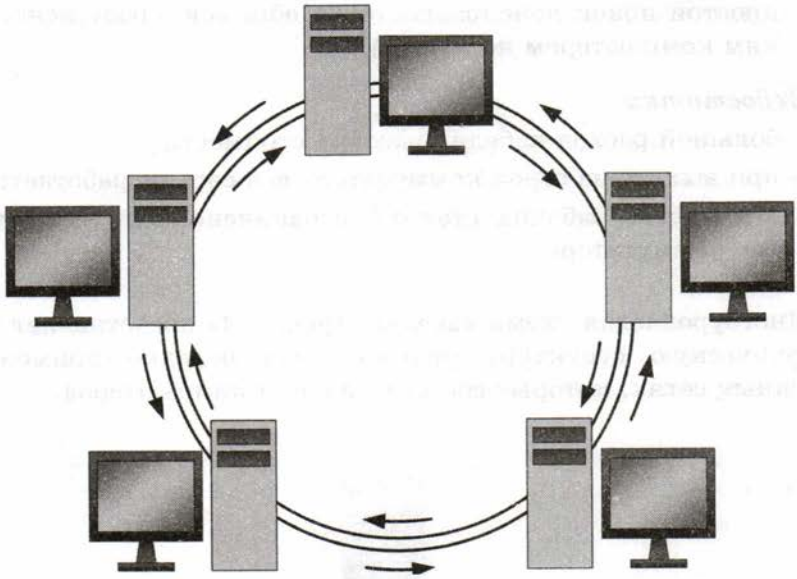


Рис. 7.5

Каждый компьютер участвует в передаче сигнала и усиливает его, поэтому размер сети может быть очень велик, ограничено лишь расстояние между соседними узлами (для оптоволоконных сетей — до 2 км).

Достоинства схемы «кольцо»:

- большой размер сети (до 20 км);
- надёжная работа при большом потоке данных, конфликты практически невозможны;
- не нужно дополнительное оборудование (коммутаторы).

Недостатки:

- для подключения нового узла нужно останавливать сеть;
- низкая безопасность (все данные проходят через каждый компьютер);
- сложность настройки и поиска неисправностей.

В современных сетях кольцевая схема чаще всего используется в сочетании со «звездой»: компьютеры соединяются с коммутатором по схеме «звезда», а коммутаторы между собой объединяются в кольцо.

Вопросы и задания

1. Что такое топология сети?
2. Опишите структуру, достоинства и недостатки сетей типа общая шина, звезда и кольцо.
3. Какую структуру вы предложили бы использовать для школьной сети (рассмотрите разные ситуации)?

§ 46 Локальные сети

Типы локальных сетей

Локальная сеть объединяет компьютеры в одном или нескольких соседних зданиях.

Для того чтобы организовать локальную сеть, нужно использовать **сетевую операционную систему**, которая поддерживает:

- сетевое оборудование (например, сетевые карты);
- сетевые протоколы обмена данными;
- доступ к удалённым ресурсам (папкам, принтерам и т. п.).

Эти возможности существуют почти во всех современных ОС (Windows, Linux, Mac OS и др.).

В небольших организациях часто используют **одноранговые сети** (на 10–15 компьютеров), в которых все компьютеры равноправны, каждый может выступать как в роли клиента, так и в роли сервера. Пользователь может открыть общий доступ к некоторым ресурсам своего компьютера (папкам, принтерам), т. е. предоставить их в совместное использование. На каждый общий ресурс можно установить пароль для чтения и/или записи.

Достоинства одноранговых сетей:

- низкая стоимость;
- простота настройки и обслуживания;
- независимость компьютеров друг от друга;
- не нужно сложное программное обеспечение.

Недостатки:

- сложность управления и настройки прав доступа (нет единого центра, нужно создавать учётные записи для всех пользователей на каждом компьютере);
- низкая защищённость данных;
- резервное копирование данных нужно выполнять на каждом компьютере.

С увеличением количества компьютеров становится очень сложно управлять одноранговыми сетями, а также обеспечивать безопасность данных. Поэтому в крупных организациях используют **сети с выделенными серверами**, в которых один или несколько мощных компьютеров играют роль серверов (пользователи на них не работают), а остальные (клиенты) используют их ресурсы. Такие сети обладают серьёзными *достоинствами*:

- основная обработка данных выполняется на серверах;
- через сеть передаются только нужные данные;
- упрощается модернизация системы: достаточно переоборудовать серверы;
- повышенный уровень безопасности: права на доступ к данным устанавливаются на сервере;
- клиенты могут использовать различное оборудование и операционные системы;
- резервное копирование данных нужно выполнять только на серверах.

В то же время есть *недостатки*:

- высокая стоимость серверного оборудования;
- сложность настройки и обслуживания сервера;
- при выходе сервера из строя служба, которую он обеспечивал, не работает (например, недоступны хранящиеся на нём данные).

Для поддержки сервера во многих случаях требуется специальная **серверная операционная система (Windows Server, Linux, FreeBSD, Solaris)**. В таких ОС основное внимание уделяется стабильной и надежной работе с большим количеством клиентов, а не пользовательскому интерфейсу. Они содержат развитые средства для поддержки совместной работы пользователей, веб-узла, электронной почты, систем управления базами данных и т. п.

Важная возможность сетевых ОС — **терминальный доступ**, при котором пользователь со своей рабочей станции запускает программу на сервере и получает на своём экране результаты её работы.

Беспроводные сети

Беспроводные сети используются там, где создание кабельной сети невозможно или невыгодно, например за пределами зданий, в исторических помещениях и т. п. Кроме того, с их помощью мобильные компьютеры (ноутбуки, карманные компьютеры) могут легко подключаться к сети и получать выход в Интернет. Для обмена данными применяются радиоволны сверхвысокой частоты.

Самый популярный стандарт для **беспроводных персональных сетей** — **Bluetooth**, обеспечивающий обмен данными между 8 устройствами. Это могут быть карманный и обычный компьютер, мобильный телефон, ноутбук, принтер, цифровой фотоаппарат, мышь, клавиатура, наушники. Радиус действия такой сети обычно[1] составляет не более 20 м, он зависит от мощности передатчиков, а также от преград и помех. Максимальная скорость обмена данными — около 700 Кбит/с. Ожидается, что в будущем с помощью Bluetooth можно будет связывать любые электронные устройства, включая холодильники и стиральные машины. Среди достоинств Bluetooth — низкая стоимость, удобство и простота в использовании, высокая надёжность. Для обеспечения защиты данных от перехвата приёмник и передатчик 1600 раз в секунду одновременно меняют частоту сигнала.

В **локальных беспроводных сетях** применяют стандарт **Wi-Fi** (от англ. *Wireless Fidelity* — беспроводная точность). Для объединения компьютеров в беспроводную сеть чаще всего используют специальное устройство — **точку доступа** (англ. **WAP** — *Wireless Access Point* — точка беспроводного доступа). К одной точке доступа обычно подключаются не более 15 компьютеров (при увеличении этого количества падает скорость передачи данных). Часто главная задача точки доступа — обеспечить мобильным компьютерам доступ к кабельной сети и выход в Интернет (рис. 7.6).

[1] Стандарт предусматривает радиус действия до 100 м.

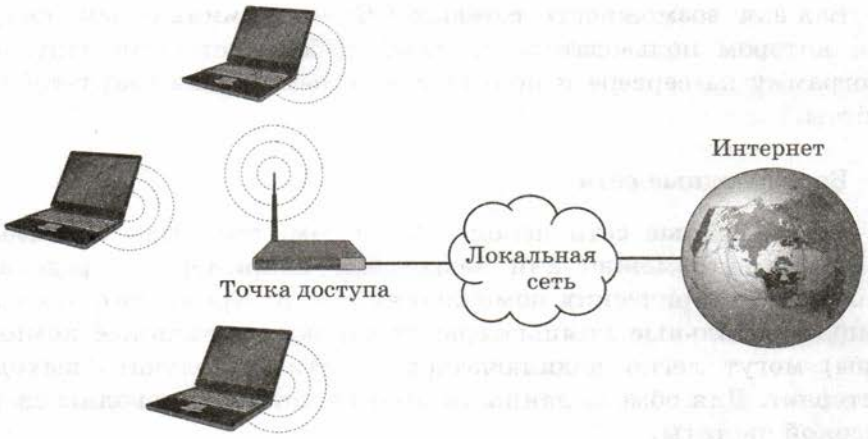


Рис. 7.6

Современные операционные системы (Windows, Mac OS, Linux) поддерживают технологию и устройства Wi-Fi.

Согласно стандарту, скорость передачи данных в сети Wi-Fi может быть от 0,1 Мбит/с до 480 Мбит/с. Обычно радиус действия сети Wi-Fi в помещениях не превышает 45 м, а вне зданий — 450 м.

Технология Wi-Fi широко используется как в офисах, так и в домашних сетях. Бесплатный выход в Интернет через Wi-Fi предоставляют многие библиотеки, университеты, кафе (для привлечения посетителей), гостиницы. Такие зоны доступа называют «хот-спот» (от англ. *hot spot* — горячая точка). В некоторых гостиницах и аэропортах эта услуга платная.

Сети Wi-Fi работают в радиоэфире, так что любое приёмное устройство, настроенное на нужную частоту, может перехватить сигнал. Поэтому в беспроводных сетях важно обеспечить защиту данных. Для этого используют специальные алгоритмы кодирования сигналов, шифрование и другие методы.

Сетевое оборудование

В современных кабельных локальных сетях используется технология пакетной передачи данных, которая называется **Ethernet** (от лат. *aether* — эфир). Для связи компьютеров могут применяться электрические кабели или оптоволокно. Существующий

стандарт определяет скорости передачи данных 10 Мбит/с, 100 Мбит/с, 1 Гбит/с и 10 Гбит/с.

Для подключения компьютера к кабельной сети он должен иметь **сетевую карту (сетевой адаптер**, англ. *network interface card*, рис. 7.7). В современных материнских платах настольных компьютеров и в ноутбуках обычно уже есть встроенная сетевая карта, поддерживающая стандарт Ethernet со скоростью до 1 Гбит/с.



Рис. 7.7. Сетевая карта

Для локальных сетей чаще всего используется восьмижильный кабель «витая пара» (рис. 7.8), который представляет собой четыре пары скрученных проводов. Восьмиконтактный разъём с защёлкой часто называют RJ-45 (рис. 7.9). Сейчас наиболее распространены сети со скоростью 100 Мбит/с, построенные с помощью кабеля «витая пара» по схеме «звезда» (с коммутатором в центре).



Рис. 7.8. Сетевой кабель «витая пара»



Рис. 7.9. Разъём RJ-45

Для передачи данных на большие расстояния применяют **оптоволоконные кабели**, в которых данные передаются с помощью светового луча. Свет идет внутри кабеля, отражаясь от стенок стеклянного или пластикового цилиндра-световода.

Коммутаторы (свитчи) (рис. 7.10) используются для объединения компьютеров в единую сеть по схеме «звезда», которая чаще всего применяется на практике.



Рис. 7.10. Коммутаторы

В отличие от **концентраторов («хабов»**, от англ. *hub*) коммутаторы передают пакет только тому узлу, которому он предназначен, а не дублируют на все выходы. Компьютер соединяют с коммутатором отрезком кабеля с двумя разъёмами RJ-45, который называется **«патч-корд»** (от англ. *patching cord* — соединительный шнур).



Рис. 7.11. USB-адаптер Wi-Fi



Рис. 7.12. Точка доступа



Рис. 7.13. Беспроводной маршрутизатор

Обычно все компьютеры, входящие в локальную сеть, получают доступ в Интернет через один канал связи. Для связи локальной сети с Интернетом необходим **маршрутизатор**, или **роутер** (англ. *router*). Задача маршрутизатора — определить дальнейший маршрут движения пакета и направить его на нужный выход (порт). Для этого используются **таблицы маршрутизации**, в которых записано, куда направлять пакеты в зависимости от адреса назначения. Роль маршрутизатора в локальной сети может выполнять обычный компьютер с несколькими сетевыми картами. С точки зрения компьютеров локальной сети, маршрутизатор является **шлюзом** — устройством, которое связывает две сети — локальную и глобальную.

При создании беспроводных сетей компьютеры должны иметь **адаптеры Wi-Fi**, они обычно встроены в современные портативные компьютеры. Если встроенного адаптера нет, можно использовать дополнительный адаптер, который подключается к USB-порту (рис. 7.11). Для связи компьютеров в беспроводной сети и для обеспечения доступа в Интернет используют точки доступа (рис. 7.12) и беспроводные маршрутизаторы (рис. 7.13).

Вопросы и задания

1. Что такое локальная сеть?
2. В каких случаях лучше использовать сеть с выделенными серверами?
3. Какие задачи решают компьютеры-серверы?
4. Чем отличаются серверные ОС от клиентских?
5. Какими возможностями должны обладать сетевые операционные системы?
6. Что такое терминальный доступ?
7. Что такое точка доступа? Что такое зона доступа Wi-Fi?
8. В каких случаях применяются стандарты беспроводных сетей Bluetooth и Wi-Fi?
9. Как обеспечивается защита данных в беспроводных сетях?
10. Какие оборудование необходимо для создания беспроводной сети? Назовите преимущества и недостатки беспроводных сетей.
11. Какое сетевое оборудование необходимо для кабельных сетей?
12. Что такое коммутатор?
13. Что такое патч-корд?
14. Что такое маршрутизатор? Какую роль он выполняет в локальной сети?

Подготовьте сообщение:

а) «Серверные операционные системы»
б) «Стандарт Ethernet»
в) «Сети Wi-Fi»
г) «Защита данных в беспроводных сетях»

§ 47
Сеть Интернет

Что такое Интернет?

Слово **Интернет** (англ. *Internet*), обозначающее глобальную компьютерную сеть, возникло как сокращение Interconnected Networks — «объединённые сети» или «сеть сетей». В отличие от локальных сетей, элементы глобальной сети — не отдельные компьютеры, а сети.

Информация в Интернете хранится на **серверах**, связанных скоростными линиями связи (оптоволоконными, спутниковыми). Практически все услуги Интернета основаны на использовании **технологии «клиент — сервер»**: программа-клиент на компьютере пользователя запрашивает данные, сервер возвращает ответ.

Пользователь получает доступ к глобальной сети через **провайдера** — фирму, локальная сеть которой непосредственно связана с Интернетом. Существует несколько **способов подключения** к провайдеру:

- с помощью **ADSL-модема**, который использует телефонную линию, но позволяет одновременно разговаривать по телефону и работать в Интернете; скорость передачи данных из Интернета к пользователю может достигать 25 Мбит/с, однако на телефонной станции необходимо устанавливать дополнительное оборудование (**сплиттер**, отделяющий низкочастотный телефонный сигнал от высокочастотного сигнала, передающего цифровые данные);
- через **локальную сеть провайдера** (если она существует в вашем доме); в этом случае телефонная линия не задействована;
- с помощью **беспроводных модемов** (USB-модемов, рис. 7.14), которые используют сети сотовых операторов и работают везде, где доступна мобильная связь; скорость передачи данных для сетей 3-го поколения (англ. **3G** — *3rd generation*) достигает 10 Мбит/с, а в сетях 4-го поколения (4G) — до 1 Гбит/с.



Рис. 7.14

Краткая история

В 1960-е гг. в Министерстве обороны США начали разработку компьютерной системы передачи данных, которая получила название **ARPANET** (англ. *Advanced Research Projects Agency*

Network — сеть агентства передовых исследований). В основу этого проекта были положены следующие идеи:

- сеть объединяет компьютеры, имеющие разное аппаратное и программное обеспечение;
- при подключении новой сети не требуется переделка существующей части;
- нет единого центра (такая сеть называется *распределённой*), это обеспечивает «живучесть» в случае выхода из строя любого узла;
- пакетная передача данных: передаваемые данные разбиваются на пакеты небольшого размера, одна линия связи используется для одновременной передачи нескольких блоков данных.

В 1969 г. состоялся первый обмен данными по сети между компьютерами, установленными в Калифорнийском университете и Стэнфордском исследовательском центре. В 1971 г. была создана программа для работы с электронной почтой, которая сразу стала очень популярной. Начиная с 1973 г., к новой сети подключаются университеты и колледжи не только США, но и Европы. В 1983 г. сеть разделяется на две части: военную сеть **MilNet** и общедоступную сеть, которая получила название **Интернет**.

История российского Интернета начинается с 1990 г., когда была организована **почтовая сеть «Релком»** — первый провайдер в Советском Союзе.

В 1991 г. британский ученый *Тим Бернес-Ли* разработал систему обмена данными в виде **гипертекста** — текста с активными ссылками на другие документы. Сейчас она называется **Всемирной паутиной** (англ. **WWW** — *World Wide Web*) и является самой мощной службой Интернета. Многие ошибочно считают, что Интернет и Всемирная паутина — это одно и то же. На самом деле это не так, потому что в Интернете есть и другие службы — электронная почта, обмен файлами, чаты, форумы и т. д.



Т. Бернес-Ли

Протоколы

Вы уже знаете, что для передачи информации источник и приёмник должны использовать один и тот же протокол — набор правил и соглашений, определяющих порядок обмена данными в сети. В Интернете в качестве стандарта принят **протокол TCP/IP**, разработанный в 1974 г. На самом деле, это не один протокол, а целое семейство, название которого происходит от двух самых важных протоколов — **TCP** (англ. *Transfer Control Protocol* — протокол управления передачей) и **IP** (англ. *Internet Protocol* — межсетевой протокол).

Попробуем разобраться, почему для работы в Интернете нужно использовать несколько протоколов (рис. 7.15). Предположим, что браузер на компьютере А запрашивает веб-страницу с сервера, который находится на компьютере Б. «Разговор» между браузером и сервером идет с помощью **протокола HTTP** (англ. *HyperText Transfer Protocol* — протокол передачи гипертекста). Браузер и веб-сервер не могут связаться напрямую. Чтобы послать запрос серверу, браузер передаёт адрес сервера и текст запроса операционной системе, которая вызывает драйвер протокола TCP.

Задача драйвера TCP — установить соединение с удалённым компьютером и обеспечить доставку данных. Передаваемый блок данных разбивается на пакеты (размер пакета обычно не превышает 1,5 Кбайт), и каждый пакет передаётся на следующий уровень — драйверу протокола IP, который посылает его в сеть по указанному адресу.

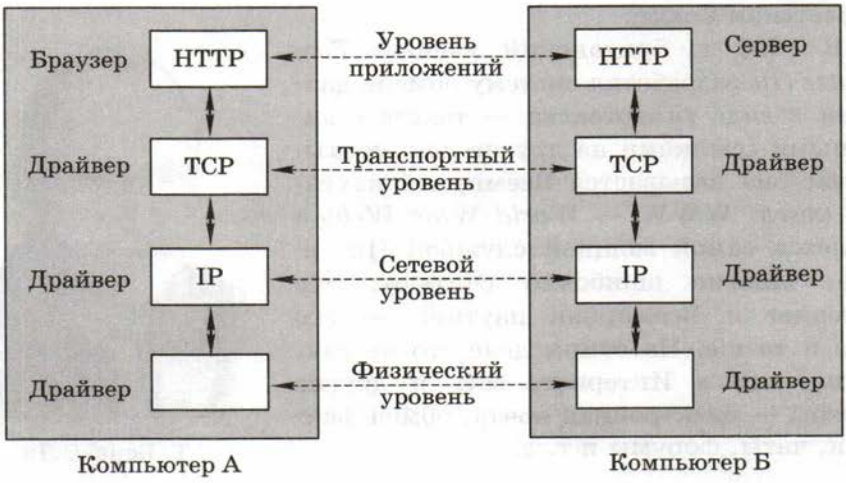


Рис. 7.15

Обычно при работе в Интернете компьютеры А и Б напрямую не связаны, поэтому задача протокола IP — определить **узел-маршрутизатор**[1], на который нужно отправить пакет, чтобы он дошёл до компьютера Б. Когда маршрут определён, пакет (с добавленной служебной информацией) передаётся на физический уровень (например, в сетевую карту), где передаётся просто как цепочка байтов. Протоколы физического уровня могут быть любыми, они не определены в стандарте.

Протокол IP не гарантирует доставку пакетов, поэтому драйвер TCP должен (с помощью установленного соединения) проверить, что данные получены, и в случае сбоя передать пакет повторно. На другом конце соединения драйвер TCP «собирает» пакеты в единый блок данных и передаёт на уровень приложения (запрос дошёл до сервера).

Таким образом, в Интернете используется четырёхуровневая система протоколов, каждый из которых «занимается своим делом»:

1) *уровень приложений* — формат запросов и ответов, которыми обмениваются программы;
2) *транспортный уровень* (TCP) — правила пакетной передачи блоков данных без учёта их содержания;
3) *сетевой уровень* (IP) — правила выбора маршрута для отдельных пакетов без гарантии их доставки;
4) *физический уровень* — правила передачи отдельных байтов по кабельной, оптоволоконной или другой линии связи.

На уровне приложений (который находится «ближе всего» к пользователю) чаще всего применяются протоколы:

HTTP	— для передачи веб-страниц;
FTP	— для передачи файлов;
SMTP	— для передачи на сервер сообщений электронной почты;
POP3 или IMAP	— для приёма сообщений электронной почты с сервера.

[1] Маршрутизаторы обмениваются информацией друг с другом, сообщая о выходе из строя или подключении каких-то участков сети. Таблицы маршрутизации обновляются автоматически, так что при выборе маршрута пакетов учитывается фактическая структура сети в данный момент.

Существуют и другие протоколы (для чатов, новостных групп и т. п.), но все они используют TCP и IP соответственно на транспортном и сетевом уровнях.

Вопросы и задания

1. Как в Интернете используется технология «клиент — сервер»?
2. Что такое «провайдер»?
3. Расскажите, как можно получить доступ в Интернет. В чем достоинства недостатки разных способов? Приведите примеры.
4. Какие идеи были положены в основу глобальной компьютерной сети?
5. Как вы думаете, почему в 1983 г. сеть разделили?
6. Что такое гипертекст?
7. Чем различаются понятия «Интернет» и «Всемирная паутина»?
8. Какое семейство протоколов используется в сети Интернет?
9. Объясните, почему применяются несколько уровней протоколов. Расскажите о роли протоколов разных уровней.
10. Какова роль узлов-маршрутизаторов?
11. Как обеспечивается гарантированная доставка сообщений в Интернете?
12. Назовите наиболее известные протоколы уровня приложений. Где они применяются?

Подготовьте сообщение

а) «Технология "клиент — сервер"»
б) «Как выбирается маршрут пакетов?»
в) «Развитие Интернета в России»
г) «Семейство протоколов TCP/IP»
д) «Тим Бернес-Ли и его вклад в развитие Интернета»

§ 48
Адреса в Интернете

IP-адреса

В Интернете любые два компьютера могут связаться друг с другом. Для этого каждый из них должен иметь уникальный адрес. С «точки зрения» компьютеров удобнее работать с числовыми адресами, каждый из которых занимает одинаковое место в памяти. Такие адреса (их называют **IP-адресами**) состоят из четырёх чисел в интервале от 0 до 255, например:

192.168.104.115

В этих числах закодированы номер сети и номер компьютера в сети. Для того чтобы выделить эти две части из IP-адреса, используют маски-шаблоны. Маска — это тоже четыре числа в интервале [0; 255], но она строится особым образом, по принципу «n единиц, потом — нули» в двоичном коде. Например, маска 255.255.255.0 в двоичном виде запишется так:

11111111.11111111.11111111.00000000

В ней сначала идут 24 единицы, а потом — нули. Это значит, что первые 24 бита адреса — номер сети (192.168.104.0), а оставшиеся 8 битов — номер компьютера (узла) в этой сети (115). Можно использовать другую запись, которая значит то же самое («/24» говорит о том, что в маске 24 единицы):

192.168.104.115/24

В такой сети может быть 254 узла, а не 256, как можно было бы ожидать. Дело в том, что младший адрес (192.168.104.0) используется для обозначения всей сети, а старший (192.168.104.255) — для так называемой широковещательной рассылки (сообщение отправляется всем компьютерам данной сети). Все узлы с адресами 192.168.104.* (здесь * — любое число от 1 до 254) находятся в той же сети, что и данный компьютер.

Тот же самый адрес с другой маской имеет совершенно иной смысл. Например, маска 255.255.255.248 в двоичной системе содержит 29 единиц и 3 нуля. На рисунке 7.16 область, отведённая номеру сети, выделена штриховой рамкой.

Адрес	192 .	168 .	104 .	115	
	11000000 .	10101000 .	01101000 .	01110	011_2
Маска	11111111 .	11111111 .	11111111 .	11111	000_2
	255 ·	255 ·	255 ·	248	

Рис. 7.16

Узел с адресом 192.168.104.115/29 — это узел номер 3 (011_2) в сети 192.168.104.112:

$192.168.104.112 = 11000000.10101000.01101000.01110000_2$

Поскольку на адрес узла отводится три бита (в маске три нуля), в такой сети доступно только $2^3 = 8$ адресов. Учитывая, что два из них специальные (номер сети и широковещательный адрес), в сеть может входить не более 6 узлов.

Существуют так называемые «серые» адреса, которые не используются в «большом» Интернете. Например, диапазоны адресов:

192.168.0.0-192.168.255.255	(192.168.0.0/16)
172.16.0.0-172.31.255.255	(172.16.0.0/12)
10.0.0.0-10.255.255.255	(10.0.0.0/8)

предназначены только для локальных сетей. Адреса 127.0.0.0 — 127.255.255.255 служат для обращения к своему компьютеру (обычно для этой цели применяют адрес 127.0.0.1).

Нужно помнить, что IP-адрес присваивается не компьютеру, а *интерфейсу* — каналу передачи данных (сетевой карте, модему). Поэтому один компьютер может иметь несколько IP-адресов, (например, если на нём установлены две сетевые карты).

В связи с бурным развитием Интернета адресов, которые можно использовать при таком кодировании, скоро не будет хватать для всех желающих. Поэтому разработана новая (шестая) версия протокола IP, которая обозначается как **IPv6**. В ней на каждый адрес отводится 128 битов, а не 32, как сейчас. Адрес IPv6 записывается в виде восьми групп по 4 шестнадцатеричных цифры, разделённых двоеточиями, например:

2001:0db8:11a3:09d7:1f34:8a2e:07a0:765d

Уже сейчас существует более 4400 сетей, где применяется IPv6; этот протокол поддерживается всеми современными операционными системами и производителями оборудования. Полный переход на IPv6 займёт несколько лет, он потребует больших денежных затрат и замены всех устаревших устройств.

Доменные имена

В отличие от компьютеров человеку неудобно работать с числовыми адресами. Они плохо запоминаются, при вводе IP-адреса легко сделать ошибку, а заметить её достаточно сложно. Поэтому в 1984 г. была разработана **система доменных имён** (англ. **DNS** — *Domain Name System*), которая позволила использовать символьные имена сайтов, например www.mail.ru.

Домен (англ. *domain* — область, район) — это группа символьных адресов в Интернете. Домены образуют многоуровневую структуру (иерархию, дерево), вкладываются друг в друга, как матрёшки (рис. 7.17).

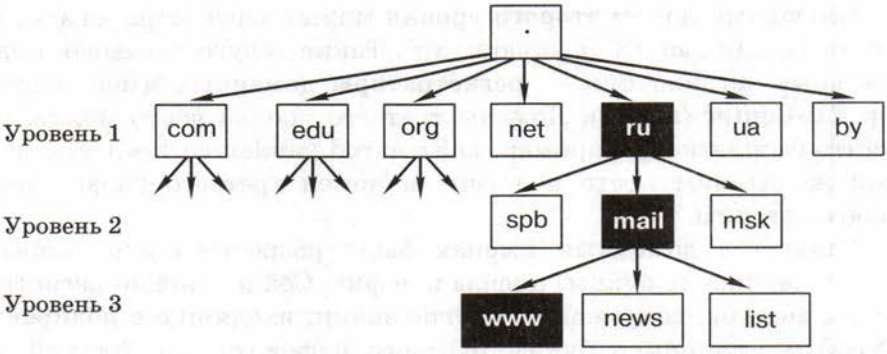


Рис. 7.17

Чем-то такая система напоминает почтовый адрес, в котором указывается страна, город, улица, дом, квартира.

Точка в корне дерева — это **корневой домен. Домены первого уровня** (они называются **доменные зоны**) могут обозначать тип организации, например[1]:

com	— коммерческие организации;
edu	— образовательные организации (университеты, колледжи);
gov	— правительство США;
biz	— бизнес;
info	— информационные сайты;
name	— личные сайты;
museum	— музеи;
net	— сетевые организации;
org	— разные организации.

Кроме того, каждая страна имеет свой двухбуквенный домен первого уровня.

Распределением IP-адресов и доменов первого уровня занимается международная организация **ICANN** (англ. *Internet Corporation for Assigned Names and Numbers*). Российский домен **ru** был зарегистрирован в 1994 году.

[1] Здесь перечислены не все тематические домены первого уровня.

Свободный **домен второго уровня** может зарегистрировать любой желающий за небольшую плату. Такие услуги оказывают специальные организации — **регистраторы доменных имён**, например **RU-Center** (nic.ru). Домены третьего уровня часто можно получить бесплатно. Например, сайт narod.yandex.ru предоставляет всем желающим место под сайт и домен третьего уровня вида ivanov.narod.ru.

Раньше в доменных именах было разрешено использовать только латинские буквы, цифры и дефис. Сейчас можно регистрировать домены, содержащие другие знаки, входящие в кодировку UNICODE, например буквы русского алфавита. За Россией закреплён домен **рф**, в котором все желающие могут зарегистрировать домены второго уровня.

Таким образом, сейчас в Интернете используются две системы адресов: IP-адреса и доменные имена. Чтобы установить соответствие между ними, на специальных серверах, которые называются **DNS-серверами**, хранятся таблицы, состоящие из пар «IP-адрес — доменное имя». Их задача — по запросу компьютера-клиента вернуть IP-адрес для заданного доменного имени (или наоборот).

Для того чтобы компьютер смог установить связь с сетью, в настройках сетевой карты (или модема) указывается IP-адрес, маска сети и адрес DNS-сервера. Иногда эти данные определяются автоматически при подключении к сети провайдера.

Когда вы вводите адрес сайта (доменное имя) в адресной строке браузера, сначала отправляется запрос на DNS-сервер, цель которого — определить IP-адрес сервера. Если это удалось, направляется запрос на получение веб-страницы, причем драйвер протокола IP использует полученный IP-адрес, а не доменное имя.

Заметим, что одному доменному имени может соответствовать несколько IP-адресов. Такой приём применяется для распределения нагрузки на сайты с большим количеством посетителей (например, www.yandex.ru, www.google.com). Таким образом, соответствие между доменными именами и IP-адресами можно описать как «многие ко многим»: с одним IP-адресом может быть связано несколько доменных имён и наоборот.

Адрес ресурса (URL)

Точный адрес имеет не только каждый компьютер в Интернете, но и каждый документ. Для такого адреса чаще всего используется английское сокращение **URL** — Uniform Resource Locator —

универсальный указатель ресурса. Типичный URL-адрес состоит из четырёх частей: протокола, имени сервера (или его IP-адреса), каталога и имени документа (файла). Такую систему записи придумал в 1990 г. создатель Всемирной паутины Бернес-Ли. Например, адрес

http://example.com/doc/new/vasya-new.htm

включает:

1) протокол HTTP — протокол для обмена гипертекстовыми документами (это веб-страница);
2) доменное имя сервера example.com;
3) каталог на сервере /doc/new;
4) имя файла vasya-new.htm.

Иначе говоря, для обращения к документу vasya-new.htm, который находится в каталоге /doc/new на сервере example.com, нужно использовать протокол HTTP.

Иногда каталог и имя файла не указывают, например: http://example.com. Это означает, что мы обращаемся к главной странице сайта. Она может иметь разные имена, в зависимости от настроек сервера (чаще всего — index.htm, index.html, index.php).

Для скачивания и загрузки файлов часто применяется протокол FTP, тогда адрес документа выглядит примерно так:

ftp://files.example.com/pub/new/vasya-new.zip

Тестирование сети

При работе с сетью возникает несколько характерных задач, связанных с проверкой доступности компьютеров и правильности работы службы DNS. Для этой цели администраторы используют утилиты, работающие из командной строки. В Linux для работы в командной строке нужно запустить программу **Терминал (Konsole)**, а в Windows — командный процессор **cmd**.

Определим IP-адрес и настройки своего компьютера. Для этого в Windows используется команда `ipconfig`, результат работы которой может быть, например, таким:

```
Подключение по локальной сети – Ethernet адаптер:
 IP-адрес:        192.168.45.48
 Маска подсети:   255.255.255.0
 Основной шлюз:   192.168.45.5
```

Последняя строка показывает **адрес шлюза** — узла, на который отправляются все пакеты, в которых указан IP-адрес получателя, не входящий в локальную сеть (в данном случае — в сеть 192.168.45.0/24). В операционной системе Linux (и других Unix-подобных системах) для той же цели используется команда `ifconfig`[1].

Команда `ping` посылает на указанный узел пакеты и ждёт ответных пакетов (по протоколу **ICMP**). По команде

```
ping 192.168.45.5
```

мы можем получить, например, такой результат:

```
Обмен пакетами с 192.168.45.5 по 32 байт:
Ответ от 192.168.45.5: число байт=32 время=5мс
Ответ от 192.168.45.5: число байт=32 время<1мс
Превышен интервал ожидания для запроса.
Ответ от 192.168.45.5: число байт=32 время<1мс
```

Для каждого пакета указано время получения отклика. В данном случае связь есть, но третий пакет был потерян. Если пакеты не доходят, это означает, что связи с узлом нет или администратор запретил отвечать на запросы по протоколу ICMP.

Теперь проверим, как работает DNS-сервер. Определим IP-адрес сервера www.altlinux.org с помощью команды `nslookup`:

```
nslookup www.altlinux.org
```

Ответ может быть таким:

```
Server: UnKnown
Address: 172.16.172.19
Name: www.altlinux.org
Address: 194.107.17.79
```

Это значит, что в настройках сетевого соединения установлен DNS-сервер 172.16.172.19, который не имеет доменного имени (англ. *UnKnown* — неизвестный). Как следует из ответа этого DNS-сервера, узел www.altlinux.org имеет IP-адрес 194.107.17.79.

---

[1] В некоторых версиях, например в AltLinux, её нужно вызывать как `/sbin/ifconfig`

Если DNS-сервер доступен, в команде `ping` можно указывать не только IP-адрес, но и доменное имя, например

```
ping www.google.ru
```

Утилита tracert (в Linux — traceroute) показывает, по какому маршруту идут пакеты к заданному сайту. Например, результат выполнения команды

```
tracert www.yandex.ru
```

может выглядеть примерно так:

```
Трассировка маршрута к www.yandex.ru [87.250.251.3]
с максимальным числом прыжков 30:
  1  <1 мс  <1 мс  <1 мс  192.168.45.5
  2    3 мс   2 мс   3 мс  193.85.124.15
  3  10 ms  12 ms   1 ms  aurora-spb-ix.yandex.net [194.85.177.90]
  4  16 ms  10 ms  12 ms  aluminium-vlan934.yandex.net [213.180.208.12]
  5  19 ms  23 ms  12 ms  silicon-vlan901.yandex.net [77.88.56.125]
  6  30 ms  32 ms  31 ms  l3link-iva1-ugr1.yandex.net [213.180.213.4]
  7  18 ms  21 ms  24 ms  www.yandex.ru [87.250.251.3]
Трассировка завершена.
```

Эти данные говорят о том, что пакет достигает узла www.yandex.ru за 7 «прыжков» («хопов»), т. е. проходит 6 промежуточных узлов-маршрутизаторов. Каждому узлу посылается 3 пакета, в ответе указано время прохождения каждого из них. Если узел имеет доменное имя, оно записывается слева от IP-адреса. С помощью утилиты `tracert (traceroute)` можно определить, где именно нарушена связь.

Вопросы и задания

1. Сколько места в памяти занимает IP-адрес?
2. Что такое маска для IP-адреса? Как она строится?
3. Как вы думаете, могут ли два компьютера иметь одинаковый IP-адрес? Ответ обоснуйте.
4. Какие IP-адреса используются для локальных сетей?
5. Какие IP-адреса используют для обращения к своему компьютеру?
6. Почему становится необходимым переход на протокол IPv6?
7. Может ли компьютер иметь несколько IP-адресов? В каких случаях?
8. Зачем нужны доменные адреса?
9. Что такое домен?
10. В виде какой структуры можно представить доменную систему имён?

11. Что такое корневой домен?
12. Что такое доменные зоны? Какие они бывают?
13. Какие домены вы можете зарегистрировать (если они свободны)?
14. Что такое DNS-сервер? Какие функции он выполняет?
15. Что такое URL? Из каких частей он обычно состоит?
16. Приведите примеры URL для веб-страниц, рисунков, файлов на FTP-серверах.
17. Определите IP-адрес своего компьютера и маску подсети. Сколько компьютеров может быть в такой сети?
18. Что такое шлюз? Какие пакеты направляются на шлюз?

Подготовьте сообщение

а) «Протокол IPv6»
б) «Доменные зоны»
в) «Как зарегистрировать домен?»
г) «Домены с русскими буквами: за и против»
д) «Как настроить сеть в Windows (Linux, Mac OS)?»

Задачи

1. Лист бумаги, на котором был записан IP-адрес компьютера, оказался разорван на 4 части. Восстановите адрес компьютера (если решений несколько, выпишите все варианты):

а)	3.212	21	2.12	.42	д)	87.2	94.1	102.	49
б)	2.19	.50	5.162	22	е)	7.2	53	102.	84.1
в)	1.13	.29	1.109	19	ж)	.177	9.56	.20	120
г)	24.12	1.96	4.2	17	з)	2.222	.32	22	2.22

2. Какие из приведённых последовательностей могут быть масками?

а) 255.255.255.128
б) 255.255.128.64
в) 255.255.128.128
г) 255.255.128.0
д) 255.255.255.192
е) 255.255.224.0
ж) 255.255.224.192
з) 255.255.248.0

3. Почему в сети с маской /24 может быть только 254 узла, а не 256?

4. По заданным IP-адресу сети и маске определите адрес сети:

IP-адрес	Маска
а) 12.16.196.10	255.255.224.0
б) 145.92.137.88	255.255.240.0
в) 217.16.246.2	255.255.252.0
г) 146.212.200.55	255.255.240.0
д) 148.8.238.3	255.255.248.0

5. По заданным IP-адресу сети и маске определите номер компьютера в сети:

IP-адрес	Маска
а) 162.198.0.157	255.255.255.224
б) 156.128.0.227	255.255.255.248
в) 192.168.156.235	255.255.255.240
г) 10.18.134.220	255.255.255.192
д) 122.191.12.220	255.255.255.128
е) 156.132.15.138	255.255.252.0
ж) 112.154.133.208	255.255.248.0

6. Для каждого приведённого адреса определите номер сети, номер узла, наибольшее возможное количество компьютеров в сети:

а) 192.168.104.109/30	д) 92.60.65.180/26
б) 172.16.12.12/29	е) 118.212.123.1/24
в) 193.25.5.136/28	ж) 85.16.172.127/23
г) 10.10.40.15/27	з) 134.5.169.172/22

7. Определите маску сети минимального размера, в которую входит N компьютеров для значений $N = 16, 25, 32, 112$.
8. Проверьте, есть ли у вашего компьютера связь с узлом www.ya.ru. Определите среднее время отклика.
9. С помощью утилиты nslookup определите IP-адрес сервера www.google.ru. Что особенное вы обнаружили?
10. Определите маршрут, по которому идут пакеты с вашего компьютера на сайт kremlin.ru. Сколько «прыжков» составляет этот маршрут?

§ 49
Всемирная паутина

Что такое Всемирная паутина?

Всемирная паутина, или «веб» (англ. **WWW** — *World Wide Web*) — это служба для доступа к гипертекстовым документам (веб-страницам), хранящимся на серверах. Сейчас WWW — наиболее популярная служба Интернета.

Гипертекст — это текст, в котором есть активные ссылки (**гиперссылки**) на другие документы.

Гиперссылки в электронных документах обычно подчёркиваются и выделяются цветом (по умолчанию синим). Если щёлкнуть левой кнопкой мыши на гиперссылке, в окно браузера загружается документ, на который указывает ссылка.

На современных веб-страницах встречается не только текст, но и графика, звук, видео, причём каждый элемент может быть гиперссылкой. Такие документ называются **гипермедиа**.

Сайт (веб-сайт) — это группа веб-страниц, которые расположены на одном сервере, объединены общей идеей и связаны с помощью гиперссылок.

Чтобы сайт стал доступен другим компьютерам, на сервере должна быть запущена специальная программа — **веб-сервер**. Наиболее популярные веб-серверы:

- *Apache* (httpd.apache.org) — свободный веб-сервер для различных операционных систем, включая Windows, Linux, Mac OS;
- *IIS* (www.iis.net) — коммерческий веб-сервер для Windows;
- *nginx* (sysoev.ru/nginx) — бесплатный веб-сервер и почтовый сервер для крупных сайтов (есть версии для Windows и UNIX-подобных систем).

Для просмотра веб-страниц на экране используются программы-**браузеры** (*Internet Explorer*, *Mozilla Firefox*, *Chrome*, *Opera*, *Safari*). Браузер отправляет веб-серверу запрос, содержащий URL-адрес документа (веб-страницы, рисунка, файла и т. п.), а сервер в ответ передаёт запрошенные данные. Обмен обычно происходит по протоколу HTTP, однако для безопасного обмена секретной информацией, например для выполнения финансовых операций через Интернет, применяют протокол **HTTPS** (англ. *HyperText Transfer Protocol Secure*), предусматривающий шифрование всех передаваемых данных.

Особенность современного Интернета — привлечение пользователей к наполнению сайтов информацией и её корректировке,

к сотрудничеству, совместной деятельности в сети. Это привело к появлению термина **Web 2.0**, которым иногда обозначают современный этап развития Всемирной паутины.

Сайты, использующие технологии Web 2.0, как правило, требуют регистрации пользователей, для этого необходим действующий адрес электронной почты. Любой желающий может создать «личную зону» с собственными настройками и хранить там файлы, фотографии, видео, заметки. Другие могут комментировать эти материалы.

Пользователи объединяются в группы (сообщества) для того, чтобы вместе обсуждать интересующие их вопросы. Часто участники могут оценивать сообщения друг друга, таким образом, изменяется «репутация» (или «карма») участников, появляется некоторое соперничество.

Социальные сети: **ВКонтакте** (www.vkontakte.ru), **Одноклассники** (www.odnolkassniki.ru), **Facebook** (www.facebook.com) для многих стали местом общения с друзьями и одноклассниками.

Появились специальные сайты, где пользователи могут вести **блоги** — сетевые дневники (www.livejournal.com, www.blogspot.com). Влияние блогов настолько возросло, что их стали приравнивать к средствам массовой информации.

Активно развиваются **вики-системы** (англ. *wiki*) — веб-сайты, структуру и содержимое которых пользователи могут изменять с помощью инструментов, которые есть на самом сайте. Самый известный вики-сайт — это свободная энциклопедия **Википедия** (русская версия размещена на сайте ru.wikipedia.org).

С одной стороны, Web 2.0 расширяет возможности пользователей. С другой стороны, нужно понимать, что размещённые данные хранятся где-то на серверах, куда в принципе может получить доступ злоумышленник. Известны случаи массовых взломов учётных записей в социальных сетях и блогах. Поэтому не следует размещать в Интернете информацию, опубликование которой как-то может вам повредить, даже теоретически.

Фактически на описанных выше сайтах пользователи сами заполняют базу данных о себе, своих друзьях, карьере и даже личной жизни. Изучая и анализируя эти данные, владельцы сайтов и спецслужб получают возможность манипулировать людьми, используя полученную информацию в своих целях, например для рекламы товаров. Очень часто социальные сети используются для распространения вредоносных программ и рекламных сообщений **(спама)**.

В начале XXI века Тим Бернес-Ли (автор Всемирной паутины) предложил развивать веб в направлении создания «семантической паутины» (Web 3.0), в которой все документы связаны по ключевым словам, как в базе данных. Это потребует переделки всех сайтов (добавления специальных смысловых «ярлыков» — тэгов), что обеспечит возможность полностью автоматического поиска и обработки информации. Вместо ручного поиска человек будет использовать программу-агент, которая подберёт возможные ответы на вопрос и даст ему право окончательного выбора. Вместе с тем поиск нового типа позволит автоматически собирать всю информацию о личности (или организации), так что область его «частного пространства», «личной тайны» значительно уменьшится.

Поиск информации в Интернете

В Интернете сейчас содержится огромное количество данных, при этом найти нужную информацию иногда оказывается достаточно сложно.



Поисковая система — это веб-сайт, предназначенный для поиска информации в Интернете.

В начале развития Интернета, когда сайтов было немного, веб-мастера (создатели сайтов) составляли списки ссылок на интересные сайты. Когда ссылок стало много, их начали объединять в группы по темам. В результате развития этой идеи появились каталоги.



Каталог ссылок (англ. *web directory*) — это разбитый по темам список ссылок на сайты с их кратким описанием.

В каталогах обычно используют многоуровневую группировку ссылок (дерево): в каждой из крупных тем (Новости, Наука, Образование и др.) есть разделы, в разделах — подразделы и т. д.

Первым крупным сайтом-каталогом стал *Yahoo* (www.yahoo.com), созданный в 1995 г. За рубежом очень популярен также *Открытый каталог* (www.dmoz.org), который поддерживается международным сообществом редакторов. Самые крупные из рос-

сийских каталогов — *Яндекс-каталог* (yaca.yandex.ru) и *Каталог@Mail.ru* (list.mail.ru).

Каталоги заполняются вручную людьми-экспертами (редакторами каталога), каждый из которых отвечает за определённый раздел. Кроме того, веб-мастера могут предложить редакторам свои сайты для включения в каталог (бесплатно или платно).

Ссылки в каталогах, как правило, точно соответствуют разделу, в котором они размещены. Однако редакторы физически не могут посетить и проверить все новые сайты, которые ежедневно появляются в Интернете, поэтому часто случается, что нужный вам сайт не включен в каталог. Поэтому возникла естественная идея — заставить компьютерную программу искать новые сайты и автоматически анализировать информацию на их страницах. Так появились поисковые машины.

Поисковая машина — это автоматическая система, которая хранит информацию обо всех известных ей веб-страницах и выдает по запросу адреса тех из них, где встречаются введённые пользователем ключевые слова.

Робот-браузер поисковой машины (его часто называют «паук», англ. *crawler*) выкачивает с сайтов веб-страницы, переходя по всем встречающимся на них ссылкам[1].

Затем другая программа (**индексный робот**) удаляет из текста страницы всю служебную информацию (например, команды оформления) и строит **индекс**, похожий на книжный (рис. 7.18) — алфавитный список слов, для каждого из которых хранится адрес веб-страницы и номер (или номера) этого слова на странице.

Пользователь вводит в запросе ключевые слова, которые его интересуют.

А
аксиома 45
алгоритм 30, 78
архиватор 125

Б
бит 5, 15, 25, 43
брандмауэр 112
браузер 322

Рис. 7.18

Ключевые слова — это набор слов и выражений, которые отражают требуемую информацию.

[1] Начальный список страниц обычно задают разработчики.

Поисковый робот с помощью индекса находит те страницы, где встречаются эти слова.

Каждая поисковая машина имеет свой язык, который позволяет составлять сложные запросы, например исключать некоторые ключевые слова из поиска или искать одно слово из заданного набора слов. Во многих системах для обозначения логической операции «ИЛИ» (нужно одно из указанных слов) используется символ |, а для логической операции «И» (нужны оба слова) — символ &. Если нужно найти словосочетание, в запросе его берут в кавычки.

Обычно поисковый робот находит тысячи страниц, соответствующих запросу. Они выдаются пользователю в том порядке, который определяется разработчиками. Чаще всего учитывается *цитируемость* — число ссылок с других сайтов на эту страницу; чем ссылок больше, чем выше «ранг» данной страницы и тем выше она расположена в результатах поиска.

Самая крупная международная поисковая машина — *Google* (www.google.com). В России лидирующие позиции занимает *Яндекс* (www.yandex.ru). Эти системы умеют искать не только текст, но также картинки и видео (правда, при поиске изображений используется текстовая информация рядом с ними). Поисковая система *TinEye* (tineye.com) позволяет находить изображения, похожие на образец.

Вопросы и задания

1. Что такое гипертекст? Что такое гиперссылка?
2. Чем отличаются понятия «гипертекст» и «гипермедиа»?
3. Что такое веб-сервер? Что такое браузер?
4. Как работает технология «клиент — сервер» при просмотре веб-страниц?
5. В каких случаях используется протокол HTTPS? Чем он отличается от протокола HTTP?
6. Что такое Web 2.0? Расскажите о достоинствах и недостатках этой технологии. Приведите примеры.
7. Что такое сообщество?
8. Что такое блог? Чем, на ваш взгляд, объясняется популярность блогов?
9. Что такое вики-сайт?
10. Что такое семантическая паутина? Каковы, на ваш взгляд, достоинства и недостатки этой идеи?
11. Что такое поисковая система? Какие типы поисковых систем вы знаете?

12. Назовите известные вам каталоги ссылок.
13. В чем достоинства и недостатки каталогов?
14. Что такое поисковая машина? Как она работает?
15. Что такое ключевые слова?
16. Какую информацию кроме текста умеют искать поисковые машины?

Подготовьте сообщение

а) «Web 2.0 и Web 3.0»
б) «Семантическая паутина»
в) «Обмен секретной информацией в Интернете»
г) «Социальные сети: за и против»
д) «Блоггер — хобби или профессия?»
е) «Сервисы Google»
ж) «Язык запросов поисковой системы Google»
з) «Язык запросов поисковой системы Яндекс»
и) «Вики-сайты»

Задачи

1. Расположите запросы к поисковому серверу в порядке возрастания количества найденных страниц:
   а) принтеры & сканеры & продажа
   б) принтеры | продажа
   в) принтеры | сканеры | продажа
   г) принтеры & продажа
2. Расположите запросы к поисковому серверу в порядке возрастания количества найденных страниц:
   а) Америка | путешественники | Колумб
   б) Америка | путешественники | Колумб | открытие
   в) Америка | Колумб
   г) Америка & путешественники & Колумб
3. Расположите запросы к поисковому серверу в порядке возрастания количества найденных страниц:
   а) семена | помидоры | огурцы
   б) семена | (огурцы & семена)
   в) семена & помидоры
   г) семена | огурцы
4. Расположите запросы к поисковому серверу в порядке возрастания количества найденных страниц:
   а) ананасы | (груши & лимоны)
   б) ананасы | груши
   в) (груши & лимоны) | (ананасы & мандарины)
   г) ананасы | лимоны | груши

§ 50
Электронная почта

Электронная почта — один из первых **сервисов (служб)** Интернета, но и сейчас она играет в сети огромную роль. Для того чтобы отправлять и принимать сообщения, пользователь должен зарегистрировать **почтовый ящик** на одном из почтовых серверов в Интернете. Многие из почтовых ящиков бесплатны и имеют **веб-интерфейс**, т. е. позволяют работать с почтой в браузере, не устанавливая на компьютер почтовые программы. Примеры таких серверов — *mail.ru*, *mail.yandex.ru*, *mail.google.com*, *mail.yahoo.com*.

Электронный адрес состоит из двух частей — названия почтового ящика и имени сервера; они разделяются символом @, сленговое название которого в России — «собака» (его официальное название — **коммерческое at**). Адрес vasya@mail.ru означает: почтовый ящик vasya на сервере mail.ru.

Для отправки сообщения компьютер пользователя (Васи) должен обменяться данными с почтовым сервером по **протоколу SMTP** (англ. *Simple Mail Transfer Protocol* — простой протокол передачи почты). Затем электронное письмо передаётся на сервер, где зарегистрирован почтовый ящик адресата (на рис. 7.19 это

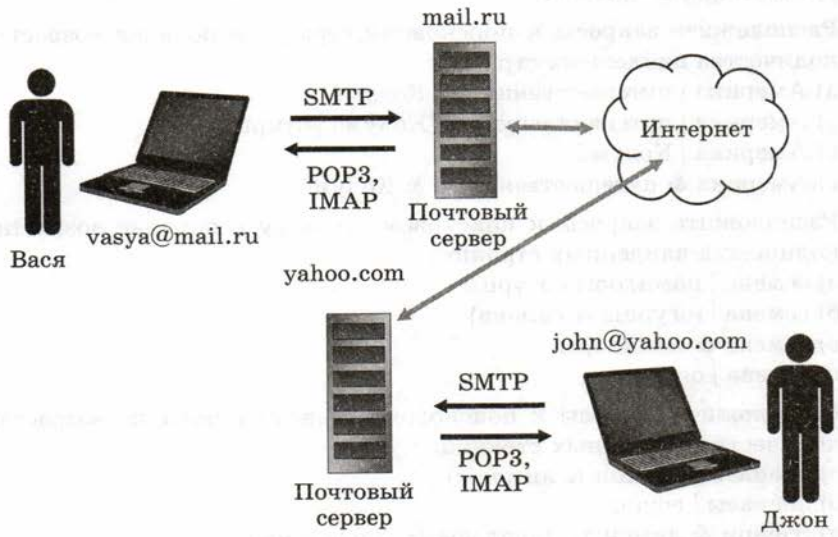


Рис. 7.19

сервер yahoo.com). Письмо сохраняется на сервере до тех пор, пока адресат (Джон) со своего компьютера не примет пришедшую ему почту, используя **протокол POP3** (англ. *Post Office Protocol* — почтовый протокол) или **протокол IMAP** (англ. *Internet Message Access Protocol* — протокол доступа к сообщениям в Интернете).

Чтобы получить свои сообщения с сервера, Джону необходимо ввести пароль, однако для отправки сообщения (по протоколу SMTP) это не нужно. Поэтому можно послать сообщение с любого адреса (без ведома его владельца), и очень сложно разобраться, кто же в самом деле автор письма. Эта ситуация привела к появлению массовых рассылок рекламы с чужих адресов — **спаму**. Для борьбы со спамом многие серверы требуют ввести пароль (выполнить авторизацию) даже при отправке сообщений.

Сообщение электронной почты состоит из заголовка, текста письма и вложенных файлов.

Заголовок письма содержит служебную информацию, необходимую для пересылки. Вот пример информации в заголовке (приведены русские и английские обозначения):

```
Кому (To):                  john@yahoo.com
От кого (From):             vasya@mail.ru
Ответить (Reply To):        vasya-home@mail.ru
Копия (CC):                 boss@mail.ru
Скрытая копия (BCC):        john2@yahoo.com
Тема (Subject):             О покупке слона
```

Поле «Ответить» используется тогда, когда сообщение посылается с одного адреса (например, с рабочего), а отвечать нужно на другой (домашний). По всем адресам, указанным в поле «Копия», отправляются копии письма. Копии отправляются ещё и по всем адресам, которые указаны в поле «Скрытая копия», но остальные получатели об этом не узнают.

Считается плохим тоном не заполнять поле «Тема» осмысленным текстом. Во-первых, часто сообщения без темы удаляются сразу как спам. Во-вторых, многие люди получают десятки сообщений в день, и им удобно сразу сортировать их по темам, а потом уже читать. В-третьих, искать нужное сообщение среди десятков других тоже удобнее всего по полю «Тема».

Основную часть письма тоже принято строить по некоторым правилам, похожим на правила составления бумажных писем.

Сначала идёт приветствие, затем суть сообщения, в конце фамилия и имя автора, а если это официальное письмо — его должность и сведения об организации. Например:

```
Здравствуйте, Джон!
Нет ли у Вас желания купить слона?
С уважением, Василий,
генеральный директор ООО «Рога и копыта»,
г. Роговск, ул. Копытная, 2,
тел. +7 (1812) 111-22-33, факс +7 (1812) 111-22-34
http://rogakopyta.ru
```

Вместе с письмом можно отправить любые небольшие файлы (ограничения на максимально допустимый размер файла устанавливается администратором сервера). Многие почтовые серверы запрещают пересылку исполняемых файлов (с расширением exe), потому что так могут распространяться вирусы и вредоносные программы.

Вопросы и задания

1. Что необходимо для отправки сообщений по электронной почте?
2. Из каких частей строится адрес электронной почты?
3. Что такое почтовый сервер?
4. Какие протоколы используются для работы с электронной почтой?
5. Что такое веб-интерфейс? Какие преимущества и недостатки он имеет в сравнении с использованием почтовых программ? Приведите примеры.
6. Объясните, как передаётся электронное сообщение от отправителя к получателю.
7. Почему на многих почтовых серверах нужно вводить пароль для отправки сообщений?
8. Что такое спам? Как вы думаете, почему спам — это плохо?
9. Какая информация включается в заголовок электронного письма?
10. Почему рекомендуется не оставлять поле «Тема» письма пустым?
11. Как принято строить основную часть электронного письма?
12. Какие файлы можно отправлять вместе с электронным сообщением?
13. Почему многие почтовые серверы не разрешают пересылку исполняемых файлов?

Подготовьте сообщение

а) «Протоколы POP3 и IMAP»
б) «Безопасность электронной почты»
в) «Почта Google»

§ 51
Другие службы Интернета

Обмен файлами (FTP)

Для обмена файлами используется **протокол FTP**, позволяющий скачивать файлы с сервера на компьютер пользователя (англ. *download*) и загружать файлы на сервер (англ. *upload*). Это возможно только тогда, когда на компьютере-сервере работает специальная программа — **FTP-сервер**, которая принимает запросы клиентов и отвечает на них по протоколу FTP.

FTP-серверы используются для распространения бесплатного программного обеспечения (свободные, бесплатные и условно-бесплатные программы, обновления антивирусных баз и т. п.) и загрузки файлов на веб-сайт.

Для работы с FTP-сервером необходимо зарегистрироваться под своим кодовым именем — так называемым **логином** (от англ. *login — log in*) и ввести **пароль** (англ. *password*). Однако многие FTP-серверы поддерживают анонимный вход: вместо имени нужно ввести «anonymous» (анонимный), а вместо пароля — любую последовательность символов.

Для работы с FTP-серверами используют программы, которые называются **FTP-клиентами**. Одна из самых популярных программ этого типа — свободный кроссплатформенный клиент FileZilla (filezilla-project.org). На рисунке 7.20 в одном окне программы показан каталог на компьютере пользователя, а в другом — каталог на FTP-сервере. Для работы с файлами можно использовать мышь.

Встроенные FTP-клиенты есть во многих других программах, например в *Far Manager*.

Браузеры тоже умеют работать по протоколу FTP. Зайдя на FTP-сервер, вместо красочных веб-страниц вы увидите просто список файлов и каталогов (рис. 7.21). Файл начинает скачиваться, если щёлкнуть на его имени, которое является ссылкой.

По умолчанию браузеры используют анонимный вход. Если нужно указать имя пользователя (логин) и пароль, их включают в адрес, который набирается в адресной строке:

ftp://user:asd@files.example.com

Здесь вход на FTP-сервер files.example.com выполняется под именем user с паролем asd.

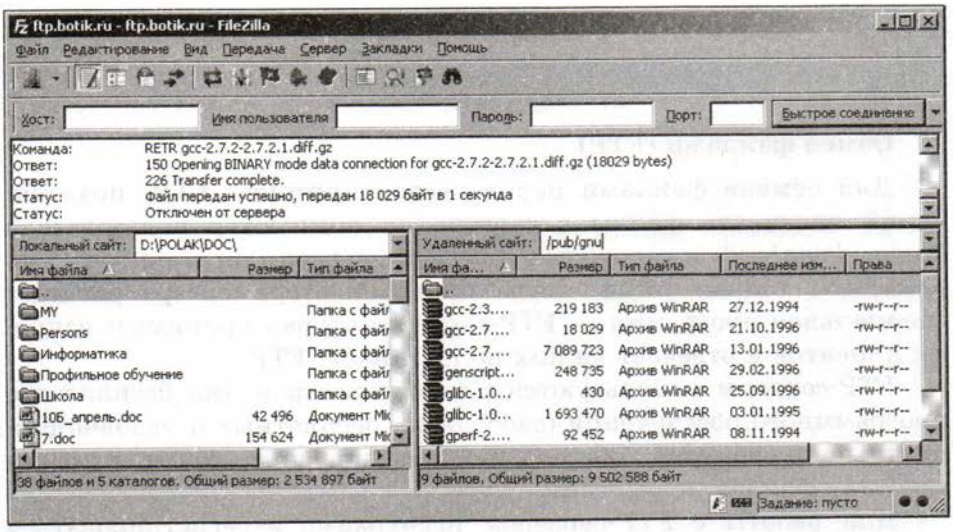


Рис. 7.20

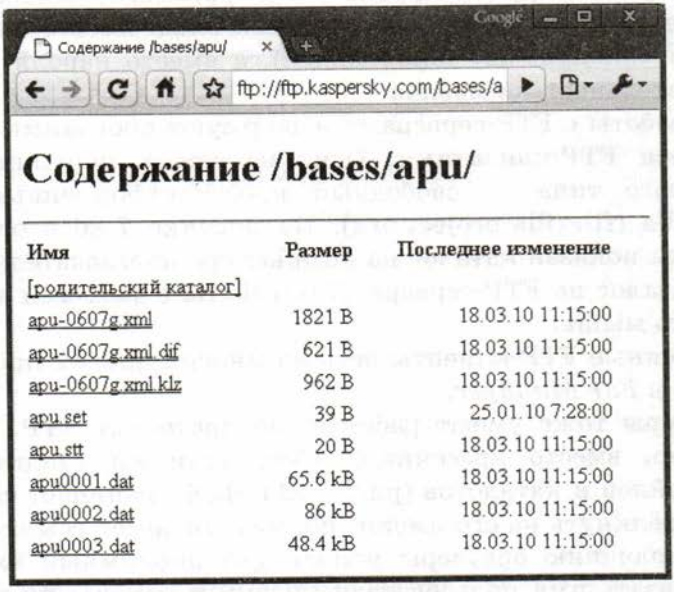


Рис. 7.21

Чтобы понять, какая информация содержится в файлах, можно поискать файлы с именами index, dirinfo, readme — обычно они содержат описание файлов текущего каталога.

Если вы точно знаете имя файла, с помощью специальных **поисковых систем** (например, *www.filesearch.ru*), можно найти FTP-сервер, где он находится.

Форумы

Форумы (рис. 7.22) — это специальные веб-сайты (или разделы веб-сайтов), предназначенные для общения посетителей в форме обмена сообщениями. Сообщения хранятся на серверах в Интернете и поэтому доступны всем участникам в любой момент.

Администратор форума создаёт несколько **разделов** форума, отличающихся по тематике. Пользователи создают в этих разделах **темы** для обсуждения (иногда тему называют «топик», от англ. *topic*, или «тред», от англ. *thread* — нить). Участники могут отвечать на любые сообщения в теме, комментировать их. Для изучения общественного мнения автор первого сообщения темы («топик-стартер», от англ. *topic starter* — тот, кто начал тему) может добавить к ней опрос **(голосование)**.

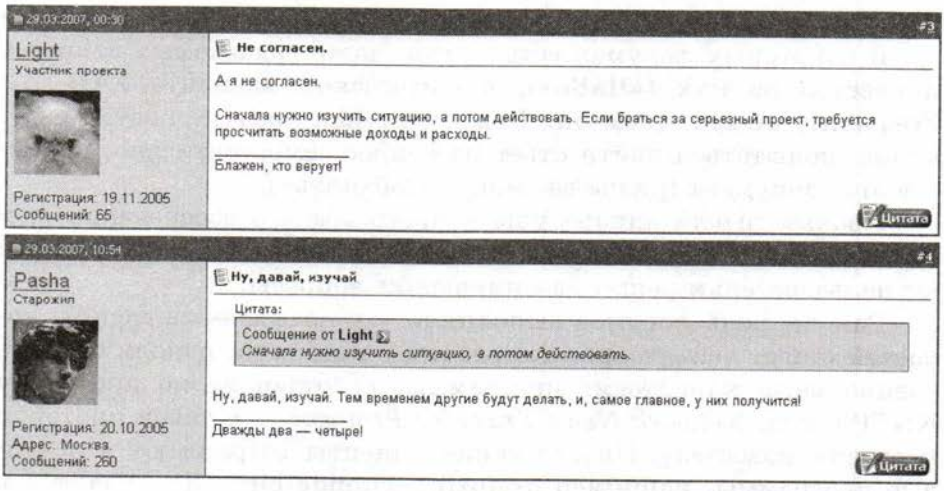


Рис. 7.22

Большинство форумов могут просматривать все желающие. Для того чтобы отправить своё сообщение, обычно требуется регистрация. Могут быть и «закрытые» разделы, для доступа в которые кроме регистрации нужно специальное разрешение администратора.

При регистрации пользователь выбирает **ник** (от англ. *nickname* — псевдоним); на сервере создаётся **профиль** участника форума — страница, где он может оставить информацию о себе, загрузить **аватар** — картинку, которая его может характеризовать, это может быть фотография. Иногда используют также слова «аватарка» или «юзерпик» (от англ. *user picture* — картинка пользователя), имеющие то же значение, что и аватар.

На большинстве форумов работает **система личных сообщений** — внутренняя электронная почта форума.

За порядком следят ответственные участники — администратор форума и назначенные им **модераторы**. Они могут изменять, перемещать и удалять любые сообщения, удалять профили пользователей и ограничивать им доступ — **«банить»**, т. е. запрещать отправлять сообщения (от англ. *ban* — запрет). Обычно в форумах наказывается:

- отклонение от темы — **оффтопик** (от англ. *off-topic* — «вне темы»);
- оскорбление участников, нецензурные выражения;
- реклама.

В некоторых форумах есть список часто задаваемых вопросов и ответов на них (**«ЧаВо»**), в английском варианте **FAQ** — Frequently Asked Question. Перед тем как создать новую тему, нужно попытаться найти ответ на вопрос самостоятельно, прочитав этот документ (иначе вас могут «забанить»).

Производители аппаратуры и программного обеспечения часто создают на своих сайтах форумы, где их представители помогают пользователям решать возникающие вопросы.

Раньше роль форумов выполняли так называемые **группы новостей** (англ. *newsgroup*), для работы с которыми использовались специальные клиентские программы, работающие по **протоколу NNTP** (англ. *Network News Transfer Protocol* — сетевой протокол передачи новостей). Иногда такие клиенты встраивают в почтовые программы, например в Mozilla Thunderbird. Для участия в современных группах новостей достаточно использовать любой браузер (см., например, groups.google.com).

Общение в реальном времени



Выражение **«в реальном времени»**, или **онлайн** (англ. *on-line* — на линии) означает, что все участники в момент обмена информацией находятся за компьютерами.

Простейший вариант общения в реальном времени — обмен текстовыми сообщениями. **Чаты** (англ. *chat* — болтовня) позволяют «разговаривать» группе людей, которые как бы находятся в одной комнате. Для личного общения используют **программы для обмена мгновенными сообщениями (мессенджеры)**, например: *ICQ* (www.icq.com), *Mail.ru Агент* (www.mail.ru), *Kopete* (для Linux), *iChat* (для компьютеров фирмы Apple). Возможно, в недалеком будущем их заменит бесплатная система мгновенного обмена сообщениями *Jabber* (www.jabber.org). С помощью мессенджеров можно передавать файлы напрямую с компьютера на компьютер или через сервер.

Особой популярностью пользуется бесплатная программа *Skype* (skype.com), которая позволяет:

- организовывать личные и групповые чаты;
- передавать сообщения и файлы с компьютера на компьютер;
- устанавливать голосовую и видеосвязь (для этого необходимы наушники, микрофон и веб-камера);
- звонить на стационарные и мобильные телефоны;
- принимать звонки с телефонов на специальный номер;
- отправлять SMS-сообщения;
- организовывать конференции (совещания через Интернет).

Часть из этих функций (например, звонки на телефоны и отправка SMS) платные.

Skype использует **технологию VoIP** (англ. *Voice over Internet Protocol* — передача голоса через интернет-протокол), все передаваемые данные шифруются. Звонок через Skype в другой город или в другую страну обходится значительно дешевле, чем обычная телефонная связь.

Skype — это кроссплатформенная программа, существуют её версии для операционных систем Windows, Linux, Mac OS, а также для смартфонов. Центральный сервер используется только для установки связи, а дальше компьютеры обмениваются данными напрямую.

Протокол VoIP, по которому работает Skype, закрыт, так же как и исходный код программы. Все данные многократно шифруются, и их не могут анализировать антивирусы. Поэтому специалисты по компьютерной безопасности предупреждают о возможности использования Skype для распространения вредоносных программ и шпионажа.

Информационные системы

Информационные системы в Интернете позволяют быстро находить нужную в данный момент информацию. Информационная система состоит из базы данных и программного обеспечения для поиска информации, размещённого на сайте.

На многих сайтах доступны **прогнозы погоды** на разные сроки (pogoda.ru, gismeteo.ru, pogoda.yandex.ru). С помощью сервиса rasp.yandex.ru можно узнать расписание электричек, поездов дальнего следования и самолётов по всей России. На сайтах аэропортов постоянно обновляется табло фактического прибытия и отправления самолётов с учетом задержки рейсов.

Заказывать билеты на поезда и самолёты удобно на специальных сайтах, которые связаны с системой продажи билетов железной дороги и авиакомпаний. Здесь же можно получить полную информацию о расписании, возможных вариантах проезда (перелёта), стоимости и наличии билетов. Оплатить билеты можно прямо на сайте с помощью банковской карты или платёжных систем, а также через терминалы приёма платежей.

Часто используются **электронные билеты** (англ. *e-ticket*) на поезда и самолёты. Электронный билет — это информация о заказе, сделанном через веб-сайт, которая внесена в базу данных. По номеру заказа в кассе вокзала можно получить бумажный билет, а в аэропортах для регистрации по электронному билету достаточно просто предъявить паспорт.

Сервисы **веб-картографии** Google Maps (maps.google.ru), Яндекс.Карты (maps.yandex.ru), Карты@Mail.ru (maps.mail.ru) позволяют найти на карте любой адрес, проложить маршрут и оценить его длину. На этих сайтах доступны не только карты (хранящиеся в векторном формате), но также снимки многих районов из космоса.

Вопросы и задания

1. Зачем используются FTP-серверы?
2. Как и в каком случае можно зайти на FTP-сервер, не имея своей учётной записи?
3. Что такое FTP-клиент?
4. Как работать с FTP-серверами с помощью браузера?
5. Как узнать, что находится в файлах, не скачивая их? Всегда ли это возможно?
6. Как найти сервер, откуда можно загрузить интересующий вас файл?
7. Что такое форумы?

8. Объясните значение слов «топик», «ник», «аватар», «модератор», «бан», «оффтопик».
9. Как поддерживается порядок в форуме?
10. Что такое ЧаВо (FAQ)?
11. Как вы думаете, зачем производители аппаратуры организуют форумы на своих сайтах? Приведите примеры.
12. Что такое общение в реальном времени? Относятся ли форумы к онлайн-общению?
13. Расскажите о достоинствах и недостатках программы Skype.
14. Какие информационные системы существуют в Интернете?
15. Что такое электронный билет? Чем он удобнее бумажного?

Подготовьте сообщение

а) «Служба FTP»
б) «Онлайн-общение в Интернете»
в) «Картографические сервисы Интернета»
г) «Поиск и заказ билетов в Интернете»
д) «Онлайн-переводчики»
е) «Онлайн-словари»

§ 52 Электронная коммерция

Что такое электронная коммерция?

В начале своего развития Интернет был полностью некоммерческим — в его развитии участвовали инженеры, программисты, учёные и военные. Однако к началу 1990-х гг. стало ясно, что глобальная сеть может быть источником огромной прибыли.

Электронная коммерция (англ. *e-commerce*) — это покупка и продажа товаров и услуг с помощью электронных систем, например через Интернет.

Развитие электронной коммерции в Интернете началось в 1994 г., когда на сайте американской сети ресторанов Pizza Hut появилась возможность заказать пиццу с доставкой на дом. В том же году открылись сайты некоторых банков в Интернете, и пользователи получили возможность управления своими счетами через сеть. В 1995 г. был создан первый книжный интернет-магазин Amazon (www.amazon.com), который и сейчас остаётся самым крупным в мире.

Электронная коммерция включает в себя:

- исследование рынка;
- обмен данными и документами в электронном виде;
- денежные операции в электронной форме;
- продажу товаров, услуг и информации;
- поддержку покупателей после продажи.

Сейчас многие покупатели начинают поиски нужного товара в Интернете, а не в обычных магазинах. Чтобы привлечь внимание клиентов, фирмы размещают подробную информацию о товарах на своих сайтах, делают рассылки по электронной почте, создают сообщества в социальных сетях, организуют дискуссии на форумах. Посетителям сайтов предлагается оставить отзыв о товарах на специальной веб-странице, чтобы остальные смогли его прочитать.

Через Интернет удобно приобретать книги и электронику, авиа- и железнодорожные билеты, оплачивать услуги операторов сотовой связи.

Для пользователя всегда важно знать, что после покупки он не окажется один на один со своими проблемами и сможет получить консультацию. Поэтому поддержка покупателей — тоже важный элемент электронной коммерции. На сайтах производителей оборудования (например, жёстких дисков, принтеров, сканеров и т. п.) всегда можно найти всю документацию по настройке и обслуживанию устройств и бесплатно скачать самые новые драйверы. На форумах фирм-изготовителей пользователи получают консультацию службы технической поддержки и делятся друг с другом решениями возникших проблем.

Бизнес в Интернете предоставляет дополнительные **возможности для компаний:**

- расширение сферы влияния: фирмы любого размера могут принимать заказы со всего мира;
- увеличение конкурентоспособности: быстрая реакция на претензии клиентов и изменение ситуации на рынке;
- индивидуальный подход (система собирает информацию о клиенте и пытается предложить именно те товары, которые он чаще покупает);
- уменьшение затрат: не нужно арендовать помещение для магазина и нанимать много сотрудников.

Покупатели тоже получают серьезные преимущества:

- выбор товаров и услуг из большого количества вариантов;
- легко сравнить разные предложения;
- можно узнать отзывы других пользователей;
- можно заказывать и оплачивать товары в удобное время;
- цены на товары и услуги обычно ниже, чем в обычных магазинах.

Интернет-магазины

Всё больше людей используют **интернет-магазины** — веб-сайты, которые рекламируют товары или услуги, принимают заказы на покупку, предлагают варианты оплаты и получения заказа. Выбрав товары в интернет-магазине, пользователь может «положить их в корзину», т. е. включить в список для приобретения. Когда состав покупки полностью определен, оформляется заказ: покупатель указывает свои данные, способ оплаты и доставки. На многих сайтах возможен заказ товара по телефону.

Оплата выполняется разными способами:

- через банковскую карту, пригодную для оплаты в Интернете (Visa, MasterCard);
- банковским переводом (например, через Сбербанк);
- почтовым переводом;
- с помощью электронных платёжных систем (электронными деньгами);
- наличными при получении товара;
- через отправку платного SMS-сообщения (для небольших покупок).

Для «вещественных» товаров существуют три способа **доставки**:

- курьерская доставка по указанному адресу;
- почтовая доставка;
- «самовывоз» с пунктов выдачи товара (в некоторых городах).

Товары в электронной форме (программы, коды доступа, тексты, статьи, фотографии и т. п.) обычно высылаются по электронной почте или покупателю предоставляется персональная ссылка на нужный файл на сервере фирмы-продавца.

Для управления интернет-магазином используют специальное программное обеспечение, например кроссплатформенные системы *1С-Битрикс* (www.1c-bitrix.ru) и *osCommerce* (www.oscommerce.com).

Ещё один вид электронной коммерции — **интернет-аукционы** («онлайновые аукционы»), в которых фирма-организатор — это просто посредник между продавцом и покупателем. Любой желающий может делать ставки, используя веб-сайт аукциона. Когда интернет-аукцион завершен, покупатель, сделавший наибольшую ставку, должен перевести деньги продавцу, а продавец обязан выслать товар покупателю по почте. Крупнейший интернет-аукцион — *eBay* (www.ebay.com).

Электронные платежные системы

Электронная торговля не была бы столь удобной, если бы не было возможности оплачивать покупку прямо в Интернете, не выходя из дома. Для этого должны были появиться **электронные деньги**, которые можно свободно купить и продать. Системы расчётов электронными деньгами называются **электронными платёжными системами**. Среди российских платёжных систем наиболее известны *WebMoney* (www.webmoney.ru) и *ЯндексДеньги* (money.yandex.ru).

В платёжной системе можно завести **электронный кошелёк**, который пополняется с помощью специальных карт оплаты или через терминалы. Из такого кошелька можно оплачивать коммунальные услуги, сотовую связь и Интернет, покупать авиа- и железнодорожные билеты, товары в интернет-магазинах. По условиям пользовательского соглашения кошелёк системы Яндекс. Деньги нельзя использовать для коммерческой деятельности (например, для получения оплаты за выполненную работу), иначе его может заблокировать служба безопасности.

Пользователи могут переводить электронные деньги не только на счета интернет-магазинов, но и на кошельки других пользователей. Чтобы убедиться в том, что вы не ошиблись при вводе номера кошелька и переводите деньги именно тому, кому нужно, применяют перевод с кодом протекции. **Код протекции** — это некоторое число, которое должен ввести получатель для получения перевода на свой счет. Этот код можно сообщить по электронной почте или по телефону. Если получатель вводит неверный код несколько раз подряд или истекает срок действия перевода, средства возвращаются в кошелёк отправителя.

При необходимости можно вывести средства из электронного кошелька, т. е. получить их в виде наличных денег в банке (за вычетом небольшой комиссии).

С электронными кошельками можно работать в браузере (через веб-интерфейс сайта) или с помощью специальной программы, которая устанавливается на компьютер пользователя.

Вопросы и задания

1. Что такое электронная коммерция?
2. Какие направления включает электронная коммерция?
3. Какие преимущества предоставляет электронная торговля для продавцов и покупателей?
4. Что такое интернет-аукцион?
5. Зачем используется код протекции при переводе электронных денег?
6. Что такое интернет-магазин? Объясните на примере.
7. Как можно оплатить покупку в интернет-магазине? Как доставляются покупки?
8. Что такое электронные деньги? Чем они, на ваш взгляд, отличаются от «обычных» денег?

Подготовьте сообщение

а) «Интернет-магазины: за и против»
б) «Интернет-аукционы»
в) «Электронные платёжные системы»

§ 53
Право и этика в Интернете

Интернет и закон

Как только в сети появилась купля-продажа товаров и услуг, возникла необходимость регулировать эти отношения с помощью закона, защищать интересы пользователей и предотвращать мошенничество.

Интернет — это не организация, он не принадлежит ни одной стране, развивается во многом стихийно и не может быть юридическим лицом. В связи с этим возникает множество проблем, с которыми юристы раньше не сталкивались. Например, не вполне ясно:

- несёт ли провайдер ответственность за действия пользователей, которым он предоставляет доступ в Интернет (в том числе за нарушения авторских прав);

- можно ли признавать доказательствами цифровые документы (сообщения электронной почты, рисунки, звукозаписи, видео);
- как доказать условия заключённой сделки, если фирма может в любой момент изменить условия договора на сайте;
- какую ответственность несут платёжные системы перед государством и пользователями.

Соблюдение **авторских прав**, т. е. прав автора на результаты своего интеллектуального труда, — один из самых актуальных правовых вопросов Интернета. Практически вся информация на сайтах защищается авторским правом: программное обеспечение, тексты, рисунки и фотографии, музыка и видеофильмы. Часто на веб-страницах публикуются условия использования материала (англ. *terms of use*), где указывается, можно ли сохранять и копировать материал, вставлять его в другие документы, распечатывать. Если такой информации нет, следует получить разрешение на использование материала, отправив сообщение веб-мастеру (автору сайта) по электронной почте.

На своём веб-сайте **можно без разрешения**:

- размещать гиперссылки на другие сайты;
- использовать бесплатную графику.

Без разрешения нельзя:

- копировать содержание других сайтов;
- объединять информацию из разных источников для создания «собственного» документа;
- изменять чужой текст или изображение;
- размещать любые изображения с других сайтов, о которых явно не написано, что они бесплатные.

В Гражданском кодексе Российской Федерации (ГК РФ) определяется, что можно без согласия автора использовать его произведения в научных, учебных или культурных целях (статья 1274). При этом обязательно указать имя автора и источник (книгу, статью, сайт). Например, разрешается:

- цитировать произведения (для того, чтобы подтвердить или опровергнуть какую-то мысль автора) в объёме, оправданном целью цитирования;
- использовать произведения и их отрывки в учебных материалах в объёме, оправданном поставленной целью;
- использовать произведения для создания пародии или карикатуры.

Использование чужого произведения (например, текста, музыки или видеозаписи) как составной части в своих работах является нарушением авторского права независимо от объёмов (например, нельзя включить в свой фильм даже 1 секунду из другого фильма или звукозаписи). При этом не имеет значения, что вы не получили от этого коммерческой выгоды (это влияет только на размер штрафа).

Переработка чужого материала (например, наложение текста, графики, звука, монтаж видео) — это серьёзное нарушение права автора на неприкосновенность произведения (статьи 1255 и 1266 ГК РФ), за это в законе предусмотрены значительные штрафы.

Незаконный доступ к информации — это уголовное преступление, за которое в Уголовном кодексе Российской Федерации предусмотрен штраф или лишение свободы на срок до двух лет (статья 272). Если ваш сайт (или учётную запись на сайте) взломали или вы стали жертвой интернет-мошенничества, нужно обращаться в **отдел по борьбе с компьютерными преступлениями** (отдел «К») полиции.

Нетикет

На ранних этапах развития, когда к Интернету были подключены только научные центры и университеты, общение основывалось на взаимном уважении и доверии пользователей. В результате сложились неофициальные правила общения, которые называются **сетевым этикетом** или **нетикетом** (фр. *netiquette*).

Несмотря на то что при общении в Интернете вы, как правило, не видите собеседника, нужно вести себя так, как будто вы говорите с человеком лично: не пишите то, что вы не смогли бы сказать ему в лицо; не оскорбляйте собеседника, не ругайтесь. Перед тем как послать сообщение, прочтите его внимательно ещё раз и поставьте себя на место получателя.

Передавая информацию по открытым каналам, помните, что копии всех ваших сообщений могут храниться, например, у провайдера. Не посылайте информацию, доступ к которой ограничен. Уважайте авторские права, не используйте чужой материал без разрешения. Уважайте тайну переписки: если вы хотите опубликовать личное сообщение, нужно спросить разрешения у автора.

В сообщениях **электронной почты** и **форумов**:

- пишите кратко и точно, цените время других людей;
- не пишите слова заглавными буквами (это воспринимается как крик или визг);

- не используйте сленг, пишите грамотно, не пропускайте пробелы и знаки препинания;
- для передачи тона письма используйте **смайлики** (англ. *smiley*) — значки для обозначения эмоций, например:
  :-) или :) — улыбка;
  :-(или :(— несчастное лицо, сожаление или разочарование;
  ;-) или ;) — подмигивающее лицо, слова не следует понимать слишком серьезно;
- цитируйте те высказывания, на которые отвечаете (собеседник может забыть содержание предыдущего письма);
- не распространяйте спам — нежелательную рекламу.

Посылая сообщения **по электронной почте**:

- обязательно пишите тему сообщения, отражающую содержание письма;
- ставьте подпись (имя и фамилию) в конце письма;
- не посылайте большие файлы без согласия получателя.

Участвуя **в форумах**:

- сначала прочитайте список часто задаваемых вопросов и прошлые сообщения, возможно, вы найдете там ответ на свой вопрос;
- отправляя сообщение, осознайте, что многие увидят ваш текст;
- не отклоняйтесь от темы обсуждения;
- не участвуйте во **флейме** (от англ. *flame* — огонь, пламя) — так называется «спор ради спора», словесная война; обычно за флейм наказывают модераторы форума;
- не разжигайте **холивары** (от англ. *holy war* — «священная война») — так называется спор о двух идеях, каждая из которых имеет своих сторонников (например, что лучше: Windows или Linux, Паскаль или Си и т. п.).

Общаясь **в чатах**:

- не перебивайте собеседника;
- не обижайтесь, если незнакомые люди не хотят с вами разговаривать;
- не пытайтесь выведывать личную информацию;
- уважайте анонимность, не разглашайте реальное имя другого участника без разрешения, если вы его знаете;
- будьте снисходительны к ошибкам других;
- не обижайтесь, если собеседник неожиданно покинул чат.

Вопросы и задания

1. Какие юридические проблемы возникают в связи с куплей-продажей услуг в Интернете?
2. В каком случае можно размещать на своем веб-сайте чужой материал?
3. В каком случае можно бесплатно использовать произведения без согласия автора? Какие ограничения нужно при этом соблюдать?
4. Расскажите о наиболее распространённых нарушениях авторских прав в Интернете.
5. Является ли взлом персональной странички в социальной сети преступлением? Почему?
6. Что можно сделать, если вас обманули мошенники в Интернете?
7. Что такое нетикет?
8. Можно ли опубликовать на форуме личное сообщение?
9. Какие правила рекомендуется соблюдать в сообщениях электронной почты?
10. Как нужно вести себя в форумах и чатах?
11. Как в электронном письме можно передать свое настроение?
12. Почему при пересылке больших файлов нужно спросить согласие получателя?
13. Объясните значение слов «флейм» и «холивар».

Подготовьте сообщение

а) «Интернет и право»
б) «Авторские права в Интернете»
в) «Сетевой этикет»

Практические работы к главе 7

Работа № 23 «Тестирование сети»
Работа № 24 «Сравнение поисковых систем»

ЭОР к главе 7 на сайте ФЦИОР (http//fcior.edu.ru)

- Глобальные компьютерные сети
- Архитектура Интернета
- Адресация в Интернете
- Технология трансляции сетевых адресов
- Технология WWW
- Протоколы передачи данных в сети Интернет

- Контроль знаний: история Всемирной паутины
- Службы Интернета
- Поиск информации
- Поиск информации в Интернете
- Законодательство РФ «Об информации, информационных технологиях и о защите информации»

Самое важное в главе 7

- Компьютерная сеть — это группа компьютеров, соединенных линиями связи. Сети предназначены для ускорения обмена данными и совместного использования ресурсов. Вместе с тем объединение компьютеров в сеть снижает безопасность данных.
- Сервер — это компьютер, предоставляющий свои ресурсы в общее использование. Клиент — это компьютер, использующий ресурсы сервера.
- Протокол — это набор правил и соглашений, определяющих способ и порядок обмена данными в сети. Компьютеры могут обмениваться данными только тогда, когда они поддерживают общий протокол.
- В современных сетях используется пакетная передача данных, при которых нагрузка на сеть становится более равномерной.
- Существуют три основные структуры (топологии) сетей: общая шина, звезда и кольцо. В современных локальных сетях, как правило, используется схема «звезда».
- Интернет — это глобальная компьютерная сеть, которая использует семейство протоколов TCP/IP.
- Любой компьютер, подключенный к сети Интернет, имеет собственный IP-адрес. Если у компьютера есть несколько интерфейсов (сетевых карт или адаптеров беспроводной связи), каждый из них может иметь свой IP-адрес.
- Система доменных имён (DNS) позволяет сопоставить IP-адресу символьное имя.
- Адрес ресурса (URL) содержит протокол, имя сервера, каталог и имя документа (файла).

Глава 8
Алгоритмизация и программирование

§ 54
Алгоритм и его свойства



Мухаммед ал-Хорезми (ок. 783–ок. 850 гг.)

Что такое алгоритм?

Происхождение слова «алгоритм» связывают с именем учёного Мухаммеда ал-Хорезми (перс. خوارزمی [al-Khwārazmī]), который описал десятичную систему счисления (придуманную в Индии) и предложил правила выполнения арифметических действий с десятичными числами.

Алгоритм — это точное описание порядка действий, которые должен выполнить исполнитель для решения задачи за конечное время.

Здесь **исполнитель** — это устройство или одушёвленное существо (человек), способное понять и выполнить команды, составляющие алгоритм.

Человек как исполнитель часто действует неформально, по-своему понимая команды. Несмотря на это, ему тоже часто приходится действовать по тому или иному алгоритму. Например, рецепт приготовления какого-либо блюда можно считать алгоритмом. На уроках русского языка, выполняя разбор слова или предложения, вы тоже действуете по определённому алгоритму. Много различных алгоритмов в математике (постарайтесь вспомнить известные вам). На производстве рабочий, вытачивая деталь в соответствии с чертежом, действует по алгоритму, который разработал технолог. И таких примеров может быть множество.

В информатике рассматривают только **формальных исполнителей**, которые не понимают (и не могут понять) смысл команд. К этому типу относятся все технические устройства, в том числе и компьютер.

Каждый формальный исполнитель обладает собственной **системой команд**. В алгоритмах для такого исполнителя нельзя использовать команды, которых нет в его системе команд.

Свойства алгоритма

- **Дискретность** — алгоритм состоит из отдельных команд (шагов), каждая из которых выполняется за конечное время.
- **Детерминированность (определённость)** — при каждом запуске алгоритма с одними и теми же исходными данными должен быть получен один и тот же результат.
- **Понятность** — алгоритм содержит только команды, входящие в систему команд исполнителя, для которого он предназначен.
- **Конечность (результативность)** — для корректного набора данных алгоритм должен завершаться через конечное время с вполне определённым результатом (результатом может быть и сообщение о том, что задача не имеет решений).
- **Корректность** — для допустимых исходных данных алгоритм должен приводить к правильному результату.

Эти свойства не равноправны. Дискретность, детерминированность и понятность — фундаментальные свойства алгоритма, т. е. ими обладают все алгоритмы для формальных исполнителей. Остальные свойства можно рассматривать как требования к «правильному» алгоритму.

Иными словами, алгоритм получает на вход некоторый дискретный входной объект (например, набор чисел или слово) и обрабатывает входной объект по шагам (дискретно), строя промежуточные дискретные объекты. Этот процесс может закончиться или не закончиться. Если процесс выполнения алгоритма заканчивается, то объект, полученный на последнем шаге работы, является результатом работы алгоритма при данном входе. Если процесс выполнения не заканчивается, говорят, что алгоритм зациклился. В этом случае результат его работы не определён.

Способы записи алгоритмов

Алгоритмы можно записывать разными способами:

- **на естественном языке**, обычно такой способ применяют, записывая основные идеи алгоритма на начальном этапе;
- на **псевдокоде**, так называется смешанная запись, в которой используется естественный язык и операторы какого-либо

языка программирования; в сравнении с предыдущим вариантом такая запись гораздо более строгая;
- в виде **блок-схемы** (графическая запись);
- в виде **программы** на каком-либо языке программирования.

Мы будем записывать алгоритмы в виде программы на двух языках программирования: на **языке Паскаль** (версия FreePascal) и на алгоритмическом языке **системы КуМир**, который называют **школьным алгоритмическим языком**, а также в некоторых случаях — на **псевдокоде**.

Вопросы и задания

1. Что такое алгоритм?
2. Что такое исполнитель?
3. Чем отличаются формальные и неформальные исполнители?
4. Что такое система команд исполнителя? Придумайте исполнителя с некоторой системой команд.
5. Перечислите и объясните свойства алгоритма.
6. Какие существуют способы записи алгоритмов? Какие из них, по вашему мнению, чаще применяются на практике? Почему?

Подготовьте сообщение

а) «История слова алгоритм»
б) «Свойства алгоритма»
в) «Способы записи алгоритмов»

§ 55
Простейшие программы

Пустая программа

Пустая программа — это программа, которая ничего не делает, но удовлетворяет требованиям выбранного языка программирования. Она полезна, прежде всего, для того, чтобы понять общую структуру программы. Далее мы будем слева приводить программу на школьном алгоритмическом языке, а справа — эквивалентную программу на языке Паскаль.

```
алг Куку
нач
 | основная программа
кон
```

```
program qq;
begin
 { основная программа }
end.
```

Программа на школьном алгоритмическом языке начинается **ключевым словом алг** (сокращение от слова «алгоритм»), за которым записывают название алгоритма. Название может содержать русские и латинские буквы, цифры, знак подчёркивания «_» и даже пробелы, но не может начинаться с цифры. Между ключевыми словами **нач** и **кон** размещают **операторы** (команды) программы (**тело программы**). После вертикальной черты записывают **комментарии** — пояснения, которые не обрабатываются транслятором.

В языке Паскаль в имени программы нельзя использовать русские буквы и пробелы. Все ключевые слова записываются на английском языке. После имени программы ставится точка с запятой. Ключевые слова **begin** и **end** ограничивают тело программы, после слова **end** ставится точка. Комментарии заключают в фигурные скобки.

Вывод текста на экран

Напишем программу, которая выводит на экран такие строки:

```
2+2=?
Ответ: 4
```

Вот как она выглядит:

```
алг qq
нач
  вывод '2+'
  вывод '2=?', нс
  вывод 'Ответ: 4'
кон
```

```
program qq;
begin
  write('2+');
  writeln('2=?');
  write('Ответ: 4')
end.
```

Команда `вывод` в школьном алгоритмическом языке **выводит** на экран символы, заключённые в апострофы или в кавычки. Для перехода на новую строку в **списке вывода** нужно указать `нс` (новая строка).

В языке Паскаль для вывода данных используют процедуру `write` (без перехода на новую строку) или `writeln` (после окончания вывода происходит переход на новую строку). Параметры этих процедур заключаются в круглые скобки, а символьные строки — в апострофы. Каждый оператор заканчивается точкой с запятой, перед словом **end** можно её не ставить.

Переменные

Напишем программу, которая выполняет сложение двух чисел:

1) запрашивает у пользователя два целых числа;
2) складывает их;
3) выводит результат сложения.

Программу на псевдокоде (смеси русского и школьного алгоритмического языков) можно записать так:

```
алг Сумма
нач
  ввести два числа
  сложить их
  вывести результат
кон
```

Компьютер не может выполнить псевдокод, потому что команд «ввести два числа» и ей подобных нет в его системе команд. Поэтому нам нужно «расшифровать» все такие команды через операторы языка программирования.

В отличие от предыдущих задач здесь требуется хранить данные в памяти. Для этого используют переменные.

Переменная — это величина, которая имеет имя, тип и значение. Значение переменной может изменяться во время выполнения программы.

Переменная определяет область памяти, где хранится только одно значение. При записи в неё нового значения «старое» стирается, и его уже никак не восстановить.

Переменные в программе необходимо объявлять[1]. При **объявлении** указывается **тип** переменной и её **имя (идентификатор)**. **Значение** переменной сразу после объявления неопределённое: переменной выделяется некоторая область памяти, и там могло быть до этого записано любое число.

Вот так объявляются целочисленные переменные (в которых могут храниться только целые значения) с именами *a*, *b* и *c*:

```
цел a, b, c
```

```
var a, b, c: integer;
```

[1] В некоторых языках программирования, например в языке Python, переменные не объявляются. Но это может привести к ошибкам, которые трудно обнаружить.

В языке Паскаль описание переменных начинается с ключевого слова **var**, после него записывают список переменных и в конце через двоеточие — их тип, в данном случае **целочисленный** (англ. **integer**).

Имена переменных строятся по тем же правилам, что и имена программ. В языке Паскаль можно использовать в именах латинские буквы (строчные и заглавные буквы не различаются), цифры (но имя не может начинаться с цифры, иначе транслятору будет сложно различить, где начинается имя, а где — число) и знак подчёркивания «_».

В школьном алгоритмическом языке в именах разрешены кроме перечисленных символов ещё пробелы и русские буквы, причём строчные и заглавные буквы различаются (поэтому `x` и `X` — это разные имена переменных).

Желательно давать переменным «говорящие» имена, чтобы можно было сразу понять, какую роль выполняет та или иная переменная.

Тип переменной нужен для того, чтобы:

- определить область допустимых значений переменной;
- определить допустимые операции с переменной;
- определить, какой объём памяти нужно выделить переменной и в каком формате будут храниться данные (вспомните, что целые и вещественные числа хранятся в компьютере по-разному, см. главу 4);
- предотвратить случайные ошибки; например, при попытке записать символ в целую переменную выдаётся сообщение об ошибке[1].

В алгоритмическом языке можно при объявлении задать начальные значения переменных:

```
цел a, b=1, c=55
```

Значение переменной `a` осталось неопределённым.

Большинство трансляторов Паскаля автоматически заполняют нулями все переменные основной программы, однако лучше не надеяться на это и явно задавать начальные значения всех переменных.

[1] Некоторые языки, например язык Си, позволяют записывать в переменную значение другого типа, но вся ответственность за результат ложится на программиста.

Приведём полную программу сложения двух чисел:

```
алг Сумма
нач
  цел a, b, c
  ввод a, b
  c:=a+b
  вывод c
кон
```

```
program Sum;
var a, b, c: integer;
begin
  read(a, b);
  c:=a+b;
  write(c)
end.
```

Оператор ввод (в Паскале — процедура read) предназначен для **ввода** данных (например, с клавиатуры).

Оператор, содержащий символы «:=», — это **оператор присваивания**, с его помощью изменяют значение переменной. Он выполняется следующим образом: вычисляется выражение справа от символов «:=», а затем результат записывается в переменную, записанную слева от символов «:=». Поэтому, например, оператор

```
i:=i+1
```

```
i:=i+1;
```

увеличивает значение переменной i на 1.

У приведённой выше программы сложения чисел есть два недостатка:

1) перед вводом данных пользователь не знает, что от него требуется (сколько чисел нужно вводить и каких);
2) результат выдаётся в виде числа, которое означает неизвестно что.

Хотелось бы, чтобы диалог программы с пользователем выглядел так:

```
Введите два целых числа: 2 3
2+3=5
```

Подсказку для ввода вы можете сделать самостоятельно. При выводе результата ситуация несколько усложняется, потому что нужно вывести значения трёх переменных и два символа: «+» и «=». Для этого строится **список вывода**, элементы в котором разделены запятыми:

```
вывод a, '+', b, '=', c
```

```
write(a, '+', b, '=', c);
```

Обратите внимание, что имена переменных записаны без апострофов.

В принципе можно было бы обойтись и без переменной c, потому что элементом списка вывода может быть арифметическое выражение, которое сразу вычисляется и на экран выводится результат:

```
вывод a, '+', b, '=', a+b        write(a, '+', b, '=', a+b);
```

В языке Паскаль можно использовать **форматный вывод**: после двоеточия указать общее количество знакомест, отводимое на число. Например, программа

```
a:=123;
write( a:5 );
```

выведет значение целой переменной *a*, заняв ровно 5 знакомест:

␣␣123

Поскольку само число занимает только 3 знакоместа, перед ним выводятся два пробела, которые здесь мы обозначили как «␣».

Вопросы и задания

1. Сравните структуру программ на языке Паскаль и школьном алгоритмическом языке.
2. Что такое идентификатор?
3. Чем различаются правила построения имён в школьном алгоритмическом языке и в Паскале?
4. Как записываются комментарии на этих языках? Подумайте, как комментирование можно использовать при поиске ошибок в алгоритме.
5. Сравните операторы вывода в КуМире и в Паскале. Как выполняется переход на новую строку?
6. Что такое переменная? Как вы думаете, зачем нужно объявлять переменные?
7. Зачем нужен тип переменной? Почему нельзя записывать в переменную значение другого типа?
8. Какое значение записано в переменной сразу после объявления? Можно ли его использовать?
9. Как задать начальные значения переменных? Сравните школьный алгоритмический язык и Паскаль.
10. Что такое оператор присваивания?
11. Почему следует выводить на экран подсказку перед вводом данных?

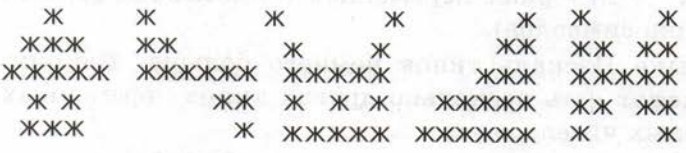
12. Подумайте, когда можно вычислять результат прямо в операторе вывода, а когда нужно заводить отдельную переменную.
13. Что такое форматный вывод? Как вы думаете, где он может быть полезен?

Подготовьте сообщение

а) «Операторы вывода в языке Си»
б) «Операторы вывода в языке Python»

Задачи

1. Используя оператор вывода, постройте на экране следующие рисунки из символов (такие рисунки называются псевдографикой):



2. Выберите правильные имена переменных (для школьного алгоритмического языка и Паскаля):

1	Vasya	СУ-27	@mail_ru
m11	Петя	СУ_27	lenta.ru
1m	Митин брат	_27	"Pes barbos"
m 1	Quo vadis	СУ(27)	<Ладья>

3. Пусть a и b — целые переменные. Что будет выведено в результате работы фрагмента программы?

а)
```
цел a=5, b=3
вывод a, '=Z(', b, ')'
```
```
a:=5; b:=3;
write(a,'=Z(',b,')');
```

б)
```
цел a=5, b=3
вывод 'Z(a)=', '(b)'
```
```
a:=5; b:=3;
write('Z(a)=','(b)');
```

в)
```
цел a=5, b=3
вывод 'Z(',a,')=(',a+b,')'
```
```
a:=5; b:=3;
write('Z(',a,')=(',a+b,')');
```

4. Запишите оператор для вывода значений целых переменных $a = 5$ и $b = 3$ в следующем формате:
   а) `3+5=?`
   б) `Z(5)=F(3)`
   в) `a=5;  b=3;`
   г) `Ответ:  (5;3)`

§ 56 Вычисления

Типы данных

Любая переменная имеет какой-либо тип, т. е. может хранить данные только того типа, который был указан при её объявлении.

В школьном алгоритмическим языке используются такие типы:

- **цел** — целые значения;
- **вещ** — вещественные значения;
- **лог** — логические значения (**да** или **нет**);
- **сим** — символ;
- **лит** — литерная переменная (символьная строка, т. е. цепочка символов).

В языке Паскаль типов немного больше. Например, кроме типа **integer** есть несколько других типов переменных для хранения целых чисел:

- **byte** — целые числа в диапазоне[1] 0..255;
- **shortint** — целые числа в диапазоне –128..127;
- **word** — целые числа в диапазоне 0..65535;
- **longint** — целые числа в диапазоне –2147483648..2147483647.

На переменную типа **byte** и **shortint** в памяти выделяется 1 байт, на переменную типа word — 2 байта, на переменную типа **longint** — 4 байта. Размер переменных типа **integer** зависит от версии языка (2 или 4 байта).

Для хранения вещественных переменных тоже существует несколько типов:

- **single** — число одинарной точности (4 байта);
- **real** — классический тип языка Паскаль (6 байтов);
- **double** — число двойной точности (8 байтов);
- **extended** — число расширенной точности (10 байтов).

Как мы обсуждали в главе 4, большинство вещественных чисел хранится в памяти неточно, и в результате операций с ними накапливается вычислительная ошибка. Поэтому для работы с целочисленными данными не нужно использовать вещественные переменные.

---

[1] Диапазон состоит из всех целых чисел от минимального до максимального значения (включая обе эти границы).

Логические переменные в Паскале относятся к типу **boolean** и принимают значения `True` (истина) и `False` (ложь).

Переменные типа **char** могут хранить один символ (точнее, его код) размером 1 байт. Символьные строки относятся к типу **string** (во многих версиях Паскаля длина строки не может быть больше 255 символов).

Арифметические выражения и операции

В языках программирования используется линейная запись арифметических выражений (без многоэтажных дробей). В школьном алгоритмическом языке выражение записывается в одну строку, а в Паскале запись можно переносить на следующие строки.

Арифметические выражения могут содержать константы (постоянные значения), имена переменных, знаки арифметических операций, круглые скобки (для изменения порядка действий) и вызовы функций. Например:

```
a:=(c+5-1)/2*d                a:=(c+5-1)/2*d;
```

При определении порядка действий используется **приоритет** (старшинство) операций. Они выполняются в следующем порядке:

- действия в скобках;
- умножение и деление, слева направо;
- сложение и вычитание, слева направо.

Таким образом, умножение и деление имеют одинаковый приоритет, более высокий, чем сложение и умножение. Поэтому в приведённом примере значение выражения, заключённого в скобки, сначала разделится на 2, а потом умножится на `d`.

Если в выражение входят переменные разных типов, в некоторых случаях происходит автоматическое приведение типа к более «широкому». Например, результат умножения целого числа на вещественное — это вещественное число. Переход к более «узкому» типу автоматически не выполняется, поэтому, например, вещественное значение нельзя записать в целую переменную. Нужно помнить, что результат деления (операции «/») — это вещественное число, даже если делимое и делитель — целые и делятся друг на друга нацело[1].

Часто нужно получить целый результат деления целых чисел и остаток от деления. В этом случае в школьном алгоритмиче-

[1] В некоторых языках, например в Си, это не так: при делении целых чисел получается целое число и остаток отбрасывается.

ском языке используют функции div и mod, а в Паскале — одноимённые операции (они имеют такой же приоритет, как умножение и деление):

```
d:= 85                    d:= 85;
a:=div(d,10) | = 8        a:=d div 10; { = 8 }
b:=mod(d,10) | = 5        b:=d mod 10; { = 5 }
```

Нужно учитывать, что для отрицательных чисел эти операции по-разному выполняются в разных языках. Например, внешне следующие программы аналогичны:

```
вывод div(-7,2), ','      write(-7 div 2, ',');
вывод mod(-7,2)           write(-7 mod 2);
```

Но программа на школьном алгоритмическом языке выведет результат «–4,1», а программа на Паскале — «–3,–1». Дело в том, что с точки зрения теории чисел остаток — это неотрицательное число, поэтому $-7 = (-4) \cdot 2+1$, т. е. частное от деления (–7) на 2 равно –4, а остаток равен 1. Поэтому в среде КуМир эти операции выполняются математически правильно. В то же время во многих языках (например, в Паскале и в Си) при целочисленном делении используется модуль числа, а затем к частному и остатку добавляется знак «минус»:

$$7 = 3 \cdot 2 + 1 \Rightarrow -7 = (-3) \cdot 2 - 1.$$

При таком подходе частное от деления (–7) на 2 равно –3, а результат операции mod равен –1.

В школьном алгоритмическом языке есть операция возведения в степень, которая обозначается двумя звездочками: «**». Например, выражение $y = 2x^2 + z^3$ запишется так:

```
y:= 2*x**2+z**3
```

Возведение в степень имеет более высокий приоритет, чем умножение и деление. В языке Паскаль операции возведения в степень нет.

Вещественные значения

При записи **вещественных чисел** в программе целую и дробную части разделяют не запятой (как принято в отечественной математической литературе), а точкой. Например:

```
вещ x                     var x: double;
x:= 123.456               ...
                          x:= 123.456;
```

Вещественное значение нельзя записывать в целочисленную переменную.

В языке Паскаль при выводе на экран вещественных значений по умолчанию используется так называемый **научный** (или **экспоненциальный**) формат, предназначенный для записи как очень больших, так и очень маленьких чисел. Например, программа

```
a:=1;
write(a/3);
```

выведет на экран результат в виде

```
 3.333333E-001
```

что означает $3{,}333333 \cdot 10^{-1} = 0{,}3333333$, т. е. до буквы E указывают значащую часть числа, а после неё — порядок (см. § 29). Для того чтобы вывести вещественное число в «нормальном» виде, используют форматный вывод: после двоеточия записывают общее количество знакомест, которое нужно отвести на число, а затем, снова через двоеточие — количество знаков в дробной части. Например, программа

```
a:=1;
write(a/3:7:3);
```

выводит то же значение в 7 позициях с тремя знаками в дробной части:

```
␣␣0.333
```

Поскольку эта запись занимает 5 позиций (а под неё отведено 7 позиций), перед числом добавляются два пробела, обозначенные знаком «␣».

Стандартные функции

В арифметических выражениях можно использовать стандартные математические функции, например:

`abs(x)` — модуль числа x;
`sqrt(x)` — квадратный корень из числа x (для $x \geq 0$);
`sin(x)` — синус угла x, заданного в радианах;
`cos(x)` — косинус угла x, заданного в радианах;
`exp(x)` — экспонента числа x, т. е. e^x, где $e \approx 2{,}718$ — основание натуральных логарифмов;
`ln(x)` — натуральный логарифм числа x (для $x > 0$).

Последние две функции позволяют выполнить возведение в степень в Паскале, используя равенство $x^y = e^{y \ln x}$.



Существуют функции для перехода от вещественных значений к целым. В языке Паскаль есть функции:

`trunc(x)` — отбрасывание дробной части вещественного числа x;

`round(x)` — округление вещественного числа x до ближайшего целого.

В школьном алгоритмическом языке есть функция выделения целой части числа: `int(x)`. Нужно иметь в виду, что функция `trunc` в Паскале отбрасывает дробную часть, а функция `int` в школьном алгоритмическом языке находит целую часть по правилам математики (как наибольшее целое число, не превосходящее данное). Поэтому при работе с отрицательными числами они могут давать разный результат:

```
вывод int(-1.5) | = -2      write(trunc(-1.5)); { = -1 }
```

Функция `frac(x)` в языке Паскаль выделяет дробную часть вещественного числа x.

Случайные числа

В некоторых задачах необходимо моделировать случайные события, например результат бросания игрального кубика (на нём может выпасть число от 1 до 6). Как сделать это на компьютере, который по определению «неслучаен», т. е. строго выполняет заданную ему программу?

Случайные числа — это последовательность чисел, в которой невозможно предсказать следующее число, даже зная все предыдущие. Чтобы получить истинно случайные числа, можно, например, бросать игральный кубик или измерять какой-то естественный шумовой сигнал (например, радиошум или электромагнитный сигнал, принятый из космоса). На основе этих данных составлялись и публиковались таблицы случайных чисел, которые использовали в разных областях науки.

Вернёмся к компьютерам. Ставить сложную аппаратуру для измерения естественных шумов или космического излучения на каждый компьютер очень дорого, и повторить эксперимент будет невозможно — завтра все значения будут уже другими. Существующие таблицы слишком малы, когда, скажем, нужно получать 100 случайных чисел каждую секунду. Для хранения больших таблиц требуется много памяти.

Чтобы выйти из положения, математики придумали алгоритмы получения **псевдослучайных** («как бы случайных») **чисел**. Для «постороннего» наблюдателя псевдослучайные числа практически неотличимы от случайных, но они вычисляются по некоторой математической формуле[1]: зная первое число («зерно»), можно по формуле вычислить второе, затем третье и т. п.

В школьном алгоритмическом языке существуют две функции для получения случайных (т. е. псевдослучайных) чисел в заданном интервале:

`rand(a,b)` — случайное вещественное число в полуинтервале $[a, b)$;
`irand(a,b)` — случайное целое число на отрезке $[a, b]$.

В Паскале есть аналогичные функции:

`random` — случайное вещественное число в полуинтервале $[0, 1)$ (вызов функции без параметров);
`random(N)` — случайное целое число на отрезке $[0, N-1]$;

Для того чтобы записать в целую переменную `n` случайное число в интервале от 1 до 6 (результат бросания кубика), можно использовать такие операторы:

```
n:=irand(1, 6)          n:=random(6)+1;
```

Вещественное случайное число в полуинтервале от 5 до 12 (не включая 12) получается так:

```
x:=rand(5, 12)          x:=7*random+5;
```

Вопросы и задания

1. Какие типы данных вы знаете?
2. Почему во многих языках программирования есть несколько целочисленных и вещественных типов данных?
3. Чем отличается символьная переменная от строковой?
4. Какие данные записываются в логические переменные? Сколько места в памяти они занимают?

---

[1] В стандартные библиотеки языков программирования обычно входит *линейный конгруэнтный генератор* псевдослучайных целых чисел, использующий формулу X_{k+1} = остаток от деления $(aX_k + c)$ на m, где X_{k+1} и X_k — следующее и предыдущее псевдослучайные числа; a, c и m — целые числа, подобранные так, чтобы в получаемой цепочке чисел было как можно меньше закономерностей. Например, в трансляторе Borland Delphi использовались значения $a = 134775813$, $c = 1$ и $m = 2^{32}$.

5. Что такое приоритет операций? Зачем он нужен?
6. В каком порядке выполняются операции, если они имеют одинаковый приоритет?
7. Зачем в выражениях используются скобки?
8. Что происходит, если в выражение входят переменные разных типов? Какого типа будет результат?
9. Опишите операции `div` и `mod`. Подумайте, почему они не определены для вещественных чисел.
10. Расскажите о проблеме вычисления остатка в различных языках программирования. Обсудите в классе этот вопрос.
11. Как выполняется операция возведения в степень?
12. Какие стандартные математические функции вы знаете? В каких единицах задаётся аргумент тригонометрических функций?
13. Как выполнить округление (к ближайшему целому) в школьном алгоритмическом языке?
14. Какие числа называют случайными? Зачем они нужны?
15. Как получить «естественное» случайное число? Почему такие числа почти не используются в цифровой технике?
16. Чем отличаются псевдослучайные числа от случайных?
17. Какие функции для получения псевдослучайных чисел вы знаете?

Подготовьте сообщение

а) «Типы данных и переменные в языке Си»
б) «Типы данных и переменные в языке Javascript»
в) «Типы данных и переменные в языке Python»
г) «Вычисление остатка от деления в языках программирования»
д) «Датчики псевдослучайных чисел»

Задачи

1. Найдите в справочной системе или в Интернете диапазон значений для вещественных типов данных.
2. Напишите программу, которая находит сумму, произведение и среднее арифметическое трёх целых чисел, введённых с клавиатуры. Например, при вводе чисел 4, 5 и 7 мы должны получить ответ

   $4 + 5 + 7 = 16$, $4 \cdot 5 \cdot 7 = 140$, $(4 + 5 + 7)/3 = 5{,}333333$.
3. Напишите программу, которая вводит радиус круга и вычисляет площадь этого круга и длину окружности. На языке Паскаль можно использовать встроенную константу `Pi`, равную числу π.
4. Напишите программу, которая меняет местами значения двух переменных в памяти.

*5. В задаче 4 попробуйте найти решение, которое не использует дополнительные переменные.

6. Напишите программу, которая возводит введённое число в степень 10, используя только операции сложения и умножения. Что произойдёт, если ввести большое число, например 78? Попытайтесь объяснить полученный результат.
7. Вычислите значение вещественной переменной c при $a = 2$ и $b = 3$:

а)
```
c:=a+1/3
```

б)
```
c:=a+4/2*3+6
```

в)
```
c:=a+4)/2*3
```

г)
```
c:=(a+4)/(b+3)*a
```

8. Вычислите значение целочисленной переменной c при $a = 26$ и $b = 6$:

а)
```
c:=mod(a,b)+b
```
```
c:=a mod b + b;
```

б)
```
c:=div(a,b)+a
```
```
c:=a div b + a;
```

в)
```
b:=div(a,b)
c:=div(a, b)
```
```
b:=a div b;
c:=a div b;
```

г)
```
b:=div(a,b)+b
c:=mod(a,b)+a
```
```
b:=a div b + b;
c:=a mod b + a;
```

д)
```
b:=mod(a,b)+4
c:=mod(a,b)+1
```
```
b:=a mod b + 4;
c:=a mod b + 1;
```

е)
```
b:=div(a,b)
c:= mod(a,b+1)
```
```
b:=a div b;
c:=a mod (b+1);
```

ж)
```
b:=mod(a,b)
c:=div(a,b+1)
```
```
b:=a mod b;
c:=a div (b+1);
```

9. Выполните задание 8 при $a = -22$ и $b = 4$. Чем различаются результаты работы программ на школьном алгоритмическом языке и на Паскале?
10. Напишите программу, которая вводит трёхзначное число и разбивает его на цифры. Например, при вводе числа 123 программа должна вывести «1,2,3».
11. Напишите программу, которая вводит координаты двух точек на числовой оси и выводит расстояние между ними. Учитывайте, что первой может быть введена меньшая координата.
12. Напишите программы на обоих языках, которые вводят два вещественных числа (x и y) и вычисляют значение x^y.
13. Напишите программу на школьном алгоритмическом языке, которая округляет вещественное число до ближайшего целого.
14. Напишите программу, которая вводит два целых числа, a и b ($a < b$) и выводит на экран 5 случайных целых чисел на отрезке $[a, b]$.
15. Напишите программу, которая моделирует бросание двух игральных кубиков: при запуске выводит случайное число в диапазоне от 2 до 12.

16. Напишите программу, которая случайным образом выбирает дежурных: выводит два случайных числа в диапазоне от 1 до N, где N — количество учеников вашего класса. С какой проблемой вы можете столкнуться?
17. Напишите программу, которая вводит два вещественных числа, a и b ($a < b$) и выводит на экран 5 случайных вещественных чисел в полуинтервале $[a, b)$.

§ 57 Ветвления

Условный оператор

Возможности, описанные в предыдущих параграфах, позволяют писать **линейные программы**, в которых операторы выполняются последовательно друг за другом, и порядок их выполнения не зависит от входных данных.

В большинстве реальных задач порядок действий может несколько изменяться, в зависимости от того, какие данные поступили. Например, программа для системы пожарной сигнализации должна выдавать сигнал тревоги, если данные с датчиков показывают повышение температуры или задымлённость.

Для этой цели в языках программирования предусмотрены **условные операторы**. Например, для того чтобы записать в переменную M максимальное из значений переменных a и b, можно использовать оператор:

```
если a>b то
  M:=a
иначе
  M:=b
все
```

```
if a>b then
  M:=a
else
  M:=b;
```

Видно, что запись на школьном алгоритмическом языке — это практически точный перевод записи ключевых слов языка Паскаль на русский язык. Обратите внимание, что в языке Паскаль перед ключевым словом **else** (иначе) точка с запятой не ставится.

В приведённом примере условный оператор записан в **полной форме**: в обоих случаях (истинно условие или ложно) нужно выполнить некоторые действия. Программа выбора максимального значения может быть написана иначе:

```
M:=a
если b>a то
  M:=b
все
```

```
M:=a;
if b>a then
  M:=b;
```

Здесь использован условный оператор в **неполной форме**, потому что в случае, когда условие ложно, ничего делать не требуется (нет слова **иначе** и операторов после него).

Для того чтобы сделать текст программы более понятным, всё тело условного оператора сдвинуто вправо. Вообще говоря, это не обязательно: в Паскале вообще вся программа может быть записана в одну строку, а в школьном алгоритмическом языке важно только разбиение программы на строки, а отступы игнорируются. Тем не менее запись с отступами значительно повышает читаемость программ, и мы далее будем её использовать[1].

Часто при каком-то условии нужно выполнить сразу несколько действий. Например, в задаче сортировки значений переменных a и b по возрастанию нужно поменять местами значения этих переменных, если $a > b$:

```
если a>b то
  c:=a
  a:=b
  b:=c
все
```

```
if a>b then begin
  c:=a;
  a:=b;
  b:=c
end;
```

В школьном алгоритмическом языке форма записи совсем не меняется, а на Паскале после ключевого слова **then** нужно записать **составной оператор**, в котором между словами **begin** и **end** может быть сколько угодно команд.

Кроме знаков < , > в условиях можно использовать другие знаки отношений: <= (меньше или равно), >= (больше или равно), = (равно) и <> (не равно, два знака < , > без пробела).

Внутри условного оператора могут находиться любые операторы, в том числе и другие условные операторы. Например, пусть возраст Андрея записан в переменной a, а возраст Бориса — в переменной b. Нужно определить, кто из них старше. Одним условным оператором тут не обойтись, потому что есть три возможных

[1] В некоторых языках, например в языке Python, отступы обязательны и все строки одного уровня вложенности должны иметь одинаковые отступы.

результата: старше Андрей, старше Борис и оба одного возраста. Решение задачи можно записать так:

```
если a>b то
  вывод 'Андрей старше'
иначе
  если a=b то
    вывод 'Одного возраста'
  иначе
    вывод 'Борис старше'
  все
все
```

```
if a>b then
  writeln('Андрей старше')
else
  if a=b then
    writeln('Одного возраста')
  else
    writeln('Борис старше');
```

Условный оператор, проверяющий равенство, находится внутри блока **иначе (else)**, поэтому он называется **вложенным условным оператором**. Как видно из этого примера, использование вложенных условных операторов позволяет выбрать один из нескольких (а не только из двух) вариантов.

При работе с вложенными условными операторами в языке Паскаль нужно помнить правило: любой блок **else** относится к ближайшему предыдущему оператору **if**, у которого такого блока ещё не было. Например, оператор

```
if a>b then write('A') else if a=b then write('=')
else write('Б');
```

может быть записан с выделением структуры так:

```
if a>b then
  write('A')
else
  if a=b then write('=')
         else write('Б');
```

Здесь второй блок **else** относится к ближайшему (второму, вложенному) условному оператору, поэтому буква Б будет выведена только тогда, когда оба условия окажутся ложными.

Сложные условия

Предположим, что некоторая фирма набирает сотрудников, возраст которых — от 25 до 40 лет включительно. Нужно написать программу, которая запрашивает возраст претендента и выдает ответ: подходит он или не подходит по этому признаку.

В качестве условия в условном операторе можно указать любое логическое выражение, в том числе сложное условие, составленное из простых отношений с помощью логических операций (связок) «И», «ИЛИ» и «НЕ» (см. главу 3). На языке Паскаль они записываются на английском языке: **and**, **or** и **not**.

Пусть в переменной v записан возраст сотрудника. Тогда нужная программа будет выглядеть так:

```
если v >= 25 и v <= 40 то
  вывод 'подходит'
иначе
  вывод 'не подходит'
все
```

```
if (v >= 25) and (v <= 40) then
  writeln('подходит')
else
  writeln('не подходит');
```

Обратите внимание, что в Паскале каждое простое условие заключается в скобки. Это связано с тем, что в этом языке отношения имеют более низкий приоритет, чем логические операции, которые в обоих языках выполняются в таком порядке: сначала все операции «НЕ», затем — «И», и в самом конце — «ИЛИ» (во всех случаях слева направо). Для изменения порядка действий используют круглые скобки.

В языке Паскаль есть операция «исключающее ИЛИ» (**xor**), которая имеет такой же приоритет, что и операция «ИЛИ».

Множественный выбор

Условный оператор предназначен, прежде всего, для выбора одного из двух вариантов (простого ветвления). Иногда нужно сделать выбор из нескольких возможных вариантов.

Пусть, например, в переменной m хранится номер месяца, и нужно вывести на экран его русское название. Конечно, в этом случае можно использовать 12 условных операторов:

```
если m=1 то
  вывод 'январь' все
если m=2 то
  вывод 'февраль' все
...
если m=12 то
  вывод 'декабрь' все
```

```
if m=1 then
  write('январь');
if m=2 then
  write('февраль');
...
if m=12 then
  write('декабрь');
```

Вместо многоточий могут быть записаны аналогичные операторы для остальных значений m. Но во многих языках программирования для подобных случаев есть специальный **оператор выбора:**

```
выбор
  при m=1: вывод 'январь'
  при m=2: вывод 'февраль'
  ...
  при m=12: вывод 'декабрь'
иначе вывод 'ошибка'
все
```

```
case m of
  1: write('январь');
  2: write('февраль');
  ...
  12: write('декабрь')
else write('ошибка')
end;
```

Кроме очевидных 12 блоков здесь добавлен ещё один, который сигнализирует об ошибочном номере месяца. Он начинается ключевым словом **иначе** (в Паскале — **else**).

В школьном алгоритмическом языке для выбора можно использовать любые условия (а не только равенство). Например, следующий оператор записывает в переменную *sgn* знак значения переменной x:

```
выбор
  при x<0: sgn:=-1
  при x=0: sgn:=0
  при x>0: sgn:=1
все
```

В Паскале можно через запятую указывать список значений, для которых выполняются одинаковые действия. Например, программа для определения количества дней в месяце (для невисокосного года) может быть записана так:

```
case m of
  2: d:=28;
  1,3,5,7,8,10,12: d:=31
  else d:=30
end;
```

Допускаются также интервалы (диапазоны), в них начальное и конечное значения отделены двумя точками. Следующая программа выводит социальный статус человека в зависимости от возраста:

```
case v of
  0..6:  write('дошкольник');
  7..17: write('школьник')
  else   write('взрослый')
end;
```

Если в каком-то из вариантов нужно выполнить несколько действий, в языке Паскаль используется составной оператор: нужные команды заключаются в **операторные скобки begin** и **end**.

Подготовьте сообщение

а) «Условные операторы и операторы выбора в языке Си»
б) «Условные операторы и операторы выбора в языке Python»

Вопросы и задания

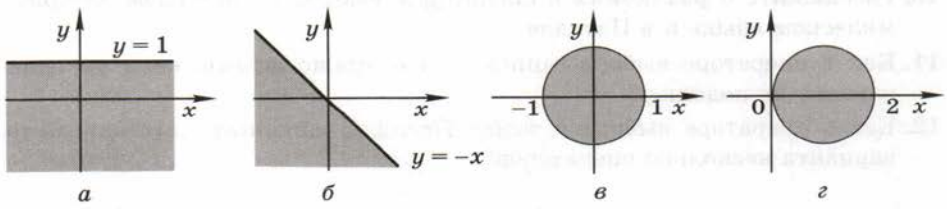
1. Чем отличаются разветвляющиеся алгоритмы от линейных?
2. Как вы думаете, почему не все задачи можно решить с помощью линейных алгоритмов? Приведите примеры таких задач.
3. Как вы думаете, хватит ли линейных алгоритмов и ветвлений для разработки любой программы?
4. Почему нельзя выполнить обмен значений двух переменных в два шага: `a:=b; b:=a`?
5. Чем различаются условные операторы в полной и неполной формах? Как вы думаете, можно ли обойтись только неполной формой?
6. Какие отношения вы знаете? Как обозначаются отношения «равно» и «не равно»?
7. Что такое сложное условие?
8. Как определяется порядок вычислений в сложном условии? Расскажите об особенностях вычисления логических выражений в языке Паскаль.
9. Зачем нужен оператор выбора? Как можно обойтись без него?
10. Расскажите о различиях в операторах выбора в школьном алгоритмическом языке и в Паскале.
11. Как в операторе выбора записать, что нужно делать, если ни один вариант не подошёл?
12. Как в операторе выбора в языке Паскаль выполнить для какого-то варианта несколько операторов?

Задачи

1. Покажите, что приведённая программа не всегда верно определяет максимальное из трёх чисел, записанных в переменные a, b и c:

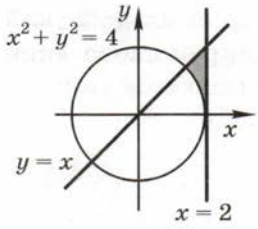
```
если a>b то M:=a                if a>b then M:=a
иначе      M:=b все             else        M:=b;
если c>b то M:=c                if c>b then M:=c
иначе      M:=b все             else        M:=b;
```

   Приведите контрпример, т. е. значения переменных, при котором в переменной M будет получен неверный ответ. Как нужно доработать программу, чтобы она всегда работала правильно?
2. Напишите программу, которая выбирает максимальное и минимальное из пяти введённых чисел.
3. Напишите программу, которая определяет, верно ли, что введённое число трёхзначное.
4. Напишите программу, которая вводит номер месяца и выводит название времени года. Оператор выбора использовать не разрешается. При вводе неверного номера месяца должно быть выведено сообщение об ошибке.
5. Решите предыдущую задачу с помощью оператора выбора.
6. Напишите программу, которая вводит с клавиатуры номер месяца и определяет, сколько дней в этом месяце. При вводе неверного номера месяца должно быть выведено сообщение об ошибке.
7. Напишите программу, которая вводит с клавиатуры номер месяца и день и определяет, сколько дней осталось до Нового года. При вводе неверных данных должно быть выведено сообщение об ошибке.
8. Напишите программу, которая вводит возраст человека (целое число, не превышающее 120) и выводит этот возраст со словом «год», «года» или «лет». Например: «21 год», «22 года», «25 лет».
9. Напишите программу, которая вводит целое число, не превышающее 100, и выводит его прописью, например: 21 → «двадцать один».
10. Напишите программу, которая вводит координаты точки на плоскости и определяет, попала ли эта точка в заштрихованную область.

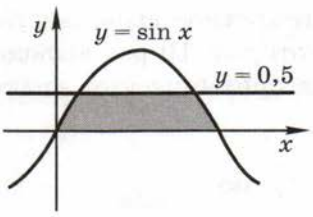


а *б* *в* *г*

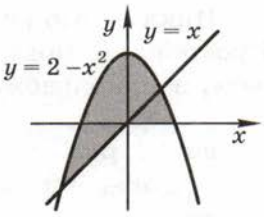
11. Напишите два варианта программы, которая вводит координаты точки на плоскости и определяет, попала ли эта точка в заштрихованную область. Один вариант программы должен использовать сложные условия, второй — обходиться без них.



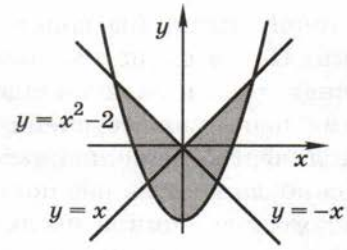
а



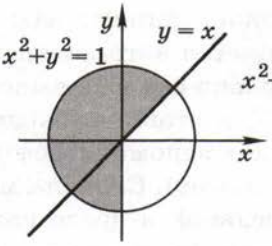
б



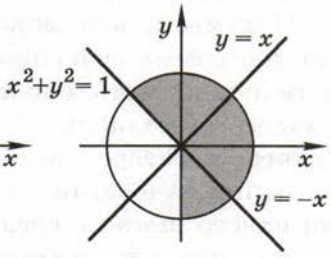
в



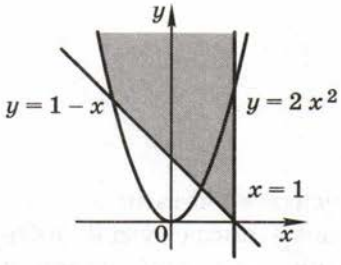
г



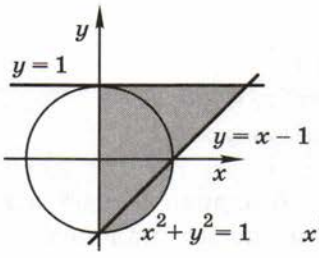
д



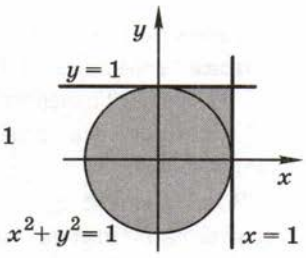
е



ж



з



и

§ 58 Циклические алгоритмы

Как организовать цикл?

Цикл — это многократное выполнение одинаковых действий. Простейший цикл, который 10 раз выводит на экран слово «привет», на школьном алгоритмическом языке записывается так:

```
нц 10 раз
  вывод 'привет', нс
кц
```

Здесь **нц** и **кц** — это сокращения от выражений «начало цикла» и «конец цикла».

Подумаем, как можно организовать такой цикл. Вы знаете, что программа выполняется автоматически. При этом на каждом шаге нужно знать, сколько раз уже выполнен цикл и сколько ещё осталось выполнить. Для этого необходимо использовать ячейку памяти, в которой будет запоминаться количество выполненных шагов цикла (счётчик шагов). Сначала можно записать в неё ноль (ни одного шага не сделано), а после каждого шага цикла увеличивать значение ячейки на единицу. На псевдокоде алгоритм можно записать так (здесь и далее операции, входящие в тело цикла, выделяются отступами):

```
счётчик:=0
пока счётчик<10
  вывод 'привет', нс
  увеличить счётчик на 1
```

Возможен и другой вариант: сразу записать в счётчик нужное количество шагов и после каждого шага цикла уменьшать счётчик на 1. Тогда цикл должен закончиться при нулевом значении счётчика:

```
счётчик:=10
пока счётчик>0
  вывод 'привет', нс
  уменьшить счётчик на 1
```

В этих примерах мы использовали **цикл с условием**, который выполняется до тех пор, пока некоторое условие не станет ложным.

Циклы с условием

Рассмотрим следующую задачу: определить количество цифр в десятичной записи целого положительного числа. Будем предполагать, что исходное число записано в целую переменную n.

Сначала составим алгоритм решения этой задачи. Чтобы считать что-то в программе, нужно использовать переменную, которую называют **счётчиком**. Для подсчёта количества цифр нужно как-то отсекать эти цифры по одной, с начала или с конца, каждый раз увеличивая счётчик. Начальное значение счётчика должно быть равно нулю, так как до выполнения алгоритма ещё не найдено ни одно цифры.

Для отсечения первой цифры необходимо заранее знать, сколько цифр в десятичной записи числа, т. е. нужно заранее решить ту задачу, которую мы решаем. Следовательно, этот метод не подходит.

Отсечь последнюю цифру проще — достаточно разделить число нацело на 10 (поскольку речь идёт о десятичной системе). Операции отсечения и увеличения счётчика нужно выполнять столько раз, сколько цифр в числе. Как же «поймать» момент, когда цифры кончатся? Несложно понять, что в этом случае результат очередного деления на 10 будет равен нулю, это и говорит о том, что отброшена последняя оставшаяся цифра. Изменение переменной n и счётчика для начального значения 1234 можно записать в виде таблицы на рис. 8.1. Псевдокод выглядит так:

n	*счётчик*
1234	0
123	1
12	2
1	3
0	4

Рис. 8.1

```
счётчик:=0
пока n>0
  отсечь последнюю цифру n
  увеличить счётчик на 1
```

Программа на школьном алгоритмическом языке очень похожа на псевдокод, а программа на Паскале почти совпадает с его переводом на английский язык:

```
count:=0
нц пока n>0
  n:=div(n,10)
  count:=count+1
кц
```

```
count:=0;
while n>0 do begin
  n:=n div 10;
  count:=count+1
end;
```

Здесь целочисленная переменная-счётчик имеет имя count. Отметим, что в Паскале после ключевого слова **do** стоит составной оператор, ограниченный ключевыми словами **begin** ... **end**. Если в теле цикла нужно выполнить только один оператор, эти операторные скобки можно не ставить.

Обратите внимание, что проверка условия выполняется в начале цикла. Такой цикл называется **циклом с предусловием** (т. е. с предварительной проверкой условия) или **циклом «пока»**. Если в начальный момент значение переменной n будет нулевое или отрицательное, цикл не выполнится ни одного раза.

В данном случае количество шагов цикла «пока» неизвестно, оно равно количеству цифр введённого числа, т. е. зависит от исходных данных. Цикл этого вида может быть использован и в том случае, когда число шагов известно заранее или может быть вычислено:

```
k:=0
нц пока k<10
  вывод 'Привет', нс
  k:=k+1
кц
```

```
k:=0;
while k<10 do begin
  writeln('Привет');
  k:=k+1
end;
```

Если условие в заголовке цикла никогда не нарушится, цикл будет работать бесконечно долго. В этом случае говорят, что *программа зациклилась*. Например, если забыть увеличить переменную *k* в рассмотренном выше примере, программа зациклится:

```
k:=0
нц пока k<10
  вывод 'Привет', нс
кц
```

```
k:=0;
while k<10 do begin
  writeln('Привет');
end;
```

Во многих языках программирования существует **цикл с постусловием**, в котором условие проверяется в конце цикла. Это полезно в том случае, когда нужно обязательно выполнить цикл хотя бы один раз. Например, пусть пользователь должен ввести с клавиатуры положительное число. Для того чтобы защитить программу от неверных входных данных, можно использовать цикл с постусловием:

```
нц
  вывод "Введите n>0: "
  ввод n
кц при n>0
```

```
repeat
  write('Введите n>0: ');
  read(n);
until n>0;
```

Этот цикл закончится тогда, когда станет истинным условие $n>0$ в последней строке, т. е. тогда, когда пользователь введёт допустимое значение. Обратите внимание на особенности этого вида цикла:

- при входе в цикл условие не проверяется, поэтому цикл всегда выполняется хотя бы один раз;
- в последней строке указывают условие окончания цикла (а не условие его продолжения).

Цикл с переменной

В информатике важную роль играют степени числа 2 (2, 4, 8, 16 и т. д.) Чтобы вывести все степени двойки от 2^1 до 2^{10}, мы уже можем написать такую программу с циклом «пока»:

```
k:=1
n:=2
нц пока k<=10
  вывод n, нс
  n:=n*2
  k:=k+1
кц
```

```
k:=1;
n:=2;
while k<=10 do begin
  writeln(n);
  n:=n*2;
  k:=k+1
end;
```

Вы наверняка заметили, что переменная k используется трижды (см. выделенные блоки): в операторе присваивания начального значения, в условии цикла и в теле цикла (увеличение на 1). Чтобы собрать все действия с ней в один оператор, во мно-

гие языки программирования введён особый вид цикла — **цикл с переменной**. В заголовке этого цикла задаются начальное и конечное значения этой переменной, а шаг её изменения по умолчанию равен 1:

```
n:=2
нц для k от 1 до 10
  вывод n, нс
  n:=n*2
кц
```

```
n:=2;
for k:=1 to 10 do begin
  writeln(n);
  n:=n*2
end;
```

Здесь, в отличие от цикла «пока», переменная цикла может быть только целой.

С каждым шагом цикла переменная цикла может не только увеличиваться, но и уменьшаться на 1. Для этого в школьном алгоритмическом языке добавляется параметр **шаг**, а в Паскале ключевое слово **to** заменяется на **downto** («движение вниз до»). Следующая программа выводит квадраты натуральных чисел от 10 до 1 в порядке убывания:

```
нц для k от 10 до 1 шаг -1
  вывод k*k, нс
кц
```

```
for k:=10 downto 1 do
  writeln(k*k);
```

В школьном алгоритмическом языке шаг изменения переменной цикла может быть любым целым числом, а в Паскале — только 1 или (–1).

Вложенные циклы

В более сложных задачах часто бывает так, что на каждом шаге цикла нужно выполнять обработку данных, которая также представляет собой циклический алгоритм. В этом случае получается конструкция «цикл в цикле» или **вложенный цикл**.

Предположим, что нужно найти все простые числа в диапазоне от 2 до 1000. Простейший (но не самый быстрый) алгоритм решения такой задачи на псевдокоде выглядит так:

```
нц для n от 2 до 1000
  если число n простое то
    вывод n, нс
  все
кц
```

Как же определить, что число простое? Как известно, простое число делится только на 1 и само на себя. Если число n не имеет делителей в диапазоне от 2 до $n-1$, то оно простое, а если хотя бы один делитель в этом интервале найден, то составное.

Чтобы проверить делимость числа n на некоторое число k, нужно взять остаток от деления n на k. Если этот остаток равен нулю, то n делится на k. Таким образом, программу можно записать так (здесь n, k и *count* — целочисленные переменные, *count* обозначает счётчик делителей):

```
нц для n от 2 до 1000
  count:=0
  нц для k от 2 до n-1
    если mod(n,k)=0 то
      count:=count+1
    все
  кц
  если count=0 то
    вывод n, нс
  все
кц
```

```
for n:=2 to 1000 do begin
  count:=0;
  for k:=2 to n-1 do
    if n mod k=0 then
      count:=count+1;
    if count=0 then
      writeln(n);
end;
```

Попробуем немного ускорить работу программы. Делители числа обязательно идут в парах, причём в любой паре меньший из делителей не превосходит $\sqrt{n}$ (так как произведение двух делителей, каждый из которых больше $\sqrt{n}$, будет больше, чем n). Поэтому внутренний цикл можно выполнять только до значения $\sqrt{n}$ вместо $n-1$. Для того чтобы работать только с целыми числами (и таким образом избежать вычислительных ошибок), лучше заменить условие $k \le \sqrt{n}$ на равносильное ему условие $k^2 \le n$. При этом потребуется перейти к внутреннему циклу с условием:

```
count:=0
k:=2
нц пока k*k<=n
  если mod(n,k)=0 то
    count:=count+1
  все
  k:=k+1
кц
```

```
count:=0;
k:=2;
while k*k<=n do begin
  if n mod k=0 then
    count:=count+1;
  k:=k+1
end;
```

Чтобы ещё ускорить работу цикла, заметим, что когда найден хотя бы один делитель, число уже заведомо составное, и искать другие делители в данной задаче не требуется. Поэтому можно закончить цикл. Для этого в условие работы цикла добавляется условие *count* = 0, связанное с имеющимся условием с помощью операции «И»:

```
нц пока k*k<=n и count=0
...
кц
```

```
while (k*k<=n) and (count=0)
  do begin
  ...
end;
```

В любом вложенном цикле переменная внутреннего цикла изменяется быстрее, чем переменная внешнего цикла. Рассмотрим, например, такой вложенный цикл:

```
нц для i от 1 до 4
  нц для k от 1 до i
  ...
  кц
кц
```

```
for i:=1 to 4 do begin
  for k:=1 to i do begin
  ...
  end
end;
```

На первом шаге (при $i = 1$) переменная k принимает единственное значение 1. Далее, при $i = 2$ переменная k принимает последовательно значения 1 и 2. На следующем шаге при $i = 3$ переменная k проходит значения 1, 2 и 3, и т. д.

Вопросы и задания

1. Что такое цикл?
2. Сравните цикл с переменной и цикл с условием. Какие преимущества и недостатки есть у каждого из них?
3. Что означает выражение «цикл с предусловием»?
4. В каком случае цикл с предусловием не выполняется ни разу?
5. В каком случае программа, содержащая цикл с условием, может зациклиться? Приведите пример такой программы.
6. В каком случае цикл с переменной не выполняется ни разу?
7. Верно ли, что любой цикл с переменной можно заменить циклом с условием? Верно ли обратное утверждение? Ответ обоснуйте.
8. В каком случае можно заменить цикл с условием на цикл с переменной?

9. Как будет работать приведённая в параграфе программа, которая считает количество цифр введённого числа, при вводе отрицательного числа? Если вы считаете, что она работает неправильно, укажите, как её нужно доработать.

Подготовьте сообщение

а) «Операторы цикла в языке Си»
б) «Операторы цикла в языке Python»

Задачи

1. Напишите программу, которая вводит два целых числа и находит их произведение, не используя операцию умножения. Учтите, что числа могут быть отрицательными.
2. Напишите программу, которая вводит натуральное число N и находит сумму всех натуральных чисел от 1 до N. Используйте сначала цикл с условием, а потом — цикл с переменной.
3. Напишите программу, которая вводит натуральное число N и выводит первые N чётных натуральных чисел.
4. Напишите программу, которая вводит натуральные числа a и b, и выводит квадраты натуральных чисел в диапазоне от a до b. Например, если ввести 4 и 5, программа должна вывести:

```
4*4=16
5*5=25
```

5. Напишите программу, которая вводит натуральные числа a и b, и выводит сумму квадратов натуральных чисел в диапазоне от a до b.
6. Напишите программу, которая вводит натуральное число N и выводит на экран N псевдослучайных чисел. Запустите её несколько раз, объясните результаты опыта.
7. Найдите все пятизначные числа, которые при делении на 133 дают в остатке 125, а при делении на 134 дают в остатке 111.
8. Напишите программу, которая вводит натуральное число N и выводит на экран все натуральные числа, не превосходящие N и делящиеся на каждую из своих цифр.
9. *Числа Армстронга.* Натуральное число называется числом Армстронга, если сумма цифр числа, возведённых в N-ю степень, где N — количество цифр в числе, равна самому числу. Например: $153 = 1^3 + 5^3 + 3^3$. Найдите все трёхзначные и четырёхзначные числа Армстронга.

10. *Автоморфные числа*. Натуральное число называется автоморфным, если оно равно последним цифрам своего квадрата. Например: $25^2 = 625$. Напишите программу, которая вводит натуральное число N и выводит на экран все автоморфные числа, не превосходящие N.
11. Напишите программу, которая считает количество чётных цифр введённого числа.
12. Напишите программу, которая считает сумму цифр введённого числа.
13. Напишите программу, которая определяет, верно ли, что введённое число содержит две одинаковые цифры, стоящие рядом (как, например, 221).
14. Напишите программу, которая определяет, верно ли, что введённое число состоит из одинаковых цифр (как, например, 222).
*15. Напишите программу, которая определяет, верно ли, что введённое число содержит по крайней мере две одинаковые цифры, возможно, не стоящие рядом (как, например, 212).
16. Используя сначала цикл с условием, а потом — цикл с переменной, напишите программу, которая выводит на экран чётные степени числа 2 от 2^{10} до 2^2 в порядке убывания.
17. *Алгоритм Евклида* для вычисления наибольшего общего делителя (НОД) двух натуральных чисел формулируется так: нужно заменять большее число на разность большего и меньшего до тех пор, пока одно из них не станет равно нулю; тогда второе и есть НОД. Напишите программу, которая реализует этот алгоритм. Какой цикл тут нужно использовать?
18. Напишите программу, использующую *модифицированный алгоритм Евклида*: нужно заменять большее число на остаток от деления большего на меньшее до тех пор, пока этот остаток не станет равен нулю; тогда второе число и есть НОД.
19. Добавьте в решение двух предыдущих задач вычисление количества шагов цикла. Заполните таблицу (в строки «шаги-1» и «шаги-2» записывается количество шагов для двух версий алгоритма Евклида).

a	64168	358853	6365133	17905514	549868978
b	82678	691042	11494962	23108855	298294835
НОД(a, b)					
шаги-1					
шаги-2					

20. Напишите программу, которая вводит с клавиатуры 10 чисел и вычисляет их сумму и произведение.

21. Напишите программу, которая вводит с клавиатуры числа до тех пор, пока не будет введено число 0. В конце работы программы на экран выводится сумма и произведение введённых чисел.

22. Напишите программу, которая вводит с клавиатуры числа до тех пор, пока не будет введено число 0. В конце работы программы на экран выводятся минимальное и максимальное из введённых чисел.

23. Напишите программу, которая вводит с клавиатуры натуральное число N и определяет его *факториал*, т. е. произведение натуральных чисел от 1 до N: $N! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot N$. Что будет, если ввести большое значение N (например, 20)?

24. Напишите программу, которая вводит натуральные числа A и N и вычисляет A^N.

25. Напишите программу, которая выводит на экран все цифры числа, начиная с первой.

26. Ряд чисел Фибоначчи задаётся следующим образом: первые два числа равны 1 ($F_1 = F_2 = 1$), а каждое следующее равно сумме двух предыдущих: $F_n = F_{n-1} + F_{n-2}$. Напишите программу, которая вводит натуральное число N и выводит на экран первые N чисел Фибоначчи.

27. Напишите программу, которая вводит натуральные числа a и b и выводит все простые числа в диапазоне от a до b.

28. *Совершенным* называется число, равное сумме всех своих делителей, меньших его самого (например, число 6 = 1 + 2 + 3). Напишите программу, которая вводит натуральное число N и определяет, является ли число N совершенным.

29. Напишите программу, которая вводит натуральное число N и находит все совершенные числа в диапазоне от 1 до N.

30. В магазине продаётся мастика в ящиках по 15, 17, 21 кг. Как купить ровно 185 кг мастики, не вскрывая ящики? Сколькими способами можно это сделать?

***31.** Ввести натуральное число N и вывести значение числа $1/N$, выделив период дроби. Например: 1/2 = 0,5 или 1/7 = 0,(142857).

***32.** В телевикторине участнику предлагают выбрать один из трёх закрытых чёрных ящиков, причём известно, что в одном из них приз, а в двух других пусто. После этого ведущий открывает один пустой ящик (но не тот, который выбрал участник) и предлагает заново сделать выбор, но уже между двумя оставшимися ящиками. Используя псевдослучайные числа, выполните моделирование 1000 раундов этой игры и определите, как следует поступить участнику, чтобы с большей вероятностью получить приз: выбрать тот же ящик, что и в начале игры, или другой.

§ 59 Процедуры

Что такое процедура?

Предположим, что в нескольких местах программы требуется выводить на экран сообщение об ошибке: Ошибка программы. Это можно сделать, например, так:

```
вывод "Ошибка программы"
```

```
write('Ошибка программы');
```

Конечно, можно вставить этот оператор вывода везде, где нужно вывести сообщение об ошибке. Но тут есть две сложности. Во-первых, при этом строка-сообщение будет храниться в памяти много раз. Во-вторых, если мы задумаем поменять текст сообщения, нужно будет искать эти операторы вывода по всей программе. Для таких случаев в языках программирования предусмотрены **процедуры — вспомогательные алгоритмы**, которые выполняют некоторые действия.

```
алг С процедурой
нач
  цел n
  ввод n
  если n<0 то Error все
  ...
кон

алг Error
нач
  вывод 'Ошибка программы'
кон
```

```
program withProc;
var n: integer;
procedure Error;
begin
  writeln('Ошибка программы')
end;
begin
  read(n);
  if n<0 then Error;
  ...
end.
```

Обратим внимание на разницу оформления программы с процедурой в школьном алгоритмическом языке и в Паскале. В школьном алгоритмическом языке процедура оформляется точно так же, как и основной алгоритм, но размещается после основной программы.

В Паскале процедура начинается с ключевого слова **procedure**, тело процедуры начинается с ключевого слова **begin** и заканчивается ключевым словом **end** с точкой с запятой. Процедура располагается после блока объявления переменных, но выше основной программы.

Фактически мы ввели в язык программирования новую команду `Error`, которая была расшифрована прямо в теле программы. Для того чтобы процедура заработала, в основной программе (или в другой процедуре) необходимо ее **вызвать** по имени.

Как мы видели, использование процедур сокращает код, если какие-то операции выполняются несколько раз в разных местах программы. Кроме того, иногда большую программу разбивают на несколько процедур для удобства, оформляя в виде процедур отдельные этапы сложного алгоритма. Такой подход делает всю программу более понятной.

Процедура с параметрами

Процедура `Error` при каждом вызове делает одно и то же. Более интересны процедуры, которым можно передавать **параметры** (аргументы) — дополнительные данные, которые изменяют выполняемые действия.

Предположим, что в программе требуется многократно выводить на экран запись целого числа (0..255) в 8-битном двоичном коде. Старшая цифра в такой записи — это частное от деления числа на 128. Далее возьмём остаток от этого деления и разделим на 64 — получается вторая цифра и т. д. Алгоритм, решающий эту задачу для переменной n, можно записать так:

```
k:=128
нц пока k>0
  вывод div(n,k)
  n:= mod(n,k)
  k:= div(k,2)
кц
```

```
k:=128;
while k>0 do begin
  write(n div k);
  n:= n mod k;
  k:= k div 2
end;
```

Писать такой цикл каждый раз, когда нужно вывести двоичное число, очень утомительно. Кроме того, легко сделать ошибку или опечатку, которую будет сложно найти. Поэтому лучше оформить этот вспомогательный алгоритм в виде процедуры. Но этой

процедуре нужно передать значение параметра — число для перевода в двоичную систему. Программа получается такой:

```
алг Двоичный код
нач
  printBin(99)
кон
алг printBin(цел n0)
нач
  цел n, k
  n:= n0
  k:= 128
  нц пока k>0
    вывод div(n,k)
    n:= mod(n,k)
    k:= div(k,2)
  кц
кон
```

```
program binCode;
procedure printBin(n: integer);
var k: integer;
begin
  k:= 128;
  while k>0 do begin
    write(n div k);
    n:= n mod k;
    k:= k div 2
  end
end;
begin
  printBin(99)
end.
```

Основная программа содержит всего одну команду — вызов процедуры `printBin` для значения 99. В заголовке процедуры в скобках записывают тип и внутреннее имя параметра (т. е. имя, по которому к нему можно обращаться в процедуре). Значение параметра, переданное из вызывающей программы в процедуру (в этом примере — число 99), обычно называют **аргументом**.

В процедуре объявлена **локальная** (внутренняя) **переменная** k — она известна только внутри этой процедуры. Обратите внимание, что в школьном алгоритмическом языке локальные переменные объявляются после ключевого слова **нач**, а в языке Паскаль — до слова **begin**.

В школьном алгоритмическом языке запрещено изменять параметры процедуры внутри процедуры, поэтому мы назвали параметр $n0$, а в начале процедуры скопировали его значение в переменную n.

Параметров может быть несколько, в этом случае они перечисляются в заголовке процедуры через запятую (в школьном алгоритмическом языке) или точку с запятой (в Паскале). Например, процедуру, которая выводит экран среднее арифметическое двух целых чисел, можно записать так:

```
алг printSred(цел a,
              цел b)
нач
  вывод (a+b)/2
кон
```

```
procedure printSred
                (a: integer;
                 b: integer);
begin
  write((a+b)/2);
end.
```

Если несколько параметров одного типа стоят в списке один за другим, их можно определить списком:

```
алг printSred(цел a, b)
нач
  вывод (a+b)/2
кон
```

```
procedure printSred(a, b: integer);
begin
  write((a+b)/2);
end.
```

Изменяемые параметры

Напишем процедуру, которая меняет местами значения двух переменных. Проще всего для этого использовать третью переменную (пока напишем программу только на Паскале):

```
program Exchange;
var x, y: integer;
procedure Swap(a, b: integer);
var c: integer;
begin
  c:=a; a:=b; b:=c
end;
begin
  x:=2; y:=3;
  Swap(x, y);
  write(x, ' ', y)
end.
```

После запуска этой программы обнаружится, что значения переменных x и y остались прежними: на экран будет выведено: 2 3. Дело в том, что эта процедура работает с *копиями* переданных ей параметров. Это значит, что процедура Swap создаёт в памяти вре́менные локальные переменные с именами a и b и копирует в них переданные значения переменных x и y основной программы. Поэтому и все перестановки в нашей программе были

сделаны именно с копиями, а значения переменных x и y не изменились. Такая передача параметров называется **передачей по значению**.

Чтобы решить проблему, нужно явно сказать, чтобы процедура работала с теми же ячейками памяти, что и основная программа. Для этого в Паскале в заголовке процедуры перед именем изменяемого параметра пишут ключевое слово **var**:

```
procedure Swap(var a, b: integer);
```

Теперь процедура решает поставленную задачу: на выходе мы увидим: 3 2, что и требовалось. В подобных случаях говорят, что параметры **передаются по ссылке**, а не по значению. Это означает, что фактически в процедуру передаётся адрес переменной и можно изменять значение этой переменной, записывая новые данные по этому адресу.

При вызове процедуре Swap можно передавать только переменные, но не константы (постоянные) и не арифметические выражения. Например, вызовы Swap(2, 3) и Swap(x, y+3) противоречат правилам языка программирования, и программа выполняться не будет.

В школьном алгоритмическом языке все параметры делятся на **аргументы** (исходные данные, обозначаются **арг**) и **результаты** (ключевое слово **рез**, эти значения процедура передаёт вызывающей программе). По умолчанию (если не указано иначе) все параметры считаются аргументами. Поэтому два следующих заголовка равносильны:

```
алг Swap(цел a, b)
```

и

```
алг Swap(арг цел a, b)
```

В нашем случае параметры процедуры одновременно являются и аргументами, и результатами, поэтому их нужно объявлять с помощью ключевого слова **аргрез**. Приведём полную программу:

```
алг Обмен
нач
  цел x=2, y=3
  Swap(x, y)
  вывод x, ' ', y
кон
```

```
алг Swap(аргрез цел a, b)
нач
  цел c
  c:=a; a:=b; b:=c
кон
```

Вопросы и задания

1. Что такое процедуры? В чём смысл их использования?
2. Как оформляются процедуры в школьном алгоритмическом языке и в Паскале?
3. Достаточно ли включить процедуру в текст программы, чтобы она «сработала»?
4. Что такое параметры? Зачем они используются?
5. Какие переменные называются локальными? Где они объявляются?
6. Как оформляются процедуры, имеющие несколько параметров?
7. Что такое изменяемые параметры? Зачем они используются?
8. Как в заголовке процедуры отличить изменяемый параметр от неизменяемого? Приведите примеры.
9. Какие типы параметров выделяются в школьном алгоритмическом языке? Как они объявляются?

Подготовьте сообщение

а) «Процедуры в языке Си»
б) «Процедуры в языке Python»

Задачи

1. Напишите процедуру, которая выводит на экран запись числа, меньшего, чем 8^{10}, в виде 10 знаков в восьмеричной системе счисления.
2. Напишите процедуру, которая выводит на экран запись числа, меньшего, чем $16^4 = 65\,536$, в виде 4 знаков в шестнадцатеричной системе счисления.
3. Напишите процедуру, которая принимает параметр — натуральное число N и выводит на экран квадрат из звёздочек со стороной N.
4. Напишите процедуру, которая принимает числовой параметр — возраст человека в годах и выводит этот возраст со словом «год», «года» или «лет». Например, `21 год`, `22 года`, `12 лет`.

5. Напишите процедуру, которая выводит переданное ей число прописью. Например: 21 → двадцать один.
6. Напишите процедуру, которая принимает параметр — натуральное число N и выводит первые N чисел Фибоначчи (см. задачу 26 к § 58).
7. Напишите процедуру, которая определяет, верно ли, что переданное ей число — простое. (Используйте изменяемые параметры.)
8. Напишите процедуру, которая выводит на экран в столбик все цифры переданного ей числа, начиная с последней.
9. Напишите процедуру, которая выводит на экран в столбик все цифры переданного ей числа, начиная с первой.
10. Напишите процедуру, которая выводит на экран все делители переданного ей числа (в одну строчку).
11. Напишите процедуру, которая принимает параметр — натуральное число N — и выводит на экран линию из N символов '-'.

*12. Напишите процедуру, которая выводит на экран запись переданного ей числа в римской системе счисления.

§ 60 Функции

Пример функции

С функциями вы уже знакомы, потому что наверняка применяли стандартные функции языка программирования (например, abs, sin, cos). **Функция**, как и процедура, — это **вспомогательный алгоритм**, который может принимать **параметры** (аргументы). Но, в отличие от процедуры, функция всегда *возвращает* **значение-результат**. Результатом может быть число, символ, символьная строка или данные другого типа.

Составим функцию, которая вычисляет сумму цифр числа. Будем использовать следующий алгоритм (предполагается, что число записано в переменной n):

```
сумма:=0
нц пока n<>0
  сумма:=сумма+mod(n,10)
  n:=div(n,10)
кц
```

Чтобы получить последнюю цифру числа (которая добавляется к сумме), нужно взять остаток от деления числа на 10. Затем последняя цифра отсекается, и мы переходим к следующей цифре. Цикл продолжается до тех пор, пока значение n не станет равно нулю.

Как указать в программе, чему равно значение функции? Для этого часто используют такой приём: значение функции записывается в специальную переменную. В школьном алгоритмическом языке имя этой переменной — **знач**, а в Паскале оно совпадает с именем функции (во многих версиях Паскаля можно использовать вместо этого встроенную переменную Result):

```
алг Сумма цифр
нач
 вывод sumDigits(12345)
кон
```

```
алг цел sumDigits(цел n0)
нач
  цел sum=0, n
  n:=n0
  нц пока n<>0
    sum:=sum+mod(n,10)
    n:=div(n,10)
  кц
  знач:=sum
кон
```

```
program Sum;
function sumDigits(n: integer):
                          integer;
var sum: integer;
begin
  sum:=0;
  while n<>0 do begin
    sum:=sum+n mod 10;
    n:=n div 10;
  end;
  sumDigits:=sum
end;
begin
  write(sumDigits(12345))
end.
```

Обратим внимание на особенности записи: тип возвращаемого значения указывается в заголовке функции (в школьном алгоритмическом языке — перед именем функции, в Паскале — в конце заголовка через двоеточие). Так же как и в процедурах, в функциях можно объявлять и использовать локальные переменные. Они входят в «зону видимости» только этой функции, для всех остальных функций и процедур они недоступны.

Функции, созданные в программе таким образом, применяются точно так же, как и стандартные функции. Их можно вызывать везде, где может использоваться выражение того типа, кото-

рый возвращает функция. Приведём несколько примеров вызова функций на школьном алгоритмическом языке:

```
x:=2*sumDigits(n+5)
z:=sumDigits(k) + sumDigits(m)
если div(sumDigits(n),2)=0 то
  вывод 'Сумма цифр чётная', нс
  вывод 'Она равна ', sumDigits(n)
все
```

Логические функции

Достаточно часто применяют специальный тип функций, которые возвращают **логическое значение** (да или нет, True или False). Иначе говоря, такая **логическая функция** отвечает на вопрос «Да или нет?» и возвращает 1 бит информации.

Вернёмся к задаче, которую мы уже рассматривали: вывести на экран все простые числа в диапазоне от 2 до 1000. Алгоритм определения простоты числа оформим в виде функции. При этом его можно легко вызвать из любой точки программы.

Запишем основную программу на псевдокоде:

```
алг Простые числа
нач
  цел n
  нц для n от 2 до 100
    если i – простое то
      вывод n, нс
    все
  кц
кон
```

```
program PrimeNum;
var i: integer;
begin
  for i:=2 to 100 do
    if i – простое then
      writeln(i)
end.
```

Предположим, что у нас уже есть логическая функция isPrime, которая определяет простоту переданного ей числа и возвращает да (в Паскале — True), если число простое, и нет (False) в противном случае. Такую функцию можно использовать вместо выделенного блока алгоритма:

```
если isPrime(i) то
  вывод i, нс
все
```

```
if isPrime(i) then
  writeln(i);
```

Остаётся написать саму функцию isPrime. Будем использовать уже известный алгоритм: если число n в интервале от 2 до $\sqrt{n}$ не имеет ни одного делителя, то оно простое[1]:

```
алг лог isPrime(цел n)
нач
  цел count=0, k
  k:=2
  нц пока k*k<=n и count=0
    если mod(n,k)=0 то
      count:=count+1
    все
    k:=k+1
  кц
  знач:=(count=0)
кон
```

```
function isPrime(n: integer):
                          boolean;
var count, k: integer;
begin
  count:=0;
  k:=2;
  while (k*k<=n) and (count=0)
    do begin
      if n mod k=0 then
        count:=count+1;
      k:=k+1
    end;
  isPrime:=(count=0)
end;
```

Эта функция возвращает логическое значение, на это указывает ключевое слово **лог** (в Паскале — **boolean**). Значение функции определяется как

```
знач:=(count=0)
```

Это оператор присваивания. Слева от знака «:=» записана **логическая переменная**, а справа — **логическое выражение (условие)**. Если это выражение истинно, то в переменную знач записывается значение да, иначе — значение нет.

Логические функции можно использовать так же, как и любые условия: в условных операторах и циклах с условием. Например, такой цикл останавливается на первом введённом составном числе:

```
ввод n
нц пока isPrime(n)
  вывод 'простое число', нс
  ввод n
все
```

```
read(n);
while isPrime(n) do begin
  writeln('простое число');
  read(n)
end;
```

[1] Эту программу можно ещё немного усовершенствовать: после числа 2 имеет смысл проверять только нечётные делители, увеличивая на каждом шаге значение k сразу на 2.

Вопросы и задания

1. Что такое функция? Чем она отличается от процедуры?
2. Как оформляются функции в тексте программы (сравните школьный алгоритмический язык и Паскаль)?
3. Как по тексту программы определить, какое значение возвращает функция? Приведите пример.
4. Какие функции называются логическими? Зачем они нужны?

Подготовьте сообщение

а) «Функции в языке Си»
б) «Функции в языке Python»

Задачи

1. Напишите функцию, которая вычисляет максимальное из трёх чисел.
2. Напишите функцию, которая вычисляет сразу максимальное и минимальное из трёх чисел. (Используйте изменяемые параметры.)
3. Напишите функцию, которая вычисляет количество цифр числа.
4. Напишите функцию, которая вычисляет наибольший общий делитель двух чисел.
5. Напишите функцию, которая вычисляет наименьшее общее кратное двух чисел.
6. Напишите функцию, которая «разворачивает» десятичную запись числа наоборот, например из 123 получается 321, а из 3210 — 0123.
7. Напишите функцию, которая моделирует бросание двух игральных кубиков (на каждом может выпасть от 1 до 6 очков). (Используйте генератор псевдослучайных чисел.)
8. Напишите функцию, которая вычисляет факториал натурального числа N.
9. Напишите функцию, которая вычисляет N-е число Фибоначчи.
10. *Дружественные числа* — это два натуральных числа, таких что сумма всех делителей одного числа (меньших самого этого числа) равна другому числу, и наоборот. Найдите все пары дружественных чисел, каждое из которых меньше 10 000. Используйте функцию, которая вычисляет сумму делителей числа.
11. Напишите программу, которая вводит натуральное число N и находит все числа на отрезке [0, N], сумма цифр которых не меняется при умножении числа на 2, 3, 4, 5, 6, 7, 8 и 9 (например, число 9). Используйте функцию для вычисления суммы цифр числа.

12. Напишите логическую функцию, которая определяет, верно ли, что число N — совершенное, т. е. равно сумме своих делителей, меньших его самого.
13. Простое число называется *гиперпростым*, если любое число, получающееся из него отбрасыванием нескольких цифр, тоже является простым. Например, число 733 — гиперпростое, так как и оно само, и числа 73 и 7 — простые. Напишите логическую функцию, которая определяет, верно ли, что число N — гиперпростое. Используйте уже готовую функцию `isPrime`.

§ 61 Рекурсия

Что такое рекурсия?

Вспомним определение натуральных чисел. Оно состоит из двух частей:

1) 1 — натуральное число;
2) если n — натуральное число, то $n + 1$ — тоже натуральное число.

Вторая часть этого определения в математике называется **индуктивной**: натуральное число определяется через другое натуральное число, и таким образом определяется всё множество натуральных чисел. В программировании этот приём называют рекурсией.

Рекурсия — это способ определения множества объектов через само это множество на основе заданных простых базовых случаев.

Первая часть в определении натуральных чисел — это и есть тот самый базовый случай. Если убрать первую часть из определения, оно будет неполно: вторая часть даёт только метод перехода к следующему значению, но не даёт никакой «зацепки», не отвечает на вопрос «откуда начать».

Похожим образом задаются **числа Фибоначчи**: первые два числа равны 1, а каждое из следующих чисел равно сумме двух предыдущих:

1) $F_1 = F_2 = 1$;
2) $F_n = F_{n-1} + F_{n-2}$ для $n > 2$.

Популярные примеры рекурсивных объектов — **фракталы**. Так в математике называют геометрические фигуры, обладающие **самоподобием**. Это значит, что они составлены из фигур меньшего размера, каждая из которых подобна целой фигуре. На рисунке 8.2 показан **треугольник Серпинского** — один из первых фракталов, который предложил в 1915 г. польский математик В. Серпинский.

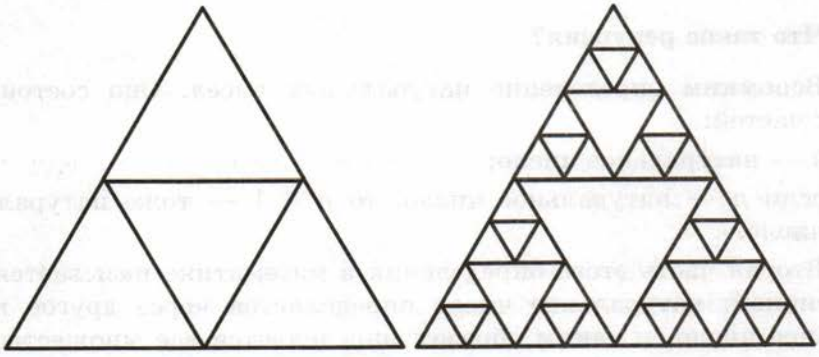


Рис. 8.2

Равносторонний треугольник делится на 4 равных треугольника меньшего размера (см. рис. 8.2, слева), затем каждый из полученных треугольников, кроме центрального, снова делится на 4 ещё более мелких треугольника и т. д. На рисунке 8.2 справа показан треугольник Серпинского с тремя уровнями деления.

Ханойские башни

Согласно легенде, конец света наступит тогда, когда монахи Великого храма города Бенарас смогут переложить 64 диска разного диаметра с одного стержня на другой. Вначале все диски на-

низаны на первый стержень. За один раз можно перекладывать только один диск, причём разрешается класть только меньший диск на больший. Есть также и третий стержень, который можно использовать в качестве вспомогательного (рис. 8.3).

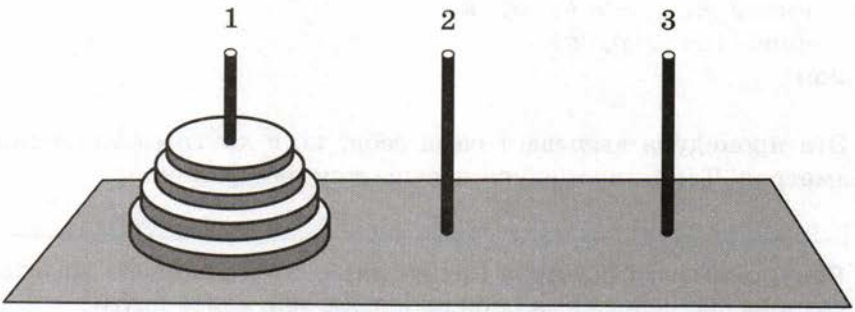


Рис. 8.3

Решить задачу для 2, 3 и даже 4 дисков довольно просто. Проблема в том, чтобы составить алгоритм для любого числа дисков.

Пусть нужно перенести n дисков со стержня 1 на стержень 3. Представим себе, что мы как-то смогли переместить $n - 1$ дисков на вспомогательный стержень 2. Тогда остаётся перенести самый большой диск на стержень 3, а затем $n - 1$ меньших дисков со вспомогательного стержня 2 на стержень 3, и задача будет решена. Получается такой псевдокод:

```
перенести (n-1, 1, 2)
1 -> 3
перенести (n-1, 2, 3)
```

Процедура `перенести` (которую нам предстоит написать) принимает три параметра: число дисков, начальный и конечный стержни. Таким образом, мы свели задачу переноса n дисков к двум задача переноса $n - 1$ дисков и одному элементарному действию — переносу 1 диска. Заметим, что при известных номерах начального и конечного стержней легко рассчитать номер вспомогательного, так как сумма номеров равна $1 + 2 + 3 = 6$. Получается такая (пока не совсем верная) процедура:

```
алг Hanoi(цел n, k, m)
нач
  цел p
  p:=6-k-m
  Hanoi(n-1, k, p)
  вывод k, ' -> ', m, нс
  Hanoi(n-1, p, m)
кон
```

Эта процедура вызывает сама себя, но с другими значениями параметров. Такая процедура называется рекурсивной.

Рекурсивная процедура (функция) — это процедура (функция), которая вызывает сама себя напрямую или через другие процедуры и функции.

Основная программа для решения задачи, например, с четырьмя дисками будет содержать всего одну строку (перенести 4 диска со стержня 1 на стержень 3):

```
Hanoi(4, 1, 3)
```

Если вы попробуете запустить эту программу, она зациклится, т. е. будет работать бесконечно долго. В чём же дело? Вспомните, что определение рекурсивного объекта состоит из двух частей. В первой части определяются базовые объекты (например, первое натуральное число), именно эта часть «отвечает» за остановку рекурсии в программе. Пока мы не определили такое условие останова, процедура будет бесконечно вызывать саму себя (это можно проверить, выполняя программу по шагам). Когда же остановить рекурсию?

Очевидно, что если нужно переложить 0 дисков, задача уже решена, ничего перекладывать не надо. Это и есть условие окончания рекурсии: если значение параметра `n`, переданное процедуре, равно 0, нужно выйти из процедуры без каких-либо действий. Для этого в школьном алгоритмическом языке используется оператор `выход` (в Паскале — оператор `exit`). Правильная версия процедуры выглядит так:

```
алг Hanoi(цел n, k, m)
нач
  если n=0 то выход все
  цел p
  p:=6-k-m
  Hanoi(n-1, k, p)
  вывод k, ' -> ', m, нс
  Hanoi(n-1, p, m)
кон
```

```
procedure Hanoi(n, k, m: integer);
var p: integer;
begin
  if n=0 then exit;
  p:=6-k-m;
  Hanoi(n-1, k, p);
  writeln(k, ' -> ', m);
  Hanoi(n-1, p, m)
end;
```

Чтобы переложить N дисков, нужно выполнить $2^N - 1$ перекладываний. Для $N = 64$ это число равно 18 446 744 073 709 551 615. Если бы монахи, работая день и ночь, каждую секунду перемещали один диск, им бы потребовалось 580 миллиардов лет.

Примеры

Пример 1. Составим процедуру, которая переводит натуральное число в двоичную систему. Мы уже занимались вариантом этой задачи, где требовалось вывести 8-битную запись числа из диапазона 0..255, сохранив лидирующие нули. Теперь усложним задачу: лидирующие нули выводить не нужно, а натуральное число может быть любым (в пределах допустимого диапазона для выбранного типа данных).

Стандартный алгоритм перевода числа в двоичную систему можно записать, например, так:

```
нц пока n<>0
  вывод mod(n,2)
  n:=div(n,2)
кц
```

Проблема в том, что двоичное число выводится «задом наперёд», т. е. первым будет выведен последний разряд. Как «перевернуть» число?

Есть разные способы решения этой задачи, которые сводятся к тому, чтобы запоминать остатки от деления (например, в символьной строке) и затем, когда результат будет полностью получен, вывести его на экран.

Однако можно применить красивый подход, использующий рекурсию. Идея такова: чтобы вывести двоичную запись числа `n`, нужно сначала вывести двоичную запись числа `div(n, 2)`, а затем последнюю цифру — `mod(n, 2)`. Если число-параметр равно

нулю, нужно выйти из процедуры. Такой алгоритм очень просто программируется:

```
алг printBin(цел n)
нач
  если n=0 то выход все
  printBin(div(n, 2))
  вывод mod(n,2)
кон
```

```
procedure printBin(n: integer);
begin
  if n=0 then exit;
  printBin(n div 2);
  write(n mod 2)
end;
```

Конечно, решить эту задачу можно было и с помощью цикла. Поэтому можно сделать важный вывод: *рекурсия заменяет цикл*. При этом программа, как правило, становится более понятной.

Пример 2. Составим функцию, которая вычисляет сумму цифр числа. Будем рассуждать так: сумма цифр числа n равна значению последней цифры плюс сумма цифр числа div(n, 10). Сумма цифр однозначного числа равна самому этому числу, это условие окончания рекурсии. Получаем следующую функцию:

```
алг цел sumDig(цел n)
нач
  знач:=mod(n,10)
  если n>=10 то
    знач:=знач+sumDig(div
                    (n,10))
 все
кон
```

```
function sumDig
                (n: integer):
                  integer;
var sum: integer;
begin
  sum:=n mod 10;
  if n>=10 then
    sum:=sum+sumDig
                  (n div 10);
  sumDig:= sum
end;
```

Пример 3. Алгоритм Евклида, один из древнейших известных алгоритмов, предназначен для поиска наибольшего общего делителя (НОД) двух натуральных чисел. Формулируется он так.

Алгоритм Евклида. Чтобы найти НОД двух натуральных чисел, нужно вычитать из большего числа меньшее до тех пор, пока меньшее не станет равно нулю. Тогда второе число и есть НОД исходных чисел.

Этот алгоритм может быть сформулирован в рекурсивном виде. Во-первых, в нём для перехода к следующему шагу используется равенство НОД(a, b) = НОД($a - b$, b) при $a \geq b$. Кроме того, задано условие останова: если одно из чисел равно нулю, то НОД совпадает со вторым числом. Поэтому можно написать такую рекурсивную функцию:

```
алг цел NOD(цел a, b)
нач
  если a=0 или b=0 то
    знач:=a+b
    выход
  все
  если a>b то
        знач:=NOD(a-b, b)
  иначе знач:=NOD(a, b-a)
  все
кон
```

```
function NOD(a, b: integer):
                        integer;
begin
  if (a=0) or (b=0) then begin
    NOD:=a+b;
    exit;
  end
  if a>b then
       NOD:=NOD(a-b, b)
  else NOD:=NOD(a, b-a)
end;
```

Заметим, что при равенстве одного из чисел нулю второе число совпадает с суммой двух чисел, поэтому в качестве результата функции принимается $a + b$.

Существует и более быстрый вариант алгоритма Евклида (модифицированный алгоритм), в котором большее число заменяется на остаток от деления большего числа на меньшее.

Как работает рекурсия

Рассмотрим вычисление *факториала* $N!$: так называют произведение всех натуральных чисел от 1 до заданного числа N: $N! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot N$. Факториал может быть также введён с помощью **рекуррентной формулы**, которая связывает факториал данного числа с факториалом предыдущего[1]:

$$N! = \begin{cases} 1, & \text{если } N = 1; \\ N \cdot (N-1)!, & \text{если } N \geq 2. \end{cases}$$

Здесь первая часть описывает базовый случай (условие окончания рекурсии), а вторая — переход к следующему шагу. Запи-

[1] В математике принято, что факториал нуля равен 1. Поэтому равенство 0! = 1 можно тоже принять за базовый случай.

шем соответствующую функцию на школьном алгоритмическом языке, добавив в начале и в конце операторы вывода:

```
алг цел Fact(цел N)                    -> N=3
нач                                     -> N=2
  вывод '-> N=', N, нс                   -> N=1
  если N<=1 то                            <- N=1
    знач:=1                              <- N=2
  иначе знач:=N*Fact(N-1)               <- N=3
  все
  вывод '<- N=', N, нс
кон
```

Справа от программы показан протокол её работы при вызове `Fact(3)` (для наглядности сделаны отступы, показывающие вложенность вызовов). Из протокола видно, что вызов `Fact(2)` происходит раньше, чем заканчивается вызов `Fact(3)`. Это значит, что компьютеру нужно где-то (без помощи программиста) запомнить состояние программы (в том числе значения всех локальных переменных) и адрес, по которому нужно вернуться после завершения вызова `Fact(2)`. Для этой цели используется стек.

Стек (англ. *stack* — кипа, стопка) — особая область памяти, в которой хранятся локальные переменные и адреса возврата из процедур и функций.

Один из регистров процессора называется **указателем стека** (англ. *stack pointer*, **SP**) — в нём записан адрес последней занятой ячейки стека. При вызове процедуры в стек помещаются значения всех её параметров, адрес возврата и выделяется место под локальные переменные.

На рисунке 8.4, *а* показано начальное состояние стека, серым цветом выделены занятые ячейки. Когда функция `Fact` вызывается из основной программы с параметром 3, в стек записывается значение параметра, затем — адрес возврата A, и выделяется место для локальной переменной `знач`, в которую будет записан результат[1] (рис. 8.4, *б*). При втором и третьем вложенных вызовах в

[1] Результат функции, как правило, передаётся в вызывающую программу через регистры, но во время работы функции хранится в стеке.

стек добавляются аналогичные блоки данных (рис. 8.4, *в* и *г*). Заметьте, что в нашем случае адрес возврата во всех трёх блоках будет один и тот же.

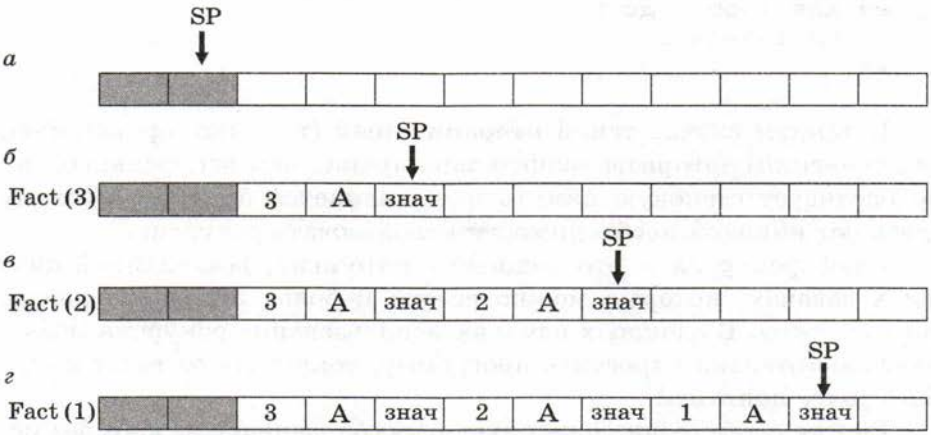


Рис. 8.4

Когда выполняется возврат из процедуры, состояние стека изменяется в обратную сторону: *г* — *в* — *б* — *а*.

Что же следует из этой схемы? Во-первых, с каждым новым вызовом расходуется дополнительная стековая память. Если вложенных вызовов будет очень много (или если процедура создаёт много локальных переменных), эта память закончится, и программа завершится аварийно.

Во-вторых, при каждом вызове процедуры некоторое время затрачивается на выполнение служебных операций (занесение данных в стек и т. п.), поэтому, как правило, рекурсивные программы выполняются несколько дольше, чем аналогичные нерекурсивные.

Всегда ли можно написать нерекурсивную программу? Оказывается, всегда. *Доказано*, что любой рекурсивный алгоритм может быть записан без использования рекурсии. Хотя:

- часто при этом программа усложняется и становится менее понятной;
- нерекурсивная версия нередко требует дополнительного расхода памяти.

Например, для вычисления факториала можно использовать обычный цикл:

```
знач:=1
нц для i от 1 до N
  знач:=знач*i
кц
```

В данном случае такой **итерационный** (т. е. повторяющийся, циклический) **алгоритм** значительно лучше, чем рекурсивный: он не расходует стековую память и выполняется быстрее. Поэтому здесь нет никакой необходимости использовать рекурсию.

Итак, рекурсия — это мощный инструмент, заменяющий циклы в задачах, которые можно свести к более простым задачам того же типа. В сложных случаях использование рекурсии позволяет значительно упростить программу, сократить её текст и сделать более понятной.

Вместе с тем, если существует простое решение задачи без использования рекурсии, лучше применить именно его. Нужно стараться обходиться без рекурсии, если вложенность вызовов получается очень большой или подпрограмма использует много локальных данных.

Вопросы и задания

1. Что такое рекурсия? Приведите примеры.
2. Почему любое рекурсивное определение состоит из двух частей?
3. Что такое рекурсивная процедура (функция)?
4. Расскажите о задаче «Ханойские башни». Попытайтесь придумать алгоритм её решения, не использующий рекурсию.
5. Процедура *А* вызывает процедуру *Б*, а процедура *Б* — процедуру *А* и саму себя. Какую из этих процедур можно назвать рекурсивной?
6. В каком случае рекурсия никогда не остановится? Докажите, что в рассмотренных в параграфе задачах этого не случится.
7. Что такое стек? Как он используется при выполнении программ?
8. Почему при использовании рекурсии может случиться переполнение стека?
9. Назовите достоинства и недостатки рекурсии. Когда её следует использовать, а когда — нет?

Подготовьте сообщение

а) «Фракталы»
б) «Числа Фибоначчи»
в) «Рекурсия вокруг нас»
г) «Рекурсия в программировании: за и против»
д) «Рекурсия в произведениях искусства»

Задачи

1. Придумайте свою рекурсивную фигуру и опишите её.
*2. Используя графические возможности языка программирования, который вы изучаете, постройте на экране треугольник Серпинского и другие фракталы.
3. Напишите рекурсивную процедуру для перевода числа в двоичную систему, которая правильно работала бы для нуля (выводила бы 0).
*4. Дано натуральное число N. Требуется получить и вывести на экран все возможные *различные* способы представления этого числа в виде суммы натуральных чисел (т. е. 1 + 2 и 2 + 1 — это один и тот же способ разложения числа 3). Решите задачу с помощью рекурсивной процедуры.
5. Напишите рекурсивную процедуру для перевода числа из двоичной системы счисления в десятичную.
6. Напишите рекурсивную и нерекурсивную функции, вычисляющие НОД двух натуральных чисел с помощью модифицированного алгоритма Евклида. Какой вариант вы предпочтёте?

§ 62 Массивы

Что такое массив?

Основное предназначение современных компьютеров — обработка большого количества данных. При этом надо как-то обращаться к каждой из тысяч (или даже миллионов) ячеек с данными. Очень сложно дать каждой ячейке собственное имя и при этом не запутаться. Из этой ситуации выходят так: дают имя не ячейке, а группе ячеек, в которой каждая ячейка имеет собственный номер. Такая область памяти называется массивом.

Массив — это группа переменных одного типа, расположенных в памяти рядом (в соседних ячейках) и имеющих общее имя. Каждая ячейка в массиве имеет уникальный номер.

Для работы с массивами нужно в первую очередь научиться:

- выделять память нужного размера под массив;
- записывать данные в ячейку массива;
- читать данные из ячейки массива.

Чтобы использовать массив, надо его объявить: определить тип массива (тип входящих в него элементов), выделить место в памяти и присвоить имя. Имена массивов строятся по тем же правилам, что и имена переменных.

В школьном алгоритмическом языке массивы называются **таблицами**. При их объявлении к названию типа данных добавляется слово **таб**:

```
целтаб A[1:5]
вещтаб V[0:5]
логтаб L[-5:5]
симтаб S[65:90]
```

В квадратных скобках через двоеточие записываются границы **индексов** — номеров ячеек массива. В приведённом примере массив A — это массив целых значений, ячейки имеют номера от 1 до 5. Массив вещественных значений V содержит 6 элементов с номерами от нуля[1] до 5. В логическом массиве L ячейки нумеруются от -5 до 5, а в символьном массиве S — от 65 до 90. В школьном алгоритмическом языке объявлять массивы (как и переменные) можно в любом месте программы.

В языке Паскаль массивы объявляются в блоке объявления переменных (выше ключевого слова **begin**) и начинаются ключевым словом **array** (в переводе с англ. — массив). Объявления, аналогичные приведённым на школьном алгоритмическом языке, выглядят так:

```
var A: array[1..5] of integer;
    V: array[0..5] of real;
    L: array[-5..5] of boolean;
    S: array[65..90] of char;
```

Минимальный и максимальный индексы разделяются двумя точками.

Для того чтобы *обратиться к элементу массива*, нужно записать имя массива и в квадратных скобках — индекс нужного

[1] Нумерация с нуля часто используется в языках программирования, например в языке Си и родственных ему языках.

элемента, например *A*[3]. Индексом может быть не только число, но значение целой переменной или арифметического выражения целого типа.

В следующем примере массив заполняется квадратами первых натуральных чисел:

```
алг Массив
нач
  цел i, N=10
  целтаб A[1:N]
  нц для i от 1 до N
    A[i]:=i*i
  кц
кон
```

```
program qq;
const N=10;
var A: array[1..N] of integer;
    i: integer;
begin
  for i:=1 to N do
    A[i]:=i*i;
end.
```

В школьном алгоритмическом языке можно при объявлении массива указывать вычисляемые границы индексов, зависящие от переменных. В данном случае массив *A* состоит из 10 элементов, потому что к моменту его объявления в переменной *N* было число 10.

В Паскале при объявлении границ индексов массивов можно использовать **константы** — постоянные величины, имеющие имя. В приведённом примере с помощью ключевого слова **const** объявлена константа *N*, равная 10. Константы обычно вводятся выше блока объявления переменных. Использование констант очень удобно, потому что при изменении размера массива в программе нужно поменять только одно число — значение этой константы.

Далее во всех примерах мы будем считать, что в программе объявлен целочисленный массив *A* с индексами от 1 до *N*, а также целочисленная переменная *i*, которая будет обозначать индекс элемента массива. Чтобы ввести такой массив или вывести его на экран, нужно использовать цикл, т. е. ввод и вывод массива выполняется поэлементно:

```
нц для i от 1 до N
  вывод 'A[',i,']='
  ввод A[i]
кц
...
нц для i от 1 до N
  вывод A[i], ' '
кц
```

```
for i:=1 to N do begin
  write('A[',i,']=');
  read(A[i])
end;
...
for i:=1 to N do
  write(A[i],' ');
```

В этом примере перед вводом очередного элемента массива на экран выводится подсказка. Например, при вводе 3-го элемента будет выведено: `A[3]=`. После вывода каждого элемента ставится пробел, иначе все значения сольются в одну строку.

В учебных примерах массивы часто заполняют случайными числами:

```
нц для i от 1 до N
  A[i]:=irand(20,100)
  вывод A[i], ' '
кц
```

```
for i:=1 to N do begin
  A[i]:=20+random(81);
  write(A[i], ' ')
end;
```

Перебор элементов

Перебор элементов состоит в том, что мы в цикле просматриваем все элементы массива и, если нужно, выполняем с каждым из них некоторую операцию. Для этого удобнее всего использовать цикл с переменной, которая изменяется от минимального до максимального индекса. Для массива, элементы которого имеют индексы от 1 до N, цикл выглядит так:

```
нц для i от 1 до N
...
кц
```

```
for i:=1 to N do begin
...
end;
```

Здесь вместо многоточия можно добавлять операторы, работающие с элементом $A[i]$.

Во многих задачах нужно найти в массиве все элементы, удовлетворяющие заданному условию, и как-то их обработать. Простейшая из таких задач — подсчёт нужных элементов. Для решения этой задачи нужно ввести переменную-счётчик, начальное значение которой равно нулю. Далее в цикле (от 1 до N) просматриваем все элементы массива. Если для очередного элемента выполняется заданное условие, то увеличиваем счётчик на 1. На псевдокоде этот алгоритм выглядит так:

```
счётчик:=0
нц для i от 1 до N
  если условие выполняется для A[i] то
    счётчик:=счётчик+1
  все
кц
```

Предположим, что в массиве A записаны данные о росте игроков баскетбольной команды (в сантиметрах). Найдем количество

игроков, рост которых больше 180 см, но меньше 190 см. В следующей программе используется переменная-счётчик *count*:

```
цел count=0
нц для i от 1 до N
  если 180<A[i]
        и A[i]<190 то
    count:=count+1
  все
кц
```

```
count:=0;
for i:=1 to N do
  if (180<A[i])
     and (A[i]<190)
    then count:=count+1;
```

Теперь усложним задачу: требуется найти средний рост этих игроков. Для этого в отдельной переменной будем складывать все нужные значения, а после завершения цикла разделим эту сумму на количество найденных элементов. Начальное значение переменной *sum*, в которой накапливается сумма, тоже должно быть равно нулю.

```
цел count=0, sum=0
нц для i от 1 до N
  если 180<A[i]
        и A[i]<190 то
    count:=count+1
    sum:=sum+A[i]
  все
кц
вывод sum/count
```

```
count:=0; sum:=0;
for i:=1 to N do
  if (180<A[i])
     and (A[i]<190)
  then begin
    count:=count+1;
    sum:=sum+A[i]
  end;
write(sum/count);
```

Вопросы и задания

1. Что такое массив? Зачем нужны массивы?
2. Зачем нужно объявлять массивы?
3. Как объявляются массивы в школьном алгоритмическом языке и в Паскале?
4. Как вы думаете, почему элементы массива расположены в памяти рядом?
5. Как обращаются к элементу массива?
6. Могут ли индексы элементов массива начинаться с 0? С –5?
7. Почему размер массива лучше вводить как константу, а не как число?
8. Как ввести массив и вывести его на экран?
9. Как заполнить массив случайными числами в диапазоне от 100 до 200?

Подготовьте сообщение

а) «Массивы в языке Си»

б) «Списки и словари в языке Python»

Задачи

1. Заполните массив элементами арифметической прогрессии. Её первый элемент и разность нужно ввести с клавиатуры.
2. Заполните массив степенями числа 2 (от 2^1 до 2^N).
3. Заполните массив первыми числами Фибоначчи.

*4. Заполните массив из N элементов случайными целыми числами в диапазоне $1..N$ так, чтобы в массив обязательно вошли все числа от 1 до N (постройте случайную перестановку).

*5. Постройте случайную перестановку чисел от 1 до N так, чтобы первое число обязательно было равно 5 ($N \geq 5$).

6. Заполните массив случайными числами в диапазоне 20..100 и подсчитайте отдельно число элементов с чётными и нечётными значениями.
7. Заполните массив случайными числами в диапазоне 1000..2000 и подсчитайте число элементов, у которых вторая с конца цифра — чётная.
8. Заполните массив случайными числами в диапазоне 0..100 и подсчитайте отдельно среднее значение всех элементов, меньших 50, и среднее значение всех элементов, которые больше или равны 50.

§ 63
Алгоритмы обработки массивов

Поиск в массиве

Требуется найти в массиве элемент, значение которого равно значению переменной X, или сообщить, что такого элемента в массиве нет. Алгоритм решения сводится к просмотру всех элементов массива с первого до последнего. Как только будет найден элемент, равный X, нужно выйти из цикла и вывести результат. Напрашивается такой алгоритм:

```
i:=1
нц пока A[i]<>X
  i:=i+1
кц
вывод 'A[', i, ']=', X
```

Он хорошо работает, если нужный элемент в массиве есть, однако приведёт к ошибке, если такого элемента нет, — получится зацикливание и выход за границы массива. Поэтому в условие нужно добавить ещё одно ограничение: `i<=N`. Если после оконча-

ния цикла это условие нарушено, значит, поиск был неудачным — элемента нет:

```
i:=1
нц пока i<=N и A[i]<>X
  i:=i+1
кц
если i<=N то
      вывод 'A[',i,']=',X
иначе вывод 'Не нашли!'
все
```

```
i:=1;
while (i<=N) and (A[i]<>X) do
  i:=i+1;
if i<=N then
     write('A[', i, ']=', X)
else write('Не нашли!');
```

Отметим одну тонкость. В сложном условии `i<=N и A[i]<>X` первой должно проверяться именно отношение `i<=N`. Если первая часть условия, образованного с помощью операции «И», ложно, то вторая часть, как правило[1], не вычисляется — уже понятно, что всё условие ложно. Дело в том, что если `i>N`, проверка условия `A[i]<>X` приводит к *выходу за границы массива*, и программа может завершиться аварийно.

Возможен ещё один поход к решению этой задачи: используя цикл с переменной, перебирать все элементы массива и досрочно завершить цикл, если найдено требуемое значение.

```
nX:= 0
нц для i от 1 до N
  если A[i]=X то
    nX:= i
    выход
  все
кц
если nX>0 то
       вывод 'A[', nX, ']=', X
  иначе вывод 'Не нашли!'
все
```

```
nX:= 0;
for i:=1 to N do
  if A[i]=X then begin
    nX:= i;
    break
  end;
if nX>0 then
     write('A[', nX, ']=', X)
else write('Не нашли!');
```

Здесь для выхода из цикла используется оператор `выход` (в Паскале — `break`), номер найденного элемента сохраняется в переменной *nX*. Если её значение осталось равным нулю (не изменилось в ходе выполнения цикла), то в массиве нет элемента, равного *X*.

---

[1] Это зависит от транслятора и его настроек. В школьном алгоритмическом языке и во многих других современных языках (например, в C, C++, Python, Javascript, PHP) такое поведение гарантировано стандартом, а в Паскале — нет.

Максимальный элемент

Найдём в массиве максимальный элемент. Для его хранения выделим целочисленную переменную M. Будем в цикле просматривать все элементы массива один за другим. Если очередной элемент массива больше, чем максимальный из предыдущих (находящийся в переменной M), запомним новое значение максимального элемента в M.

Остается решить, каково должно быть начальное значение M. Во-первых, можно записать туда значение, заведомо меньшее, чем любой из элементов массива. Например, если в массиве записаны натуральные числа, можно записать в M ноль. Если содержимое массива неизвестно, можно сразу записать в M значение $A[1]$, а цикл перебора начать со второго элемента:

```
M:=A[1]
нц для i от 2 до N
  если A[i]>M то
    M:=A[i]
  все
кц
вывод M
```

```
M:=A[1];
for i:=2 to N do
  if A[i]>M then
    M:=A[i];
write(M);
```

Теперь предположим, что нужно найти не только значение, но и номер максимального элемента. Казалось бы, нужно ввести ещё одну переменную *nMax* для хранения номера, сначала записать в неё 1 (считаем первый элемент максимальным) и затем, когда найдём новый максимальный элемент, запоминать его номер в переменной *nMax*:

```
M:=A[1]; nMax:=1
нц для i от 2 до N
  если A[i]>M то
    M:=A[i]
    nMax:=i
  все
кц
вывод 'A[', nMax, ']=', M
```

```
M:=A[1]; nMax:=1;
for i:=2 to N do
  if A[i]>M then begin
    M:=A[i];
    nMax:=i;
  end;
write('A[', nMax, ']=' ,M);
```

Однако это не самый лучший вариант. Дело в том, что по номеру элемента можно всегда определить его значение. Поэтому достаточно хранить только номер максимального элемента. Если этот номер равен *nMax*, то значение максимального элемента равно $A[nMax]$:

```
nMax:=1
нц для i от 2 до N
  если A[i]>A[nMax] то
    nMax:=i
  все
кц
вывод 'A[', nMax, ']=',
               A[nMax]
```

```
nMax:=1;
for i:=2 to N do
  if A[i]>A[nMax] then
    nMax:=i;
write('A[', nMax, ']=',
                A[nMax]);
```

Реверс массива

Реверс массива — это перестановка его элементов в обратном порядке: первый элемент становится последним, а последний — первым (рис. 8.5).

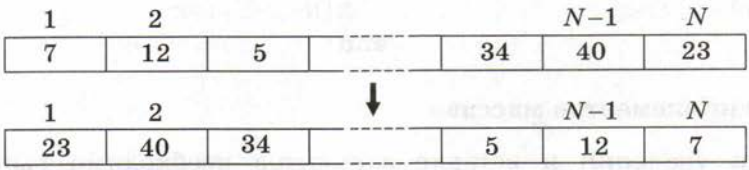


Рис. 8.5

Из рисунка 8.5 следует, что 1-й элемент меняется местами с N-м, второй — с $(N - 1)$-м и т. д. Сумма индексов элементов, участвующих в обмене, для всех пар равна $N + 1$, поэтому элемент с номером i должен меняться местами с $(N + 1 - i)$-м элементом. Кажется, что можно написать такой цикл:

```
нц для i от 1 до N
  поменять местами A[i] и A[N+1-i]
кц
```

Однако это неверно. Посмотрим, что получится для массива из четырёх элементов (рис. 8.6).

	1	2	3	4
	7	12	40	23
$A[1] \leftrightarrow A[4]$	23	12	40	7
$A[2] \leftrightarrow A[3]$	23	40	12	7
$A[3] \leftrightarrow A[2]$	23	12	40	7
$A[4] \leftrightarrow A[1]$	7	12	40	23

Рис. 8.6

Как видите, массив вернулся в исходное состояние: реверс выполнен дважды. Поэтому нужно остановить цикл на середине массива:

```
нц для i от 1 до div(N,2)
  поменять местами A[i] и A[N+1-i]
кц
```

Для обмена используется вспомогательная целая переменная c:

```
нц для i от 1 до div(N,2)
  c:=A[i]
  A[i]:=A[N+1-i]
  A[N+1-i]:=c
кц
```

```
for i:=1 to N div 2 do begin
  c:=A[i];
  A[i]:=A[N+1-i];
  A[N+1-i]:=c
end;
```

Сдвиг элементов массива

При удалении и вставке элементов необходимо выполнять **сдвиг** части или всех элементов массива в ту или другую сторону. Массив часто рисуют в виде таблицы, где первый элемент расположен слева. Поэтому сдвиг влево — это перемещение всех элементов на одну ячейку, при котором $A[2]$ переходит на место $A[1]$, $A[3]$ — на место $A[2]$ и т. д. (рис. 8.7).

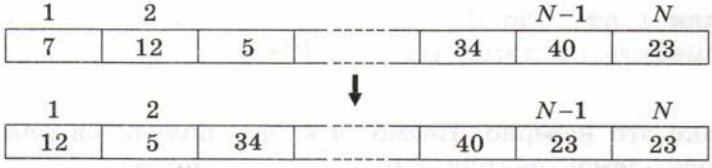


Рис. 8.7

Последний элемент остаётся на своём месте, поскольку новое значение для него взять неоткуда — массив кончился. Алгоритм выглядит так:

```
нц для i от 1 до N-1
  A[i]:=A[i+1]
кц
```

```
for i:=1 to N-1 do
  A[i]:=A[i+1];
```

Обратите внимание, что цикл заканчивается при $i = N - 1$ (а не N), чтобы не было выхода за границы массива, т. е. обращения к несуществующему элементу $A[N+1]$.

При таком сдвиге первый элемент пропадает, а последний дублируется. Можно старое значение первого элемента записать на место последнего. Такой сдвиг называется **циклическим** (см. § 28). Предварительно (до начала цикла) первый элемент нужно запомнить во вспомогательной переменной, а после завершения цикла записать его в последнюю ячейку массива:

```
c:=A[1]
нц для i от 1 до N-1
  A[i]:=A[i+1]
кц
A[N]:=c
```

```
c:=A[1];
for i:=1 to N-1 do
  A[i]:=A[i+1];
A[N]:=c;
```

Отбор нужных элементов

Требуется отобрать все элементы массива A, удовлетворяющие некоторому условию, в массив B. «Очевидное» решение:

```
нц для i от 1 до N
  если условие выполняется для A[i] то
    B[i]:=A[i]
  все
кц
```

На самом деле это решение неудачное, потому что нужные элементы в массиве B оказываются расположенными вразброс, на тех местах, где они стояли в массиве A. Поэтому работать с таким массивом B очень неудобно. На рисунке 8.8 изображён случай, когда отбираются элементы с чётными значениями.

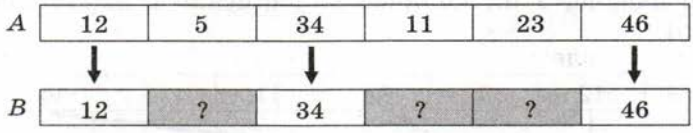


Рис. 8.8

Будет удобно, если все отобранные элементы будут стоять в начале массива B (рис. 8.9).

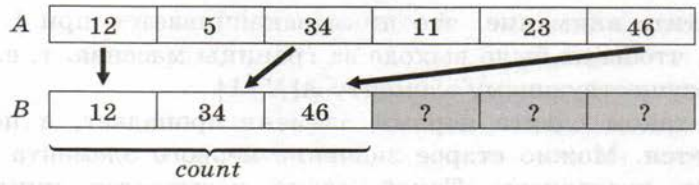


Рис. 8.9

Для этого вводят переменную-счётчик, в которой считают количество найденных элементов. Сначала её значение равно нулю. Как только очередной подходящий элемент найден, счётчик увеличивается на 1, и теперь значение счётчика — это номер первой свободной ячейки массива *B*, в неё и записывается найденное значение. Программа, отбирающая все элементы с чётными значениями, выглядит так:

```
count:=0
нц для i от 1 до N
  если mod(A[i],2)=0 то
    count:=count+1
    B[count]:=A[i]
  все
кц
```

```
count:=0
for i:=1 to N do
  if A[i] mod 2=0 then begin
    count:=count+1;
    B[count]:=A[i]
  end;
```

Нужно помнить, что только первые `count` элементов массива `B` рабочие, остальные содержат неизвестные данные. Например, вот так можно вывести найденные элементы на экран:

```
нц для i от 1 до count
  вывод B[i], ' '
кц
```

```
for i:=1 to count do
  write(B[i], ' ');
```

Если вместо массива *B* использовать тот же массив *A*, где находятся исходные числа, все «нужные» элементы будут сгруппированы в начале, а их количество записано в переменной `count` (рис. 8.10).

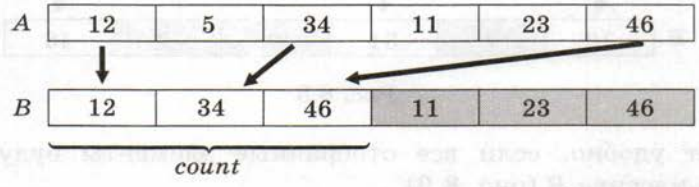


Рис. 8.10

Вопросы и задания

1. Почему при поиске индекса максимального элемента необязательно хранить само значение максимального элемента?
2. Что такое реверс массива?
3. Как вы думаете, какую ошибку чаще всего делают начинающие программисты, программируя реверс массива?
4. Как вы думаете, какие проблемы (и ошибки) могут возникнуть при циклическом сдвиге массива *вправо*?
5. Что произойдёт с массивом при выполнении следующего фрагмента программы?

```
нц для i от 1 до N-1
  A[i+1]:=A[i]
кц
```

```
for i:=1 to N-1 do
  A[i+1]:=A[i];
```

6. Как при использовании приведённого алгоритма поиска определить, что элемент не найден?
7. Что такое выход за границы массива? Почему он может быть опасен?
8. Опишите «очевидный» алгоритм отбора части элементов одного массива в другой массив. Почему его не используют?

Подготовьте сообщение

а) «Выход за границы массива»
б) «Алгоритмы работы со списками на языке Python»

Задачи

1. Напишите программу, которая находит максимальный и минимальный из элементов массива с чётными положительными значениями. Если в массиве нет элементов с чётными положительными значениями, нужно вывести сообщение об этом.
2. Введите массив с клавиатуры и найдите (за один проход) количество элементов, имеющих максимальное значение.
3. Найдите за один проход по массиву три его различных элемента, которые меньше всех остальных («три минимума»).
*4. Заполните массив случайными числами в диапазоне 10..12 и найдите длину самой длинной последовательности стоящих рядом элементов с одинаковыми значениями.
5. Заполните массив случайными числами в диапазоне 0..4 и выведите на экран номера всех элементов, значение которых равно X (X вводится с клавиатуры).

6. Заполните массив случайными числами и переставьте соседние элементы, поменяв 1-й элемент со 2-м, 3-й — с 4-м и т. д.
7. Заполните массив с чётным количеством элементов случайными числами и выполните реверс отдельно для первой и второй половин массива.
8. Заполните массив случайными числами и выполните реверс для части массива между элементами с индексами K и M (включая эти элементы).
9. Напишите программу для выполнения циклического сдвига массива вправо на 4 элемента.
10. Найдите в массиве все простые числа и скопируйте их в новый массив.

*11. Найдите в массиве все числа Фибоначчи и скопируйте их в новый массив.

§ 64 Сортировка

Сортировка — это расстановка элементов массива в заданном порядке.

Порядок сортировки может быть любым, для чисел обычно рассматривают сортировку по возрастанию (или убыванию) значений.

Возникает естественный вопрос: зачем сортировать данные? На него легко ответить, вспомнив, например, работу со словарями: сортировка слов по алфавиту облегчает поиск нужной информации.

Программисты изобрели множество способов сортировки. В целом их можно разделить на две группы: 1) простые, но медленно работающие (на больших массивах) и 2) сложные, но быстрые. Мы изучим два классических метода из первой группы и один метод из второй — знаменитую «быструю сортировку», предложенную *Ч. Хоаром*.

Далее мы будем рассматривать сортировку массива по возрастанию (или убыванию) значений. Для массивов, в которых есть одинаковые элементы, используются понятия «сортировка по неубыванию» и «сортировка по возрастанию». Сортировка по убыванию выполняется аналогично.

Метод пузырька (сортировка обменами)

Название этого метода произошло от известного физического явления — пузырёк воздуха в воде поднимается вверх. Если говорить о сортировке массива, сначала поднимается «наверх» (к началу массива) самый «лёгкий» элемент (элемент с минимальными значениями), затем следующий и т. д.

Сначала сравниваем последний элемент с предпоследним. Если они стоят неправильно (меньший элемент «ниже»), то меняем их местами. Далее так же рассматриваем следующую пару элементов и т. д. (рис. 8.11).

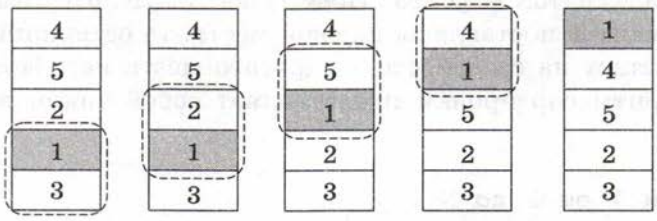


Рис. 8.11

После обработки пары ($A[1]$, $A[2]$) минимальный элемент стоит на месте $A[1]$. Это значит, что на следующих этапах его можно не рассматривать. Первый цикл, устанавливающий на свое место первый (минимальный) элемент, можно на псевдокоде записать так:

```
нц для j от N-1 до 1 шаг -1
  если A[j+1]<A[j] то
    поменять местами A[j] и A[j+1]
  все
кц
```

Здесь j — целочисленная переменная. Обратите внимание, что на очередном шаге сравниваются элементы $A[j]$ и $A[j+1]$, поэтому цикл начинается с $j = N - 1$. Если начать с $j = N$, то на первом же шаге получаем выход за границы массива — обращение к элементу $A[N+1]$.

За один проход такой цикл ставит на место один элемент. Чтобы «подтянуть» второй элемент, нужно написать ещё один почти такой же цикл, который будет отличаться только конечным

значением *j* в заголовке цикла. Так как верхний элемент уже стоит на месте, его не нужно трогать:

```
нц для j от N-1 до 2 шаг -1
  если A[j+1]<A[j] то
    поменять местами A[j] и A[j+1]
  все
кц
```

При установке 3-го элемента конечное значение для *j* будет 3 и т. д. Таких циклов нужно сделать $N - 1$: на 1 меньше, чем количество элементов массива. Почему не N? Дело в том, что если $N - 1$ элементов поставлены на свои места, то оставшийся автоматически встает на своё место — другого места нет. Поэтому полный алгоритм сортировки представляет собой такой вложенный цикл:

```
нц для i от 1 до N-1
  нц для j от N-1 до i шаг -1
    если A[j+1]<A[j] то
      поменять местами A[j] и A[j+1]
    все
  кц
кц
```

Записать полную программу на выбранном языке вы можете самостоятельно.

Метод выбора

Ещё один популярный простой метод сортировки — метод выбора, при котором на каждом этапе выбирается минимальный элемент (из оставшихся) и ставится на свое место. Алгоритм в общем виде можно записать так:

```
нц для i от 1 до N-1
  найти номер nMin минимального элемента из A[i]..A[N]
  если i<>nMin то
    поменять местами A[i] и A[nMin]
  все
кц
```

Здесь перестановка происходит только тогда, когда найденный минимальный элемент стоит не на своём месте, т. е. i<>nMin. Поскольку поиск минимального элемента выполняется в цикле, этот алгоритм сортировки также представляет собой вложенный цикл:

```
нц для i от 1 до N-1
  nMin:= i
  нц для j от i+1 до N
    если A[j]<A[nMin] то
      nMin:=j
    все
  кц
  если i<>nMin то
    поменять местами A[i] и A[nMin]
  все
кц
```

«Быстрая сортировка»



Ч. Э. Хоар (род. в 1934)

Методы сортировки, описанные ранее, работают медленно для больших массивов данных (более 1000 элементов). Поэтому в середине XX века математики и программисты серьезно занимались разработкой более эффективных алгоритмов сортировки. Один из самых популярных «быстрых» алгоритмов, разработанный в 1960 г. английским учёным *Чарльзом Хоаром*, так и называется — «быстрая сортировка» (англ. *quicksort*).

Будем исходить из того, что сначала лучше делать перестановки элементов массива на большом расстоянии. Предположим, что у нас есть n элементов и известно, что они уже отсортированы в обратном порядке. Тогда за $n/2$ обменов можно отсортировать их как нужно — сначала поменять местами первый и последний, а затем последовательно двигаться с двух сторон к центру. Хотя это справедливо только тогда, когда порядок элементов обратный, подобная идея положена в основу алгоритма Quicksort.

Пусть дан массив A из n элементов. Выберем сначала наугад любой элемент массива (назовем его X). Обычно выбирают средний элемент массива, хотя это необязательно. На первом этапе мы расставим элементы так, что слева от некоторой границы находятся все числа, меньшие или равные X, а справа — большие или равные X:

$A[i] \le X$	$A[i] \ge X$

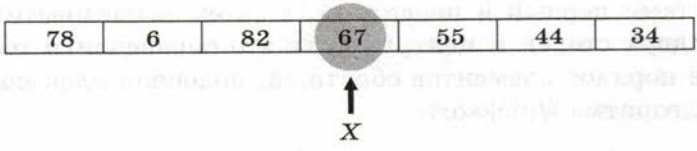
Заметим, что элементы, равные X, могут находиться в обеих частях.

Теперь элементы расположены так, что ни один элемент из первой части при сортировке не окажется во второй части и наоборот. Поэтому далее достаточно отсортировать отдельно каждую часть массива. Такой подход называют «разделяй и властвуй» (*англ.* divide and conquer).

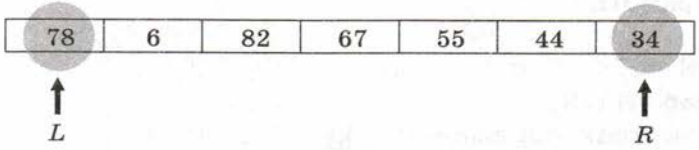
Лучше всего выбирать X так, чтобы в обеих частях было равное количество элементов. Такое значение X называется *медианой массива*. Однако для того, чтобы найти медиану, надо сначала отсортировать массив, т. е. заранее решить ту самую задачу, которую мы собираемся решить этим способом. Поэтому обычно в качестве X выбирают средний элемент массива.

Сначала будем просматривать массив слева до тех пор, пока не обнаружим элемент, который больше X (и, следовательно, должен стоять справа от X). Затем просматриваем массив справа до тех пор, пока не обнаружим элемент, меньший X (он должен стоять слева от X). Теперь поменяем местами эти два элемента и продолжим просмотр до тех пор, пока два «просмотра» не встретятся где-то в середине массива. В результате массив окажется разбитым на 2 части: левую со значениями, меньшими или равными X, и правую со значениями, большими или равными X. На этом первый этап («разделение») закончен. Затем такая же процедура применяется к обеим частям массива до тех пор, пока в каждой части не останется по одному элементу (таким образом, массив будет отсортирован).

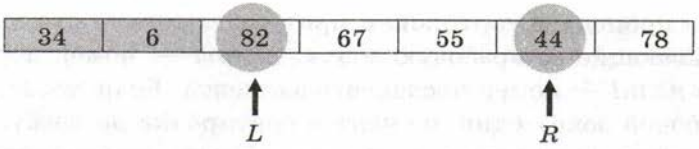
Чтобы понять сущность метода, рассмотрим пример. Пусть задан массив:



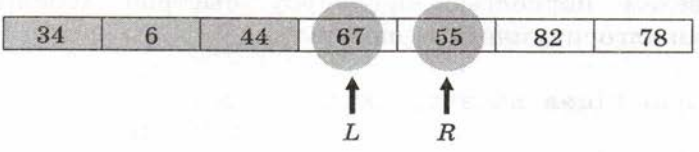
Выберем в качестве X средний элемент массива, т. е. 67. Найдем первый слева элемент массива, который больше или равен X и должен стоять во второй части. Это число 78. Обозначим индекс этого элемента через L. Теперь находим самый правый элемент, который меньше X и должен стоять в первой части. Это число 34. Обозначим его индекс через R:



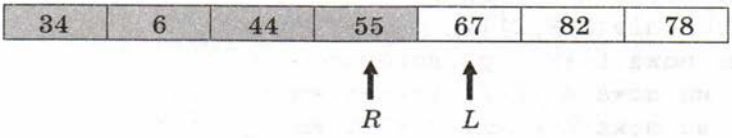
Теперь поменяем местами эти два элемента. Сдвигая переменную L вправо, а R — влево, находим следующую пару, которую надо переставить. Это числа 82 и 44:



Следующая пара элементов для перестановки — числа 67 и 55:



После этой перестановки дальнейший поиск приводит к тому, что переменная L становится больше R:



В результате все элементы массива, расположенные левее $A[L]$, меньше или равны X, а все элементы правее $A[R]$ — больше или равны X.

Теперь нужно применить тот же алгоритм к двум полученным частям массива: первая часть — с 1-го элемента до R-го элемента, вторая часть — с L-го до последнего элемента. Как вы знаете, такой приём называется *рекурсией*.

Чтобы не загромождать решение деталями оформления, которые сильно различаются в школьном алгоритмическом языке и в Паскале, предположим, что в нашей программе один **глобальный** массив A с индексами от 1 до N, который нужно сортировать. Термин «глобальный» означает, что массив доступен всем процедурам и функциям. Глобальные данные объявляются выше основной программы:

```
цел N = 5
целтаб A[1:N]
алг Быстрая сортировка
нач
...
кон
```

Тогда процедура сортировки принимает только два параметра, ограничивающие её «рабочую зону»: *nStart* — номер первого элемента, и *nEnd* — номер последнего элемента. Если *nStart* = *nEnd*, то в «рабочей зоне» один элемент и сортировка не требуется, т. е. нужно выйти из процедуры. В этом случае рекурсивные вызовы заканчиваются.

Приведём полностью процедуру быстрой сортировки на школьном алгоритмическом языке:

```
алг qSort(цел nStart, nEnd)
нач
  цел L, R, c, X
  если nStart>=nEnd то выход все
  L:=nStart; R:=nEnd;
  X:=A[div(L+R,2)]
  нц пока L<=R | разделение
    нц пока A[L]<X; L:=L+1 кц
    нц пока A[R]>X; R:=R-1 кц
    если L<=R то
      c:=A[L]; A[L]:=A[R]; A[R]:=c
      L:=L+1; R:=R-1
    все
  кц
  qSort(nStart, R) | рекурсивные вызовы
  qSort(L, nEnd)
кон
```

Для того чтобы отсортировать весь массив, нужно вызвать эту процедуру так:

```
qSort(1, N)
```

Скорость работы быстрой сортировки зависит от того, насколько удачно выбирается вспомогательный элемент X. Самый лучший случай — когда на каждом этапе массив делится на две равные части. Худший случай — когда в одной части оказывается только один элемент, а в другой — все остальные. При этом глубина рекурсии достигает N, что может привести к переполнению стека (нехватке стековой памяти).

Для того чтобы уменьшить вероятность худшего случая, в алгоритм вводят случайность: в качестве X на каждом шаге выбирают не середину рабочей части массива, а элемент со случайным номером:

```
X:=A[irand(L,R)]
```

В таблице 8.1 сравнивается время сортировки (в секундах) массивов разного размера, заполненных случайными значениями, с использованием трёх изученных алгоритмов.

Таблица 8.1

N	Метод пузырька	Метод выбора	Быстрая сортировка
1000	0,24 с	0,12 с	0,004 с
5000	5,3 с	2,9 с	0,024 с
15000	45 с	34 с	0,068 с

Как показывают эти данные, преимущество быстрой сортировки становится подавляющим при увеличении N.

Вопросы и задания

1. Что такое сортировка?
2. На какой идее основан метод пузырька? Метод выбора?
3. Объясните, зачем нужен вложенный цикл в описанных методах сортировки.
4. Сравните на примере метод пузырька и метод выбора. Какой из них требует меньше перестановок?

5. Расскажите про основные идеи метода «быстрой сортировки».
6. Как нужно изменить приведённые в параграфе алгоритмы, чтобы элементы массива были отсортированы по убыванию?
7. Как вы думаете, можно ли использовать метод «быстрой сортировки» для нечисловых данных, например для символьных строк?
8. От чего зависит скорость «быстрой сортировки»? Какой самый лучший и самый худший случай?
9. Как вы думаете, может ли метод «быстрой сортировки» работать дольше, чем метод выбора (или другой «простой» метод)? Если да, то при каких условиях?



Подготовьте сообщение

а) «Сортировка вставкой»
б) «Сортировка слиянием»
в) «Сортировка списков на языке Python»



Задачи

1. Отсортировать массив и найти количество различных чисел в нём.
2. Напишите программу, в которой сортировка выполняется «методом камня» — самый «тяжёлый» элемент опускается в конец массива.
3. Напишите вариант метода пузырька, который заканчивает работу, если на очередном шаге внешнего цикла не было перестановок.
4. Напишите программу, которая сортирует массив по возрастанию последней цифры числа.
5. Напишите программу, которая сортирует массив по убыванию суммы цифр числа.
6. Напишите программу, которая сортирует первую половину массива по возрастанию, а вторую — по убыванию (элементы из первой половины не должны попадать во вторую и наоборот).
7. Напишите программу, которая сортирует массив, а затем находит максимальное из чисел, встречающихся в массиве несколько раз.

*8. Напишите программу, которая сравнивает количество перестановок при сортировке одного и того же массива разными методами. Проведите эксперименты для возрастающей последовательности (уже отсортированной), убывающей (отсортированной в обратном порядке) и случайной.

§ 65
Двоичный поиск

Ранее мы уже рассматривали задачу поиска элемента в массиве и привели алгоритм, который сводится к просмотру всех элементов массива. Такой поиск называют **линейным**. Для массива из 1000 элементов нужно сделать 1000 сравнений, чтобы убедиться, что заданного элемента в массиве нет. Если число элементов (например, записей в базе данных) очень велико, время поиска может оказаться недопустимым, потому что пользователь не дождётся ответа.

Как вы помните, основная задача сортировки — облегчить последующий поиск данных. Вспомним, как мы ищем нужное слово (например, слово «гравицапа») в словаре. Сначала открываем словарь примерно в середине и смотрим, какие там слова. Если они начинаются на букву «Л», то слово «гравицапа» явно находится на предыдущих страницах, и вторую часть словаря можно не смотреть. Теперь так же проверяем страницу в середине первой половины, и т. д. Такой поиск называется **двоичным**. Понятно, что он возможен только тогда, когда данные предварительно отсортированы. Для примера на рис. 8.12 показан поиск числа $X = 44$ в отсортированном массиве.

Серым фоном выделены ячейки, которые уже не рассматриваются, потому что в них не может быть заданного числа. Перемен-

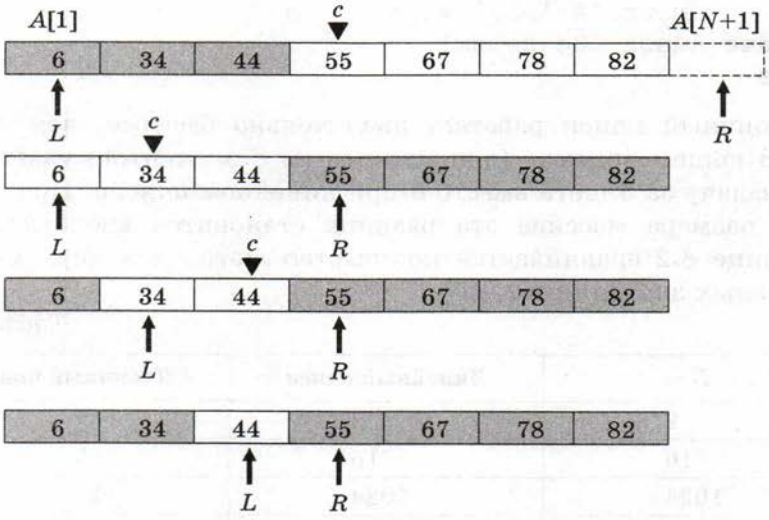


Рис. 8.12

ные L и R ограничивают «рабочую зону» массива: L содержит номер первого элемента, а R — номер элемента, *следующего за последним*. Таким образом, нужный элемент (если он есть) находится в части массива, которая начинается с элемента $A[L]$ и заканчивается элементом $A[R–1]$.

На каждом шаге вычисляется номер среднего элемента «рабочей зоны», он записывается в переменную c. Если $X<A[c]$, то этот элемент может находиться только левее $A[c]$, и правая граница R перемещается в с. В противном случае нужный элемент находится правее середины или совпадает с $A[c]$; при этом левая граница L перемещается в c.

Поиск заканчивается при выполнении условия $L = R - 1$, когда в рабочей зоне остаётся один элемент. Если при этом $A[L] = X$, то в результате найден элемент, равный X, иначе такого элемента нет.

```
цел L, R, c
L:=1; R:=N+1
нц пока L<R-1
  c:=div(L+R,2)
  если X<A[c] то
       R:=c
  иначе L:=c
  все
кц
если A[L]=X то
     вывод 'A[',L,']=',X
иначе вывод 'Не нашли!'
все
```

Двоичный поиск работает значительно быстрее, чем линейный. В нашем примере (для массива из 8 элементов) удаётся решить задачу за 3 шага вместо 8 при линейном поиске. При увеличении размера массива эта разница становится впечатляющей. В таблице 8.2 сравнивается количество шагов для двух методов при разных значениях N.

Таблица 8.2

N	Линейный поиск	Двоичный поиск
2	2	2
16	16	5
1024	1024	11
1048576	1048576	21

Однако при этом нельзя сказать, что двоичный поиск лучше линейного. Нужно помнить, что данные необходимо предварительно отсортировать, а это может занять значительно больше времени, чем сам поиск. Поэтому такой подход эффективен, если данные меняются (и сортируются) редко, а поиск выполняется часто. Такая ситуация характерна, например, для баз данных.

Вопросы и задания

1. Почему приведённый в параграфе алгоритм поиска называется двоичным?
2. Как можно примерно подсчитать количество шагов при двоичном поиске?

Подготовьте сообщение

а) «Двоичный поиск в нашей жизни»
б) «Линейный и двоичный поиск: достоинства и недостатки»

Задачи

1. Напишите программу, которая сортирует массив по убыванию и ищет в нём все значения, равные введённому числу.
2. Напишите программу, которая считает среднее число шагов при двоичном поиске для массива из 32 элементов в диапазоне 0..100. Для поиска используйте 1000 случайных чисел в этом же диапазоне.

§ 66
Символьные строки

Что такое символьная строка?

Если в середине XX века первые компьютеры использовались, главным образом, для выполнения сложных математических расчётов, сейчас их основная работа — обработка текстовой (символьной) информации.

Символьная строка — это последовательность символов, расположенных в памяти рядом (в соседних ячейках). Для работы с символами во многих языках программирования есть переменные специального типа: символы и символьные массивы. Казалось бы, массив — это и есть символьная строка, однако в школьном алго-

ритмическом языке и в Паскале для строк используются специальные типы данных.

Почему возникла потребность в создании такого специального типа данных? Дело в том, что массив — это группа символов, каждый из которых независим от других. Это значит, что (в школьном алгоритмическом языке и в Паскале) вводить символьный массив нужно посимвольно, в цикле. Более того, размер массива задается при объявлении, и не очень ясно, как использовать массивы для работы со строками переменной длины. Поэтому нужен новый тип данных, который позволяет:

- работать с целой символьной строкой как с единым объектом;
- использовать строки переменной длины.

Такой тип данных в школьном алгоритмическом языке называется **литерным** и обозначается **лит** (от слова «литерный» — буквенный), а в Паскале — строковым и обозначается **string** (в переводе с англ. — строка). Вот пример объявления строки:

```
лит s                          var s: string;
```

Для того чтобы записать в строку значение, используют оператор присваивания:

```
s:='Вася пошёл гулять'         s:='Вася пошёл гулять';
```

или оператор ввода с клавиатуры:

```
ввод s                         readln(s);
```

Обратите внимание, что при вводе строк в Паскале нужно использовать оператор `readln` (англ. *read line* — читать до конца строки) вместо `read`.

Существуют стандартные функции, которые определяют длину строки (количество символов в ней). В школьном алгоритмическом языке такая функция называется `длин`, а в Паскале — `Length` (в переводе с англ. — длина). В следующем примере в целочисленную переменную *n* записывается длина строки *s*:

```
n:=длин(s)                     n:=Length(s);
```

Для того чтобы работать с отдельными символами строки, к ним нужно обращаться так же, как к элементам массива: в квад-

ратных скобках записывают номер символа. Например, так можно изменить четвёртый символ строки на 'а':

```
s[4]:='а'
```

```
s[4]:='а';
```

Приведём полную программу, которая вводит строку с клавиатуры, заменяет в ней все буквы 'а' на буквы 'б' и выводит полученную строку на экран.

```
алг Замена а на б
нач
  лит s
  вывод 'Введите строку: '
  ввод s
  цел i
  нц для i от 1 до длин(s)
    если s[i]='а'
      то s[i]:='б'
    все
  кц
  вывод s
кон
```

```
program ReplaceAB;
var s: string;
    i: integer;
begin
  writeln('Введите строку');
  readln(s);
  for i:=1 to Length(s) do
    if s[i]='а' then
      s[i]:='б';
  writeln(s);
end.
```

Операции со строками

Оператор + используется для **объединения (сцепления)** строк, эта операция иногда называется **конкатенацией**. Например:

```
s1:='Привет'
s2:='Вася'
s:=s1 + ', ' + s2 + '!'
```

Здесь и далее считаем, что в программе объявлены **строковые (литерные)** переменные *s*, *s*1 и *s*2. В результате выполнения приведённой программы в строку *s* будет записано значение `'Привет, Вася!'`.

Для того чтобы выделить часть строки (**подстроку**), в школьном алгоритмическом языке применяется операция получения среза[1], например *s*[3:7] означает символы строки *s* с 3-го по 7-й включительно. В Паскале для этого используется функция `Copy`,

[1] В описании школьного алгоритмического языка системы КуМир эта операция называется «вырезка» (англ. *slicing*).

она принимает три параметра: имя строки, номер начального символа и количество символов. Например, оба следующих фрагмента копируют в строку *s1* символы строки *s* с 3-го по 7-й (всего 5 символов):

```
s:='123456789'
s1:=s[3:7]
```

```
s:='123456789';
s1:=Copy(s,3,5);
```

В строку *s1* будет записано значение '34567'.

Для удаления части строки нужно вызвать соответствующую подпрограмму, указав название строки, номер начального символа и число удаляемых символов:

```
s:='123456789'
удалить(s, 3, 6)
```

```
s:='123456789';
Delete(s, 3, 6);
```

В переменной *s* остаётся значение '129' (удаляются 6 символов, начиная с 3-го).

И в школьном алгоритмическом языке, и в Паскале удаление выполняет процедура, которая изменяет переданную ей строку. В Паскале нужно вызывать процедуру, которая изменяет исходную строку.

При вставке символов соответствующей процедуре передают вставляемый фрагмент, имя исходной строки и номер символа, с которого начинается вставка:

```
s:='123456789'
вставить('ABC', s, 3)
```

```
s:='123456789';
Insert('ABC', s, 3);
```

Переменная *s* получит значение '12ABC3456789'.

Поиск в строках

Существуют функции для поиска подстроки (и отдельного символа) в строке. Им нужно передать образец для поиска и строку, в которой надо искать.

```
s:='Здесь был Вася.'
n:=позиция('с', s)
если n>0 то
  вывод 'Номер символа ', n
иначе
  вывод 'Символ не найден.'
все
```

```
s:='Здесь был Вася.';
n:=Pos('с', s);
if n>0 then
  write('Номер символа ', n)
else
  write('Символ не найден.');
```

Функция позиция возвращает целое число — номер символа, с которого начинается образец (буква 'с') в строке *s*. Если в строке несколько образцов, функция находит первый из них. Если образец не найден, функция позиция возвращает 0 (недействительный номер символа).

В языке Паскаль функция Pos (от англ. *position* — позиция) работает точно так же.

Пример обработки строк

Предположим, что с клавиатуры вводится строка, содержащая имя, отчество и фамилию человека, например:

```
Василий Алибабаевич Хрюндиков
```

Каждые два слова разделены одним пробелом, в начале строки пробелов нет. В результате обработки должна получиться новая строка, содержащая фамилию и инициалы:

```
Хрюндиков В.А.
```

Возможный алгоритм решения этой задачи может быть на псевдокоде записан так:

```
ввести строку s
позиция в строке s первый пробел
имя:=всё, что слева от первого пробела
удалить из строки s имя с пробелом
позиция в строке s первый пробел
отчество:=всё, что слева от первого пробела
удалить из строки s отчество с пробелом | осталась фамилия
s:= s + ' ' + имя[1] + '.' + отчество[1] + '.'
```

Мы последовательно выделяем из строки три элемента: имя, отчество и фамилию, используя тот факт, что они разделены одиночными пробелами. После того как имя сохранится в отдельной переменной, в строке *s* останутся только отчество и фамилия. После «изъятия» отчества остаётся только фамилия. Теперь нужно собрать строку-результат из частей: «сцепить» фамилию и первые буквы имени и отчества, поставив пробелы и точки между ними. Для выполнения всех операций будем использовать стандартные функции, описанные выше.

Приведём полные программы на школьном алгоритмическом языке:

```
алг ФИО
нач
  лит s, name, name2
  цел n
  вывод 'Введите имя, отчество и фамилию:'
  ввод s
  n:=позиция(' ', s);
  name:=s[1:n-1]        | вырезать имя
  удалить(s, 1, n)
  n:=позиция(' ', s)
  name2:=s[1:n-1]       | вырезать отчество
  удалить(s, 1, n)      | осталась фамилия
  s:=s + ' ' + name[1] + '.' + name2[1] + '.'
  вывод s
кон
```

и на Паскале:

```
program FIO;
var s, name, name2: string;
    n: integer;
begin
  write('Введите имя, отчество и фамилию: ');
  readln(s);
  n:=Pos(' ', s);
  name:=Copy(s, 1, n-1);    {вырезать имя}
  Delete(s, 1, n);
  n:=Pos(' ', s);
  name2:=Copy(s, 1, n-1);   {вырезать отчество}
  Delete(s, 1, n);          {осталась фамилия}
  s:=s + ' ' + name[1] + '.' + name2[1] + '.';
  writeln(s);
end.
```

Преобразования число ↔ строка

В практических задачах часто нужно преобразовать число, записанное в виде цепочки символов, в числовое значение, и наоборот. Для этого в школьном алгоритмическом языке есть стандартные функции:

- лит_в_цел — переводит строку в целое число;
- лит_в_вещ — переводит строку в вещественное число;
- цел_в_лит — переводит целое число в строку;
- вещ_в_лит — переводит вещественное число в строку.

Разберём такой пример:

```
лит s, цел N, вещ X, лог OK
s:='123'
N:=лит_в_цел(s, OK) | N = 123
если не OK то вывод 'Ошибка!' все
s:='123.456';
X:=лит_в_вещ(s, OK) | X = 123.456
если не OK то вывод 'Ошибка!' все
```

Строку не всегда можно преобразовать в число (например, если в ней содержатся буквы). Поэтому функции лит_в_цел и лит_в_вещ используют второй параметр — логическую переменную *ОК*, в которую записывается да, если операция закончилась успешно, и «нет» в противном случае. При обратном преобразовании таких проблем быть не может:

```
N:=123
s:=цел_в_лит(N) | s = '123'
X:=123.456
s:=вещ_в_лит(X) | s = '123.456'
```

В языке Паскаль строка преобразуется в число (целое или вещественное) с помощью процедуры Val:

```
var r: integer;
...
s:='123';
Val(s, N, r); {N=123}
s:='123.456';
Val(s, X, r); {X=123.456}
```

Третий параметр *r* служит для того, чтобы зафиксировать ошибку: если после вызова процедуры Val он равен нулю, то ошибки не было, иначе в переменную *r* записывается номер первого ошибочного символа.

Преобразование числа в строку выполняет процедура `Str`:

```
N:=123;
Str(N, s); {s='123'}
X:=123.456;
Str(X, s); {s='1.234560E+002'}
Str(X:10:3, s); {s='123.456'}
```

По умолчанию вещественные числа представлены в научном (экспоненциальном) формате (`'1.234560E+002'` означает $1{,}23456 \cdot 10^2$). В последней строке примера используется **форматный вывод**: запись `X:10:3` означает «вывести число в 10 позициях с 3 знаками в дробной части».

Строки в процедурах и функциях

Строки можно передавать в процедуры и функции как аргументы (значения параметров), а также возвращать как результат функций. Построим процедуру, которая заменяет в строке *s* все вхождения слова-образца *wOld* на слово-замену *wNew* (здесь *wOld* и *wNew* — это имена переменных, а выражение «слово *wOld*» означает «слово, записанное в переменную *wOld*»).

Сначала разработаем алгоритм решения задачи. На первый взгляд кажется, что можно написать такой алгоритм на псевдокоде:

```
нц пока слово wOld есть в строке s
 удалить слово wOld из строки
 вставить на это место слово wNew
кц
```

Однако такой алгоритм работает неверно, если слово *wOld* входит в состав *wNew*, например, нужно заменить `'12'` на `'A12B'` (покажите самостоятельно, что это приведет к зацикливанию).

Чтобы избежать подобных проблем, попробуем накапливать результат в другой символьной строке *res*, удаляя из строки *s* уже обработанную часть. Предположим, что на некотором шаге в оставшейся части строки *s* обнаружено слово *wOld* (рис. 8.13, *а*).

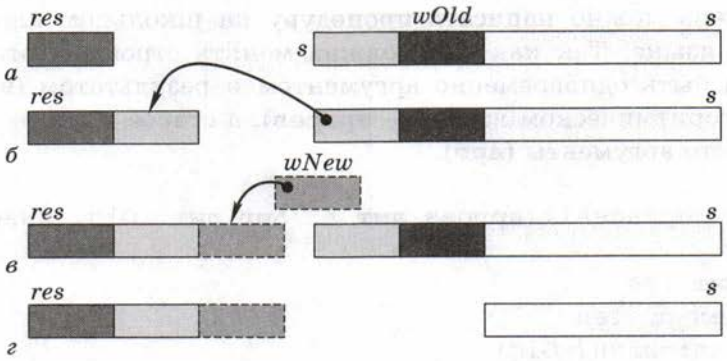


Рис. 8.13

Теперь нужно выполнить следующие действия:

1) ту часть строки *s*, которая стоит слева от образца, «прицепить» в конец строки *res* (рис. 8.13, *б*);
2) «прицепить» в конец строки *res* слово-замену *wNew* (рис. 8.13, *в*);
3) удалить из строки s начальную часть, включая найденное слово-образец (рис. 8.13, *г*).

Далее все эти операции (начиная с поиска слова *wOld* в строке *s*) выполняются заново до тех пор, пока строка *s* не станет пустой. Если очередное слово-образец найти не удалось, вся оставшаяся строка *s* приписывается в конец строки-результата, и цикл заканчивается.

В начале работы алгоритма в строку *res* записывается пустая строка `''`, не содержащая ни одного символа. В таблице 8.3 приведён протокол работы алгоритма замены для строки `'12.12.12'`, в которой нужно заменить слово `'12'` на `'A12B'`.

Таблица 8.3

Рабочая строка *s*	Результат *res*
'12.12.12'	''
'.12.12'	'A12B'
'.12'	'A12B.A12B'
''	'A12B.A12B.A12B'

Теперь можно написать процедуру на школьном алгоритмическом языке. Так как она должна менять строку *s*, эта строка должна быть одновременно аргументом и результатом (в школьном алгоритмическом языке — **аргрез**), а старое и новое слова — это просто аргументы (**арг**).

```
алг replaceAll(аргрез лит s, арг лит wOld, wNew)
нач
  лит res
  цел p, len
  len:=длин(wOld)
  res:=''
  нц пока длин(s)>0
    p:=позиция(wOld, s)
    если p<0 то res:=res+s; выход все
    если p>1 то res:=res+s[1:p-1] все
    res:=res+wNew
    если p+len>длин(s) то
      s:=''
    иначе s:=s[p+len:длин(s)]
    все
  кц
  s:=res
кон
```

Дадим некоторые пояснения к программе. Переменная *p* — это номер первого символа первого найденного слова-образца *wOld*, а в переменной *len* записана длина этого слова. Если после поиска слова значение *p* меньше нуля (образец не найден), происходит выход из цикла:

```
если p<0 то res:=res+s; выход все
```

Если $p > 1$, то слева от образца есть какие-то символы, и их нужно «прицепить» к строке *res*:

```
если p>1 то res:=res+s[1:p-1] все
```

Условие `p+len>длин(s)` означает, что образец стоит в самом конце слова, при этом остаток строки *s* — пустая строка.

В конце программы результат записывается на место исходной строки *s*.

Приведём пример использования процедуры:

```
алг Замена всех
нач
 лит s='12.12.12'
 replaceAll(s, '12', 'A12B')
 вывод s
кон
```

Построенную выше процедуру можно легко превратить в функцию. Для этого нужно:

- в заголовке функции указать, что она возвращает строку (добавить ключевое слово **лит**);
- все параметры должны быть аргументами (нужно убрать **аргрез** и **арг**);
- поскольку в школьном алгоритмическом языке нельзя менять аргументы внутри процедуры, назовём первый параметр (исходную строку) *s*0, и введём дополнительную переменную *s* для работы со строкой в процедуре;
- в конце нужно записать результат во встроенную переменную *знач*, а не в *s*.

Ниже показаны все изменённые части подпрограммы:

```
алг лит replaceAll(лит s0, wOld, wNew)
нач
  лит s
  s:=s0
  ...  | тело процедуры
  знач:=res
кон
```

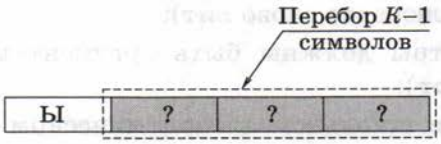
Вызывать функцию можно таким образом:

```
алг Замена всех
нач
  лит s='12.12.12'
  s:=replaceAll(s, '12', 'A12B')
  вывод s
кон
```

Рекурсивный перебор

В алфавите языке племени «тумба-юмба» четыре буквы: «Ы», «Ш», «Ч» и «О». Нужно вывести на экран все слова, состоящие из K букв, которые можно построить из букв этого алфавита.

Это типичная задача на перебор вариантов, которую удобно свести к задаче меньшего размера. Будем определять буквы слова последовательно, одну за другой. Первая буква может быть любой из четырёх букв алфавита. Предположим, что сначала первой мы поставили букву 'Ы'. Тогда для того, чтобы получить все варианты с первой буквой 'Ы', нужно перебрать все возможные комбинации букв на оставшихся $K - 1$ позициях:



Далее поочерёдно ставим на первое место все остальные буквы, повторяя процедуру:

```
алг перебор К символов
нач
 w[1]:='Ы'; перебор последних К-1 символов
 w[1]:='Ш'; перебор последних К-1 символов
 w[1]:='Ч'; перебор последних К-1 символов
 w[1]:='О'; перебор последних К-1 символов
кон
```

Здесь через w обозначена символьная строка, в которой хранится рабочее слово. Таким образом, задача для слов длины K свелась к 4 задачам для слов длины $K - 1$. Как вы знаете, такой прием называется *рекурсией*, а процедура — рекурсивной.

Когда рекурсия должна закончиться? Тогда, когда все символы будут расставлены. При этом нужно вывести получившееся слово на экран и выйти из процедуры.

Подсчитаем количество всех возможных слов длины K. Очевидно, что слов длины 1 всего 4. Добавляя ещё одну букву, получаем $4 \cdot 4 = 16$ комбинаций, для трёх букв — $4 \cdot 4 \cdot 4 = 64$ слова и т. д. Таким образом, из 4 букв можно составить 4^K слов длины K.

В основной программе построим слово (символьную строку) *word* нужной длины (что в ней будет записано, не имеет значе-

ния). Процедуре TumbaWords передаётся алфавит в виде символьной константы, слово *word* и число уже установленных символов (в начале — 0):

```
алг ЫШЧО
нач
  лит word = '...';  | из K символов
  TumbaWords('ЫШЧО', word, 0)
кон
```

В процедуре используется описанный выше рекурсивный алгоритм:

```
алг TumbaWords(лит A, w0, цел N)
нач
  если N = длин(w0) то | слово построено
    вывод w0, нс
    выход
  все
  цел i
  лит w
  w:=w0
  нц для i от 1 до длин(A)
    w[N+1]:=A[i]
    TumbaWords(A, w, N+1) | рекурсия
  кц
кон
```

Если условие в начале процедуры ложно (не все символы расставлены), в цикле перебираем все символы алфавита и поочерёдно ставим их на первое свободное место, а затем вызываем рекурсивно эту же процедуру, увеличивая третий параметр на 1. Поскольку школьный алгоритмический язык не позволяет менять переменную-аргумент, мы вынуждены ввести дополнительную локальную переменную *w*.

Приведём аналогичную программу на Паскале:

```
program YSCHO;
var word: string;
procedure TumbaWords(A, w: string; N: integer);
var i: integer;
```

```
begin
  if N=Length(w) then begin
    writeln(w);
    exit;
  end;
  for i:=1 to Length(A) do begin
    w[N+1]:=A[i];
    TumbaWords(A, w, N+1);
  end;
end;
begin
   word:= '...';
   TumbaWords('ЫШЧО', word, 0);
end.
```

Сравнение и сортировка строк

Строки, как и числа, можно сравнивать. Для строк, состоящих из одних букв (русских или латинских), результат сравнения очевиден: меньше будет та строка, которая идет раньше в алфавитном порядке. Например, слово «паровоз» будет «меньше», чем слово «пароход»: они отличаются в пятой букве: 'в' < 'х'. Более короткое слово, которое совпадает с началом более длинного, тоже будет стоять раньше в алфавитном списке, поэтому 'пар' < 'парк'.

Но откуда компьютер «знает», что такое алфавитный порядок? И как сравнивать слова, в которых есть строчные и заглавные буквы, а также цифры и другие символы? Что больше, 'ПАР', 'Пар' или 'пар'? Оказывается, при сравнении строк используются коды символов. Тогда получается, что:

'ПАР' < 'Пар' < 'пар'.

Возьмём пару 'ПАР' и 'Пар'. Первый символ в обоих словах одинаков, а второй отличается — в первом слове буква заглавная, а во втором — такая же, но строчная. Во всех современных таблицах символов (включая UNICODE) заглавные буквы стоят раньше строчных и поэтому имеют меньшие коды. Поэтому 'А' < 'а', 'П' < 'а' и 'ПАР' < 'Пар' < 'пар'.

А как же с другими символами (цифрами, латинскими буквами)? Цифры стоят в кодовой таблице по порядку, причём раньше, чем латинские буквы; латинские буквы — раньше, чем русские;

заглавные буквы (русские и латинские) — раньше, чем соответствующие строчные. Поэтому:

'5STEAM' < 'STEAM' < 'Steam' < 'steam' < 'ПАР' < 'Пар' < 'пар'.

Сравнение строк используется, например, при сортировке. Рассмотрим такую задачу: надо ввести с клавиатуры 10 фамилий и вывести их на экран в алфавитном порядке.

Для сортировки удобно выделить массив строк, который в школьном алгоритмическом языке объявляется как **литтаб**, а в Паскале — **array of string**. Ввод и вывод выполняются с помощью стандартных циклов, а сортировка — любым известным способом, например методом пузырька:

```
алг Сортировка строк
нач
  цел i, j, N=10
  литтаб S[1:N]
  лит s1
  нц для i от 1 до N
    ввод S[i]
  кц
  нц для i от 1 до N-1
    нц для j от N-1 до i шаг -1
      если S[j+1]<S[j] то
        s1:=S[j];
        S[j]:=S[j+1];
        S[j+1]:=s1
      все
    кц
  кц
  нц для i от 1 до N
    вывод S[i], нс
  кц
кон
```

```
program StringSort;
const N=10;
var i,j: integer;
  s1: string;
  S: array[1..N] of string;
begin
  for i:=1 to N do
    readln(S[i]);
  for i:=1 to N do
  for j:=N-1 downto i do
    if S[j+1]<S[j] then begin
      s1:=S[j];
      S[j]:=S[j+1];
      S[j+1]:=s1;
    end;
  for i:=1 to N do
    writeln(S[i]);
end.
```

Вопросы и задания

1. Что такое символьная строка?
2. Почему неудобно заменять строки массивами символов?
3. Как объявляются строки в школьном алгоритмическом языке и в Паскале?
4. Как обращаться к элементу строки с заданным номером?
5. Как вычисляется длина строки?
6. Что обозначает операция «+» применительно к строкам?
7. Перечислите основные операции со строками и соответствующие им стандартные функции.
8. Как определить, что при поиске в строке образец не найден?
9. Чем различаются средства школьного алгоритмического языка и Паскаля для работы со строками?
10. Как преобразовать число из символьного вида в числовой и обратно?
11. Почему строку не всегда можно преобразовать в число? Как определить, что преобразование закончилось неудачно?
12. Объясните выражение «рекурсивный перебор».
13. Сравните на примерах рекурсивные и нерекурсивные методы решения переборных задач.

Подготовьте сообщение

а) «Символьные переменные в языке Си»
б) «Символьные строки в языке Python»

Задачи

1. Напишите программу, которая во введённой символьной строке заменяет все буквы «а» на буквы «б» и наоборот, заглавные — на заглавные, строчные — на строчные. При вводе строки 'абсАБС' должен получиться результат 'басБАС'.
2. Введите символьную строку и проверьте, является ли она *палиндромом* (палиндром читается одинаково в обоих направлениях, например: «казак»).
3. Введите адрес файла и «разберите» его на части, разделённые знаком «/». Каждую часть выведите в отдельной строке.
4. Введите строку, в которой записана сумма натуральных чисел, например '1+25+3'. Вычислите это выражение.
5. Введите с клавиатуры в одну строку фамилию, имя и отчество, разделив их пробелом. Выведите инициалы и фамилию. Например, при вводе строки 'Иванов Пётр Семёнович' должно получиться 'П.С. Иванов'.

6. Разберитесь, как работает ещё одна функция замены:

```
алг лит ReplaceBad(лит s0, sOld, sNew)
нач
  лит s
  s:=s0
  цел p, len
  len:=длин(sOld)
  p:=позиция(sOld,s)
  нц пока p>0
    удалить(s, p, len)
    вставить(sNew, s, p)
    p:=позиция(sOld, s)
  кц
  знач:=s
кон
```

Приведите пример входных данных, при которых эта функция работает неправильно.

7. Напишите рекурсивную версию процедуры `replaceAll`. Сравните две версии. Какую из них вы рекомендуете выбрать и почему?
8. Напишите функцию, которая изменяет в имени файла расширение на заданное (например, на bak). Функция принимает два параметра: имя файла и нужное расширение. Учтите, что в исходном имени расширение может быть пустым.
9. Напишите функцию, которая определяет, сколько раз входит в символьную строку заданное слово.
10. С клавиатуры вводится число N, обозначающее количество футболистов команды «Бублик», а затем — N строк, в каждой из которых — информация об одном футболисте в таком формате:

<Фамилия> <Имя> <год рождения> <голы>

Данные разделяются одним пробелом. Нужно подсчитать, сколько футболистов, родившихся в период с 1998 по 2000 г., не забили мячей вообще.

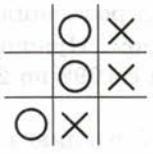
11. В условиях задачи 10 определите фамилию и имя футболиста, забившего наибольшее число голов, и количество забитых им голов.
12. В условиях задачи 10 выведите в алфавитном порядке фамилии и имена всех футболистов, которые забили хотя бы один гол. В списке не более 100 футболистов.
13. Измените программу рекурсивного перебора так, чтобы длину слова можно было ввести с клавиатуры.
14. Выведите на экран все слова из K букв, в которых буква «Ы» встречается более 1 раза, и подсчитайте их количество.

15. Выведите на экран все слова из K букв, в которых есть одинаковые буквы, стоящие рядом (например, «ЫШШШО»), и подсчитайте их количество.
16. В языке племени «тумба-юмба» запрещено ставить две гласные буквы подряд. Выведите все слова длины K, удовлетворяющие этому условию, и найдите их количество.
*17. Напишите программу перебора слов заданной длины, не использующую рекурсию. Попробуйте составить функцию, которая на основе некоторой комбинации вычисляет следующую за ней.
*18. Перестановки. К вам пришли K гостей. Напишите программу, которая выводит все *перестановки* — способы посадить их за столом. Гостей можно обозначить латинскими буквами.

§ 67 Матрицы

Что такое матрицы?

Многие программы работают с данными, организованными в виде таблиц. Например, при составлении программы для игры в крестики-нолики нужно запоминать состояние каждой клетки квадратной доски. Можно поступить так: пустым клеткам присвоить код –1, клетке, где стоит нолик, — код 0, а клетке с крестиком — код 1. Тогда информация о состоянии поля может быть записана в виде таблицы (рис. 8.14).



	1	2	3
1	–1	0	1
2	–1	0	1
3	0	1	–1

Рис. 8.14

Такие таблицы называются **матрицами** или **двумерными массивами**. Каждый элемент матрицы, в отличие от обычного (линейного) массива, имеет два индекса — номер строки и номер столбца. На рисунке 8.14 серым фоном выделен элемент, находящийся на пересечении второй строки и третьего столбца.

Матрица — это прямоугольная таблица, составленная из элементов одного типа (чисел, строк и т. д.). Каждый элемент матрицы имеет два индекса — номера строки и столбца.

При объявлении матриц указывают два диапазона индексов (для строк и столбцов):

```
цел N=3, M=4
целтаб A[1:N,1:M]
вещтаб X[-3:0,-8:M]
логтаб L[1:N,0:1]
```

```
const N=3, M=4;
var A: array[1..N,1..M] of integer;
    X: array[-3..0,-8..M] of double;
    L: array[1..N,0..1] of boolean;
```

Каждому элементу матрицы можно присвоить любое значение, допустимое для выбранного типа данных. Поскольку индексов два, для заполнения матрицы нужно использовать вложенный цикл. Далее в примерах будем считать, что объявлена матрица из N строк и M столбцов, а i и j — целочисленные переменные, обозначающие индексы строки и столбца. В следующем примере матрица заполняется случайными числами и выводится на экран:

```
нц для i от 1 до N
  нц для j от 1 до M
    A[i,j]:=irand(20,80)
    вывод A[i,j], ' '
  кц
  вывод, нс
кц
```

```
for i:=1 to N do
  for j:=1 to M do begin
    A[i,j]:=random(20)+80;
    write(A[i,j]:3)
  end;
  writeln
end;
```

Такой же двойной цикл нужно использовать для перебора всех элементов матрицы. Вот как вычисляется сумма значений всех элементов:

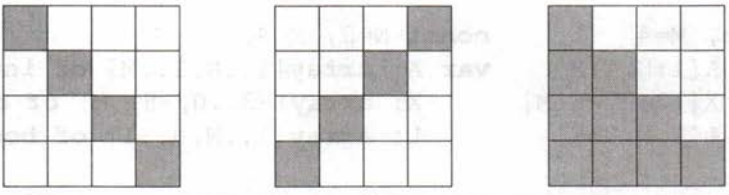
```
s:=0
нц для i от 1 до N
  нц для j от 1 до M
    s:=s+A[i,j]
  кц
кц
```

```
s:=0
for i:=1 to N do
  for j:=1 to M do
    s:=s+A[i,j];
```

Обработка элементов матрицы

Покажем, как можно обработать (например, сложить) некоторые элементы **квадратной матрицы** A, содержащей N строк и N столбцов.

На рисунке 8.15, *а* выделена главная диагональ матрицы, на рис. 8.15, *б* — вторая, побочная диагональ, на рис. 8.14, *в* — главная диагональ и все элементы под ней.



а *б* *в*

Рис. 8.15

Главная диагональ — это элементы $A[1,1]$, $A[2,2]$, ..., $A[N,N]$, т. е. элементы, у которых номер строки равен номеру столбца. Для перебора этих элементов нужен один цикл:

```
нц для i от 1 до N
  | работаем с A[i,i]
кц
```

```
for i:=1 to N do begin
  {работаем с A[i,i] }
end;
```

Элементы **побочной диагонали** — это $A[1,N]$, $A[2,N-1]$, ..., $A[N,1]$. Заметим, что сумма номеров строки и столбца для каждого элемента равны $N + 1$, поэтому получаем такой цикл перебора:

```
нц для i от 1 до N
  | работаем с A[i,N+1-i]
кц
```

```
for i:=1 to N do begin
  {работаем с A[i,N+1-i] }
end;
```

В случае обработки всех элементов на главной диагонали и под ней (см. рис. 8.15, *в*) нужен вложенный цикл: номер строки будет меняться от 1 до N, а номер столбца для каждой строки i — от 1 до i:

```
нц для i от 1 до N
  нц для j от 1 до i
    | работаем с A[i,j]
  кц
кц
```

```
for i:=1 to N do begin
  for j:=1 to i do begin
    {работаем с A[i,j] }
  end
end;
```

Чтобы переставить строки или столбцы, достаточно одного цикла. Например, переставим строки 2 и 4, используя вспомогательную целую переменную *c*:

```
нц для j от 1 до M
 c:=A[2,j]
 A[2,j]:=A[4,j]
 A[4,j]:=c
кц
```

```
for j:=1 to M do
 c:=A[2,j];
 A[2,j]:=A[4,j];
 A[4,j]:=c;
end;
```

Вопросы и задания

1. Что такое матрицы? Зачем они нужны?
2. Сравните понятия «массив» и «матрица».
3. Как вы думаете, можно ли считать, что первый индекс элемента матрицы — это номер столбца, а второй — номер строки?
4. Могут ли индексы элементов матрицы принимать отрицательные и нулевые значения?
5. Что такое главная и побочная диагонали матрицы?
6. Почему суммирование элементов главной диагонали требует одиночного цикла, а суммирование элементов под главной диагональю — вложенного?

Подготовьте сообщение

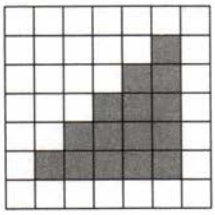
а) «Матрицы в языке Си»
б) «Матрицы в языке Python»

Задачи

1. Напишите программу, которая находит минимальный и максимальный элементы матрицы и их индексы.
2. Напишите программу, которая находит минимальный и максимальный элементы из элементов матрицы с чётными положительными значениями и их индексы. Учтите, что таких элементов в матрице может и не быть.
3. Напишите программу, которая выводит на экран строку матрицы, сумма значений элементов которой наибольшая.
4. Напишите программу, которая выводит на экран столбец матрицы, сумма значений элементов которого наименьшая.
5. Напишите программу, которая заполняет матрицу случайными числами, а затем записывает нули во все элементы выше главной диагонали.

6. Напишите программу, которая заполняет матрицу случайными числами, а затем записывает нули во все элементы выше побочной диагонали.

7. Напишите программу, которая заполняет матрицу размером 7 × 7 случайными числами, а затем записывает в элементы, отмеченные на рисунках, число 99.



а

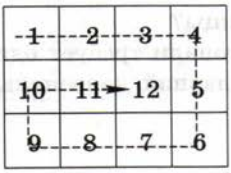


б

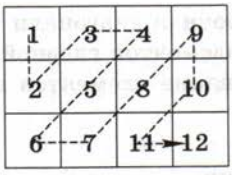


в

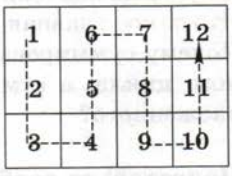
8. Заполните матрицу, содержащую N строк и M столбцов, натуральными числами по спирали и змейкой, как на рисунках.



а

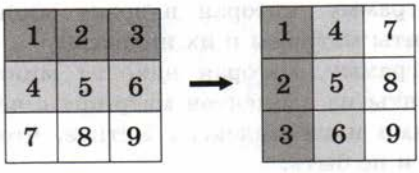


б



в

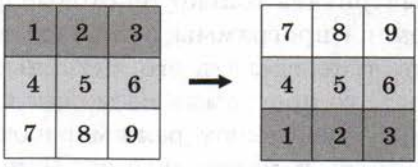
9. Заполните квадратную матрицу случайными числами и выполните её *транспонирование*: так называется процедура, в результате которой строки матрицы становятся столбцами, а столбцы — строками.



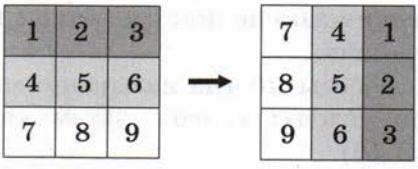
10. Пиксели рисунка закодированы числами от 0 до 255 (обозначающими яркость пикселей) в виде матрицы, содержащей N строк и M столбцов. Нужно преобразовать рисунок в чёрно-белый по следующему алгоритму:

- вычислить среднюю яркость пикселей по всему рисунку;
- все пиксели, яркость которых меньше средней, сделать чёрными (записать код 0), а остальные — белыми (код 255).

11. Пиксели рисунка закодированы числами (обозначающими цвет) в виде матрицы, содержащей N строк и M столбцов. Выполните отражение рисунка сверху вниз, как показано на рисунке.



12. Пиксели рисунка закодированы числами (обозначающими цвет) в виде матрицы, содержащей N строк и M столбцов. Выполните поворот рисунка вправо на 90 градусов, как показано на рисунке.



***13.** Напишите программу, которая играет с человеком в крестики-нолики.

***14.** В матрице, содержащей N строк и M столбцов, «записана» карта островного государства Лимония (нули обозначают море, а единицы — сушу). Все острова имеют форму прямоугольника. Напишите программу, которая по готовой карте определяет количество островов.

§ 68
Работа с файлами

Как работать с файлами?

Файл — это набор данных на диске, имеющий имя. С точки зрения программиста, бывают файлы двух типов:

1) **текстовые**, которые содержат текст, разбитый на строки; таким образом, из всех специальных символов в текстовых файлах могут быть только символы перехода на новую строку;

2) **двоичные**, в которых могут содержаться любые данные и любые коды без ограничений; в двоичных файлах хранятся рисунки, звуки, видеофильмы и т. д.

Мы будем рассматривать только текстовые файлы.

Работа с файлом из программы включает три этапа. Сначала надо **открыть** файл, т. е. сделать его активным для программы. Если файл не открыт, то программа не может к нему обращаться. При открытии файла указывают режим работы: чтение, запись или добавление данных в конец файла. Чаще всего открытый файл блокируется так, что другие программы не могут использовать его. Когда файл открыт (активен), программа выполняет все необходимые операции с ним. После этого нужно **закрыть** файл, т. е. освободить его, разорвать связь с программой. Именно при закрытии все последние изменения, сделанные программой в файле, записываются на диск.

Такой принцип работы иногда называют «принципом сэндвича», в котором три «слоя»: «хлеб», затем «начинка», и потом снова «хлеб» (рис. 8.16).

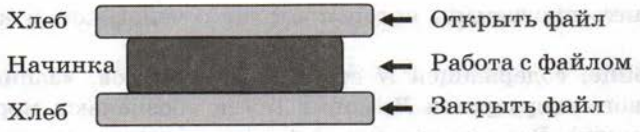


Рис. 8.16

В большинстве языков программирования с файлами работают через вспомогательные переменные (их называют указателями, идентификаторами и т. п.).

Например, в школьном алгоритмическом языке существуют стандартные функции:

- `открыть на чтение`;
- `открыть на запись`;
- `открыть на добавление`,

которые принимают имя файла (символьную строку) и возвращают ссылку на открытый файл (ключ файла), которая записывается в специальную переменную типа `файл`. Команда `закрыть` закрывает открытый файл:

```
файл Fin, Fout
Fin:=открыть на чтение('input.txt')
Fout:=открыть на запись('output.txt')
```

```
| здесь работаем с файлами
закрыть(Fin)
закрыть(Fout)
```

После закрытия файла файловую переменную можно использовать повторно, для работы с этим или другим файлом. После окончания работы программы все открытые файлы закрываются автоматически.

Если файл, который открывается на чтение, не найден, возникает ошибка. Если существующий файл открывается на запись, его содержимое уничтожается.

Чтение из текстовых файлов выполняет команда `ввод`, а запись — команда `вывод`. При работе с файлами на первом месте в списке параметров нужно указать файловую переменную, в которой записан номер открытого файла. В остальном чтение и запись происходит так же, как и для стандартных устройств — клавиатуры и текстового экрана (эта пара устройств называется *терминалом* или *консолью*).

Если в переменных *Fin* и *Fout* записаны ключи файлов, открытых соответственно на ввод и на вывод, можно написать так:

```
ввод Fin, a, b
вывод Fout, a, '+', b, '=', a+b
```

У нас получилась версия программы, которая складывает два числа, прочитанные из одного файла, и записывает результат в другой файл.

Как правило, текстовый файл — это «устройство» последовательного доступа к данным. Это значит, что для того, чтобы прочитать 100-е по счёту значение из файла, нужно сначала прочитать предыдущие 99. В своей внутренней памяти система хранит положение **указателя** (файлового курсора), который определяет текущее место в файле. При открытии файла указатель устанавливается в самое начало файла, при чтении смещается на позицию, следующую за прочитанными данными, а при записи — на следующую свободную позицию.

Если нужно повторить чтение с начала файла, нужно закрыть его, а потом снова открыть. В школьном алгоритмическом языке вместо этого используется команда `начать чтение`:

```
начать чтение(Fin)
```

Когда файловый курсор указывает на конец файла, логическая функция конец файла возвращает значение «истина»:

```
если конец файла (Fin)
  вывод 'Данные кончились'
все
```

В языке Паскаль для работы с текстовыми файлами используются файловые переменные типа Text:

```
var Fin, Fout: Text;
```

Сначала их нужно с помощью процедуры Assign связать с именами файлов, к которым будем обращаться:

```
Assign(Fin, 'input.txt');
Assign(Fout, 'output.txt');
```

Эти файлы ещё не открыты, и работать с ними нельзя. Для открытия файлов в разных режимах существуют стандартные процедуры

- Reset — открыть на чтение,
- Rewrite — открыть на запись,
- Append — открыть на добавление.

У них один параметр — файловая переменная, которая была предварительно связана с именем файла. После окончания работы файлы закрываются командой Close:

```
Reset(Fin);
Rewrite(Fout);
  {здесь работаем с файлами}
Close(Fin);
Close(Fout);
```

Для чтения и записи используются процедуры read, readln, write и writeln, в которых в качестве первого параметра указывают файловую переменную:

```
readln(Fin, a, b);
writeln(Fout, a, '+', b, '=', a+b);
```

Напомним, что readln, в отличие от read, читает всё до конца строки. В приведённом выше примере после прочтения значений переменных *a* и *b* все оставшиеся символы до конца строки игнорируются, а следующий вызов read или readln берёт данные уже с новой строки.

Функция Eof (англ. **EOF** — *end of file* — конец файла) возвращает значение True (истина), если указатель стоит на последней позиции файла:

```
if Eof(Fin) then
  write('Данные кончились');
```

Неизвестное количество данных

Предположим, что в текстовом файле записано в столбик неизвестное количество чисел и требуется найти их сумму. В этой задаче не нужно одновременно хранить все числа в памяти (и не нужно выделять массив!), достаточно читать по одному числу:

```
нц пока не конец файла
  прочитать число из файла
  добавить его к сумме
кц
```

Далее будем считать, что файловая переменная *Fin* связана с файлом, открытым на чтение. Основная часть программы (без объявления переменных, открытия и закрытия файлов) выглядит так:

```
S:=0
нц пока не конец файла(Fin)
  ввод Fin, x
  S:=S+x
кц
```

```
S:=0;
while not Eof(Fin) do begin
  readln(Fin, x);
  S:=S+x
end;
```

Обработка массивов

В текстовом файле записано не более 100 целых чисел. Требуется вывести в другой текстовый файл те же числа, отсортированные в порядке возрастания.

Особенность этой задачи в том, что для сортировки нам нужно удерживать в памяти все числа, т. е. для их хранения нужно вы-

делить массив. Косвенно на это указывает ограничение — чисел не более 100. Поэтому массив должен иметь не менее 100 элементов:

```
цел MAX=100
целтаб A[1:MAX]
```

```
const MAX=100;
var A: array[1..MAX] of integer;
```

Основная интрига состоит в том, что количество чисел точно не известно. Следовательно, нам нужно считывать значения и записывать их последовательно в первые ячейки массива, пока данные не закончатся — тогда цикл чтения останавливается. Кроме того, нужно сделать защиту от слишком большого количества данных: если 100 чисел уже записаны в массив, цикл должен остановиться. Ниже приведены циклы чтения на школьном алгоритмическом языке:

```
N:=0;
нц пока не конец файла(Fin) и N<MAX
  N:=N+1
  ввод Fin, A[N]
кц
```

и на Паскале:

```
while (not eof(Fin)) and (N<MAX) do begin
  N:=N+1;
  readln(Fin, A[N]);
end;
```

Целая переменная N служит счётчиком прочитанных из файла чисел.

Теперь нужно отсортировать первые N значений массива A (этот код вы уже можете написать самостоятельно) и вывести их во второй файл, открытый на запись:

```
Fout:= открыть на запись('output.txt')
нц для i от 1 до N
  вывод Fout, A[i], нс
кц
закрыть(Fout)
```

Вариант на Паскале:

```
Assign(Fout, 'output.txt');
Rewrite(Fout);
for i:=1 to N do
  writeln(Fout, A[i]);
Close(Fout);
```

Обработка строк

Как известно, современные компьютеры бóльшую часть времени заняты обработкой символьной, а не числовой информации. Предположим, что в текстовом файле записаны данные о собаках, привезённых на выставку: в каждой строчке кличка собаки, её возраст (целое число) и порода, например,

```
Мухтар 4 немецкая овчарка
```

Элементы отделяются друг от друга одним пробелом. Нужно вывести в другой файл сведения о собаках, которым меньше 5 лет.

В этой задаче данные можно обрабатывать по одной строке (не нужно загружать все строки в оперативную память):

```
нц пока не конец файла (Fin)
  прочитать строку из файла Fin
  разобрать строку – выделить возраст
  если возраст<5 то
    записать строку в файл Fout
  все
кц
```

Здесь, как и раньше, *Fin* и *Fout* — файловые переменные, связанные с файлами, открытыми на чтение и запись соответственно.

Будем считать, что все данные корректны, т. е. первый пробел отделяет кличку от возраста, а второй — возраст от породы. Тогда разбор строки можно выполнить так:

```
позиция в строке пробел
удалить из строки кличку с первым пробелом
позиция в строке пробел
выделить возраст перед пробелом
преобразовать возраст в числовой вид
```

Для этого используются стандартные функции для работы с символьными строками (см. § 66):

```
лит s, sAge
цел age, p
лог OK
...
p:=позиция(' ', s)
s:=s[p+1:длин(s)]
p:=позиция(' ',s)
sAge:=s[1:p-1]
age:=лит_в_цел(sAge, OK)
```

```
var s, sAge: string;
 age, p, r: integer;
...
p:=Pos(' ', s);
Delete(s, 1, p);
p:=Pos(' ', s);
sAge:=Copy(s, 1, p-1);
Val(sAge, age, r);
```

В результате работы этого фрагмента возраст собаки оказывается в переменной age. Конечно, при этом исходная строка «портится», поэтому все операции нужно выполнять с её копией. Основной цикл будет выглядеть так:

```
лит s, s0
нц пока не конец файла(Fin)
  ввод Fin, s0
  s:=s0
    | обработка строки s
  если age<5 то
    вывод Fout, s0, нс
  все
кц
```

```
var s, s0: string;
...
while not Eof(Fin) do begin
  readln(Fin, s0);
  s:=s0;
    {обработка строки s}
  if age<5 then
    writeln(Fout, s0);
end;
```

Вопросы и задания

1. Чем различаются текстовые и двоичные файлы по внутреннему содержанию? Можно ли сказать, что текстовый файл — это частный случай двоичного файла?
2. Объясните «принцип сэндвича» при работе с файлами.
3. Как вы думаете, почему открытый программой файл, как правило, блокируется и другие программы не могут получить к нему доступ?
4. Почему рекомендуется вручную закрывать файлы, хотя при закрытии программы они закроются автоматически? В каких ситуациях это может быть важно?
5. Что такое файловая переменная? Почему для работы с файлом используют не имя файла, а файловую переменную?

6. В каком случае одна и та же файловая переменная может быть использована для работы с несколькими файлами, а в каком — нет?
7. Что такое последовательный доступ к данным?
8. Как можно начать чтение данных из файла с его начала?
9. Как определить, что данные в файле закончились?
10. В каких случаях нужно знать максимальное количество данных в файле, а в каких — нет?
11. В каких случаях нужно открывать одновременно несколько файлов?

Подготовьте сообщение

а) «Работа с файлами в языке Си»
б) «Работа с файлами в языке Python»

Задачи

1. Напишите программу, которая находит среднее арифметическое всех чисел, записанных в файле в столбик, и выводит результат в другой файл.
2. Напишите программу, которая находит минимальное и максимальное среди чётных положительных чисел, записанных в файле, и выводит результат в другой файл. Учтите, что таких чисел может вообще не быть.
3. В файле в столбик записаны целые числа. Напишите программу, которая определяет длину самой длинной цепочки идущих подряд одинаковых чисел и выводит результат в другой файл.
4. В файле записано не более 100 чисел. Отсортировать их по возрастанию последней цифры и записать в другой файл.
5. В файле записано не более 100 чисел. Отсортировать их по возрастанию суммы цифр и записать в другой файл.
6. В двух файлах записаны отсортированные по возрастанию массивы неизвестной длины. Объединить их и записать результат в третий файл. Полученный массив также должен быть отсортирован по возрастанию.
7. Дополните решение задачи о собаках, так чтобы программа обрабатывала ошибки в исходных данных. При любых ошибках программа не должна завершаться аварийно.
8. В исходном файле записана речь подростка, в которой часто встречается слово-паразит «короче», например: «Мама, короче, мыла, короче, раму». Убрать из текста все слова-паразиты (должно остаться «Мама мыла раму»).
9. Прочитать текст из файла и подсчитать количество слов в нём.
10. Прочитать текст из файла и вывести в другой файл только те строки, в которых есть слова, начинающиеся с буквы «А».

11. Прочитать текст из файла и вывести в другой файл в столбик все слова, которые начинаются с буквы «А».
12. Прочитать текст из файла, заменить везде слово «паровоз» на слово «пароход» и записать в другой файл.
13. В файле записаны данные о результатах сдачи экзамена. Каждая строка содержит фамилию, имя и количество баллов, разделённые одним пробелом:

    ```
    <Фамилия> <Имя> <Количество баллов>
    ```

    Вывести фамилии и имена тех учеников, которые получили больше 80 баллов.
14. В задаче 13 добавить к списку нумерацию, например:

    ```
    1) Иванов Вася
    2) Петров Петя
    ```

15. В задаче 14 сократить имя до одной буквы и поставить перед фамилией:

    ```
    1) В. Иванов
    2) П. Петров
    ```

16. В задаче 15 отсортировать список по алфавиту (по фамилии).

*17. В задаче 15 отсортировать список по убыванию полученного балла (вывести балл в выходной файл).

Практические работы к главе 8

Работа № 25	«Простые вычисления»
Работа № 26	«Ветвления»
Работа № 27	«Сложные условия»
Работа № 28	«Множественный выбор»
Работа № 29	«Задачи на ветвления»
Работа № 30	«Циклы с условием»
Работа № 31	«Циклы с условием»
Работа № 32	«Циклы с переменной»
Работа № 33	«Вложенные циклы»
Работа № 34	«Процедуры»
Работа № 35	«Процедуры с изменяемыми параметрами»
Работа № 36	«Функции»
Работа № 37	«Логические функции»
Работа № 38	«Рекурсия»
Работа № 39	«Стек»
Работа № 40	«Перебор элементов массива»
Работа № 41	«Линейный поиск»
Работа № 42	«Поиск максимального элемента массива»

Работа № 43 «Алгоритмы обработки массивов»
Работа № 44 «Отбор элементов массива по условию»
Работа № 45 «Метод пузырька»
Работа № 46 «Метод выбора»
Работа № 47 «Быстрая сортировка»
Работа № 48 «Двоичный поиск»
Работа № 49 «Посимвольная обработка строк»
Работа № 50 «Функции для работы со строками»
Работа № 51 «Преобразования "строка-число"»
Работа № 52 «Строки в процедурах и функциях»
Работа № 53 «Рекурсивный перебор»
Работа № 54 «Сравнение и сортировка строк»
Работа № 55 «Обработка символьных строк: сложные задачи»
Работа № 56 «Матрицы»
Работа № 57 «Обработка блоков матрицы»
Работа № 58 «Файловый ввод и вывод»
Работа № 59 «Обработка массивов из файла»
Работа № 60 «Обработка строк из файла»
Работа № 61 «Обработка смешанных данных из файла»

ЭОР к главе 8 на сайте ФЦИОР (http//fcior.edu.ru)

- Понятие алгоритма, его свойства. Линейный алгоритм
- Основные структуры данных
- Классификация языков программирования. Компиляторы и интерпретаторы
- Реализация основных алгоритмических конструкций
- Алгоритмы сортировки
- Алгоритмы поиска
- Организация работы с текстовыми файлами

Самое важное в главе 8

- Алгоритмы могут записываться на псевдокоде, в виде блок-схем и на языках программирования. Алгоритм, записанный на языке программирования, называется программой.
- Данные, с которыми работает программа, хранятся в переменных. Переменная — это величина, которая имеет имя, тип и значение. Значение переменной может изменяться во время выполнения программы.

- Любой алгоритм можно записать, используя последовательное выполнение операторов, ветвления и циклы. Ветвления предназначены для выбора вариантов действий в зависимости от выполнения некоторых условий. Цикл — это многократное повторение одинаковых действий.
- Подпрограммы — это вспомогательные алгоритмы, которые могут многократно вызываться из основной программы и других подпрограмм. Подпрограммы-процедуры выполняют описанные в них действия, а подпрограммы-функции в дополнение к этому возвращают результат этих действий (число, символ, логическое значение и т. д.). Дополнительные данные, передаваемые в подпрограмму, называют аргументами. В подпрограмме эти данные представлены как локальные переменные, которые называются параметрами подпрограммы.
- Рекурсивные алгоритмы основаны на последовательном сведении исходной задачи ко всё более простым задачам такого же типа (с другими значениями параметров). Рекурсия служит заменой циклу. Любой рекурсивный алгоритм можно записать без рекурсии, но во многих случаях такая запись более длинная и менее понятная.
- Массив — это группа переменных одного типа, расположенных в памяти рядом (в соседних ячейках) и имеющих общее имя. Каждая ячейка массива имеет уникальный индекс (как правило, это номер элемента).
- Сортировка — это расстановка элементов массива в заданном порядке. Цель сортировки — облегчить последующий поиск. Для отсортированного массива можно применить двоичный поиск, который работает значительно быстрее, чем линейный.
- Символьная строка — это последовательность символов, расположенных в памяти рядом (в соседних ячейках). Строка представляет собой единый объект, она может менять свою длину во время работы программы.
- Матрица — это прямоугольная таблица, составленная из элементов одного типа (чисел, строк и т. д.). Каждый элемент матрицы имеет два индекса — номера строки и столбца.
- До выполнения операций с файлом нужно открыть файл (сделать его активным), а после завершения всех действий — закрыть (освободить).

Глава 9
Решение вычислительных задач на компьютере

§ 69
Точность вычислений

«Недостатки математического образования с наибольшей отчётливостью проявляются в чрезмерной точности численных расчётов», — писал выдающийся немецкий математик первой половины XIX века Карл Фридрих Гаусс.

Все практические расчёты выполняются неточно, с некоторой погрешностью (ошибкой, отклонением от истинного значения). В первую очередь это связано с тем, что неточно известны исходные данные, которые получаются в результате измерений.

Погрешности измерений

Окружающие нас предметы имеют различные числовые характеристики (длину, массу, объём и др.), которые часто приходится измерять для решения практических задач. Для измерений используются приборы, каждый из которых имеет определённую точность. Это значит, что с помощью данного прибора невозможно зафиксировать изменение величины, меньшее, чем *цена деления шкалы* этого прибора. Поэтому измеренное значение величины всегда отличается от точного (истинного), разность между ними называют **погрешностью измерения**.

Пусть нужно найти толщину дна кружки, используя только линейку с ценой деления 1 мм. Линейкой можно измерить высоту кружки снаружи и глубину внутри (рис. 9.1). При этом точность измерений будет не выше чем 1 мм (0,1 см). Например, если измеренная высота кружки оказалась примерно 8,2 см, на самом деле она может быть от 8,1 до 8,3 см. Если измеренная глубина равна 7,8 см, фактическая может быть от 7,7 до 7,9 см.

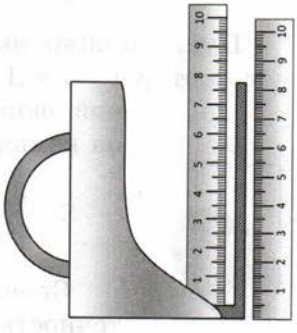


Рис. 9.1

Используя данные измерений, можно найти толщину дна как разность

$$8{,}2 - 7{,}8 = 0{,}4 \text{ см}.$$

Это не означает, что толщина дна действительно такая. Действительно, с учётом ошибок измерений она может быть равна как 8,1 – 7,9 = 0,2 см, так и 8,3 – 7,7 = 0,6 см. Таким образом, реальная толщина может быть от 0,2 до 0,6 см (разница в 3 раза!), и в полученном ответе (0,4 см) нет ни одной верной значащей цифры! Обычно в этом случае пишут ответ в виде 0,4 ± 0,2 см.

В приведённом примере погрешность 0,2 см — это так называемая **абсолютная погрешность** Δx. Для оценки качества измерений чаще используют **относительную погрешность**, которая вычисляется как отношение абсолютной погрешности Δx к истинному значению величины x^*:

$$\delta_x = \frac{\Delta x}{x^*} \cdot 100\%.$$

Поскольку истинное значение, как правило, неизвестно, его обычно заменяют на полученный результат изменений. В данном случае относительную погрешность можно оценить как

$$\delta_x = \frac{0{,}2}{0{,}4} \cdot 100\% = 50\%.$$

Это очень большое значение, которое говорит о низкой точности измерений.

Погрешности вычислений

Пусть нужно вычислить площадь сечения цилиндра, диаметр которого D = 1,2 см известен с точностью 0,1 см. По известной формуле площади круга получаем (например, на калькуляторе):

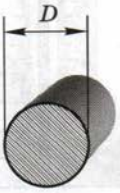


Рис. 9.2

$$S = \frac{\pi \cdot D^2}{4} = 1{,}130973355923255658465516179806\ldots$$

Значит ли это, что мы нашли площадь с такой точностью? Конечно, нет. Вспомним, что диаметр был измерен с точностью 0,1 см, т. е. он мог быть в самом

деле равен как 1,1 см, так и 1,3 см. В этих «крайних» случаях получаем площадь

$$S_{\min} = \frac{\pi \cdot 1{,}1^2}{4} = 0{,}950\ldots \text{ и } S_{\max} = \frac{\pi \cdot 1{,}3^2}{4} = 1{,}327\ldots$$

Таким образом, следует записать ответ в виде $S \approx 1{,}1 \pm 0{,}2$ см^2. Относительную погрешность результата можно оценить как

$$\delta_x = \frac{0{,}2}{1{,}1} \cdot 100\% \approx 18\%.$$

Заметим, что мы не учитывали погрешность, связанную с неточностью задания иррационального числа π.

Все практические расчёты выполняются неточно. Погрешность результата вычислений определяется в первую очередь погрешностью исходных данных.

Теперь вернёмся к расчётам с помощью компьютера. Как вы знаете из главы 4, данные записываются в память в двоичном коде ограниченной длины, при этом практически все вещественные числа хранятся неточно. При выполнении вычислений погрешности накапливаются, поэтому при сложных расчётах может получиться неверный ответ. Например, с точки зрения точности очень плохо, если ответ — это небольшое (по модулю) число, которое вычисляется как разность двух неточных больших чисел (вспомните пример с кружкой!).

Погрешность резко возрастает при делении на неточное малое по модулю число. Предположим, что нужно вычислить значение

$$x = \frac{a}{b} - \frac{c}{d},$$

причём a, b, c и d — вещественные числа, которые получены в результате вычислений с погрешностью 0,001:

$$a = 1000 \pm 0{,}001; \quad b = 0{,}002 + 0{,}001; \quad c = 1000 \pm 0{,}001; \quad d = 0{,}003 + 0{,}001.$$

Легко проверить, что вычисленное значение x может находиться в интервале от −166 667 до 750 001, т. е. относительная

погрешность превышает 300%! Такой метод расчётов **вычислительно неустойчив**: малые погрешности в исходных данных могут привести к огромным погрешностям в решении.

Подводя итог, можно выделить несколько источников погрешностей при компьютерных вычислениях:

- неточность исходных данных;
- неточность записи вещественных чисел в двоичном коде конечной длины;
- погрешности приближённого вычисления некоторых стандартных функций (например, $\sin(x)$ или $\cos(x)$);
- накопление погрешностей при арифметических действиях с неточными данными;
- собственная погрешность используемого метода (для приближённых методов, рассматриваемых в следующем параграфе).

Проблемы, возникающие при вычислениях с конечной точностью, изучает **вычислительная математика**, задача которой — разработать **вычислительно устойчивые методы** решения задач, при которых небольшие погрешности исходных данных мало влияют на результат. Иногда этого удаётся добиться простым изменением порядка действий или преобразованием формул.

Вопросы и задания

1. Как вы понимаете приведённое в параграфе высказывание К. Ф. Гаусса?
2. Какие величины можно измерять? Какие приборы для этого используются? Приведите примеры.
3. Какова цена деления у ваших наручных часов?
4. Как определить цену деления для приборов с цифровыми индикаторами? Приведите примеры.
5. Что такое абсолютная и относительная погрешности? Какое из этих значений более важно в практических задачах?
6. Что такое вычислительно неустойчивый метод?
7. Перечислите источники погрешностей при компьютерных вычислениях.
8. Какие задачи изучает вычислительная математика?

Подготовьте сообщение

а) «Абсолютная и относительная погрешность»

б) «Вычислительная устойчивость методов»

Задачи

1. Как изменятся абсолютная и относительная погрешности результата, если при вычислении площади сечения (см. задачу в тексте параграфа) в качестве π использовать число 3,14?
2. Радиус шарика R измерили с точностью 0,1 см и получили 1,2 см. Затем рассчитали его объём по формуле
$$V = \frac{4 \cdot \pi \cdot R^3}{3} = 7{,}23822947387088...$$
Запишите ответ в этой задаче, оставив нужно количество значащих цифр.
3. С помощью рулетки размеры бруса (220 см × 11 см × 10 см) измерили с точностью 1 см. Определите его объём. Запишите ответ с учётом точности полученного результата.
*4. Пётр и Павел измеряют плотность меди. У них есть медные бруски разной величины, линейка и весы. Пётр взял брусок с размерами (по результатам измерений) 2 × 2 × 2 см, а Павел — с размерами 5 × 5 × 5 см. При измерении массы брусков у Петра получилось 70 г, а у Павла — 1120 г. Погрешность измерения длины — 1 мм, погрешность измерения массы — 10 г. С какой абсолютной и относительной погрешностью определили плотность меди Пётр и Павел? Какой брусок нужно было выбирать, чтобы погрешность была наименьшей?

§ 70
Решение уравнений

Приближённые методы

На уроках математики вас учили искать решение уравнения в виде формулы, выражающей неизвестную величину через известные. Например, решение уравнения $ax + b = 1$ при $a \neq 0$ можно записать в виде $x = (1 - b)/a$. Такое решение называется **аналитическим**, оно может быть использовано для теоретического исследования (изучения влияния исходных данных на результат).

Однако не все уравнения можно (на современном уровне развития математики) решить аналитически. Иногда решение есть, но очень сложное. Например, уравнение $x = \cos x$ так просто не решается. В этом случае приходится использовать другие методы решения, например графический: построить по точкам графики функций, стоящих в левой и правой частях равенства, и посмотреть, где они пересекаются (рис. 9.3). Затем решение можно уточнять,

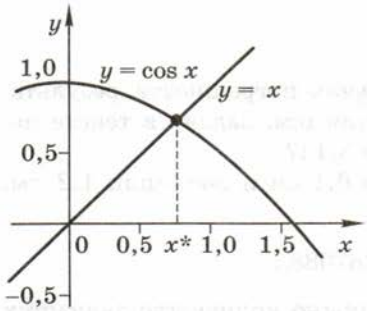


Рис. 9.3

уменьшая шаг при построении графика до получения требуемой точности.

Если нужна высокая точность, графический метод требует очень большого объёма вычислений, который имеет смысл поручить компьютеру. Однако нужно как-то учесть, что компьютер не способен «посмотреть» на график, и для уточнения решения может использовать только числовые данные. Компьютерный алгоритм решения уравнения может выглядеть примерно так:

1) выбрать отрезок $[a_0, b_0]$ для поиска решения (обычно предполагается, что на этом отрезке решение есть, и притом только одно);
2) с помощью некоторого алгоритма уточнить решение, перейдя к меньшему отрезку $[a, b]$;
3) повторять шаг 2, пока длина отрезка (содержащего решение) не станет достаточно мала.

Здесь не совсем ясно, что значит «пока длина отрезка не станет достаточно мала». Обычно задается нужная точность ε: это означает, что отклонение полученного решения от истинного x^* не должно быть больше ε. Если корень уравнения находится на отрезке $[a, b]$, то в качестве решения лучше всего взять его середину $(a+b)/2$. В этом случае погрешность будет минимальной: не больше половины длины отрезка. Поэтому цикл нужно остановить, когда длина отрезка станет меньше, чем 2ε.

Часто используется другой вариант, когда отрезок не нужен, а требуется знать только одну точку вблизи решения:

1) выбрать **начальное приближение** x_0 около решения;
2) с помощью некоторого алгоритма перейти к следующему приближению x, которое находится ближе к точному решению x^*;
3) повторять шаг 2, пока на очередном шаге решение не изменится на величину, меньшую, чем допустимая погрешность ε.

Подобные методы решения уравнений называются **приближёнными**. Их суть в том, что решение последовательно уточняется до тех пор, пока не будет найдено с требуемой точностью. Поскольку при каждом уточнении выполняются одинаковые действия, можно назвать такие методы **итерационными** (от *лат.* iteration — повторение).

Приближённые методы имеют ряд **недостатков**:

- получается приближённое решение, а не точное; это значит, что нельзя написать $x^* = 1{,}2345$, нужно использовать знак приближённого равенства: $x^* \approx 1{,}2345$ (отметим ещё раз, что практически все вычисления с дробными числами на компьютере выполняются неточно);
- мы получаем не формулу, а число, по которому невозможно оценить, как меняется решение при изменении исходных данных (сложно выявить зависимость от параметра);
- объём вычислений может быть слишком велик, часто это не позволяет использовать приближённые методы в системах реального времени;
- не всегда можно оценить погрешность результата.

Однако в некоторых практических случаях приближённые методы более полезны, чем аналитические. Они обладают следующими **достоинствами**:

- часто получение аналитического решения невозможно или требует значительных усилий, тогда как приближенные методы позволяют достаточно быстро решить задачу с заданной точностью;
- при компьютерных расчётах (с конечной точностью) вычисления по «точным» аналитическим формулам часто могут давать очень неточный результат из-за вычислительной неустойчивости метода. Нередко для таких задач (например, для решения систем линейных уравнений) специально разрабатываются приближённые методы, которые дают значительно более точное решение.

Метод перебора

Как принято в вычислительной математике, далее мы будем рассматривать уравнение общего вида $f(x) = 0$, к которому можно привести любое заданное уравнение. Например, для уравнения $x = \cos x$ получаем $f(x) = x - \cos x$.

Предположим, что нужно найти решение уравнения $f(x) = 0$ с точностью ε, причём известно (например, видно на графике), что решение находится справа от некоторой точки $x = a$. В этом случае можно разбить всю область, где может быть решение, на узкие полоски шириной $\delta = 2\varepsilon$, и выбрать такую полоску, где график функции пересекает ось OX (рис. 9.4).

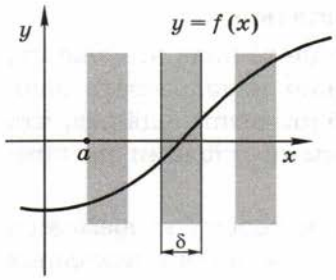


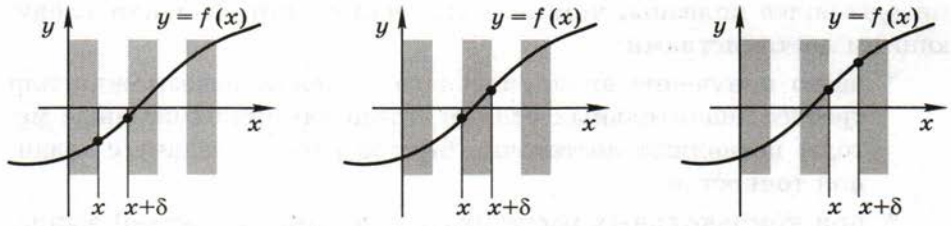
Рис. 9.4

Для того чтобы поручить решение этой задачи компьютеру, нужно ответить на два вопроса:

1) Как с помощью математических операций определить, что в полосе $[x, x+\delta]$ есть решение?
2) Что считать решением уравнения, когда такая полоса определена?

Проще ответить на второй вопрос: лучше всего взять в качестве решения середину полосы, т. е. точку $x_* = x + \varepsilon$ (в этом случае погрешность будет не больше, чем ε).

Для того чтобы определить, есть ли решение на отрезке $[x, x+\delta]$, сравним значения функции на концах этого отрезка. Рассмотрим три случая, показанные на рис. 9.5, *а–в*.



а *б* *в*

Рис. 9.5

Несложно понять, что если график пересекает ось OX, то на концах отрезка функция имеет разные знаки, т. е. $f(x) \cdot f(x+\delta) \le 0$. При этом важно, чтобы график не имел разрывов.

Если непрерывная функция имеет разные знаки на концах отрезка $[x_1, x_2]$, то в некоторой точке внутри этого отрезка она равна нулю.

Таким образом, нужно найти отрезок шириной 2ε, на концах которого функция имеет разные знаки, и взять в качестве решения его середину. Решение этой задачи при $a = 0$ может выглядеть, например, так (слева приведена программа на школьном алгоритмическом языке, а справа — на языке Паскаль):

```
алг Перебор
нач
 вещ eps, x, delta
 eps:=0.001
 x:=0
 delta:=2*eps
 нц пока f(x)*f(x+delta)>0
  x:=x+delta
 кц
 вывод 'x = ', x+eps
кон
алг вещ f(вещ x)
нач
 знач:=x-cos(x)
кон
```

```
const eps=0.001;
var x, delta: real;
function f(x: real):real;
begin
 f:=x-cos(x);
end;
begin
 x:=0;
 delta:=2*eps;
 while f(x)*f(x+delta)>0 do
  x:=x+delta;
 writeln('x=',(x+eps):6:3);
end.
```

Здесь заданная точность ε хранится в виде константы *eps*, а вычисление функции $f(x)$ оформлено в виде подпрограммы-функции `f`. Поиск решения начинается с нуля (в других задачах начальное значение может быть другим). Цикл останавливается, когда для очередного отрезка получаем $f(x) \cdot f(x+\delta) \le 0$.

Нужно понимать, что при вычислениях на реальном компьютере мы не можем задавать произвольно малое значение ε. Точность ограничена типом данных, с помощью которого выполняются вычисления. В практических расчётах чаще всего используются данные с двойной точностью (англ. *double*), для которых предельная точность близка к 10^{-16} (вспомните материал § 26 и 29).

Обратите внимание, что в этом простейшем варианте программа зациклится, если справа от нуля решения нет. Подумайте, как изменить программу так, чтобы зацикливания не было.

Кроме того, в приведённом алгоритме есть и ещё одна возможная неточность: подумайте, что случится, если случайно получится $f(x) = 0$ или $f(x+\delta) = 0$. Поскольку условие цикла при этом нарушается, цикл закончится, и будет получен результат с требуемой точностью ε. Однако в этом случае можно было бы определить решение более точно, что вам и предлагается сделать самостоятельно.

Главный недостаток метода перебора — большое количество операций. Например, для решения уравнения с точностью 0,001 может понадобиться несколько тысяч вычислений значения функции $f(x)$. Поэтому сначала можно использовать перебор с достаточно большим шагом, чтобы *отделить корни*, т. е. найти ин-

тервалы, в каждом из которых есть только один корень. После этого выполняется *уточнение корней* — перебор внутри каждого из таких интервалов с шагом $\delta = 2\varepsilon$, достаточным для определения решения с заданной точностью.

Метод деления отрезка пополам

Есть старая задача-шутка: «Как поймать льва в Африке?» Предлагается перегородить забором всю Африку, разбив её на две равные части, и ждать, где появится лев. Затем часть, в которой есть лев, разделить забором на две равные области и т. д. В конце концов, лев окажется в маленькой клетке. На самом деле эта шутка иллюстрирует **метод деления отрезка пополам** (метод **бисекции**), с помощью которого можно найти решение уравнения на некотором интервале (если оно там есть).

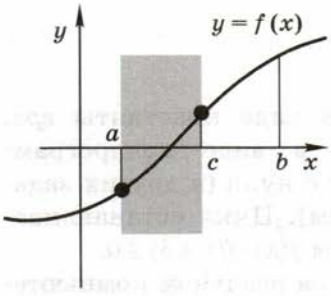


Рис. 9.6

Пусть некоторая функция $y = f(x)$ непрерывна на отрезке $[a, b]$ и имеет разные знаки в точках $x = a$ и $x = b$ (рис. 9.6). Тогда в одной из промежуточных точек $x = x^*$ она обращается в ноль, т. е. уравнение $f(x) = 0$ имеет решение. Найти его можно так:

1) вычислить середину отрезка $c = \dfrac{a+b}{2}$;
2) если на отрезке $[a, c]$ есть решение, присвоить $b := c$, иначе присвоить $a := c$;
3) повторять шаги 1–2 до тех пор, пока $b - a > 2\varepsilon$.

В пункте 2 нам нужно ответить на вопрос, если решение на отрезке $[a, c]$. Мы уже умеем это делать: решение есть, если $f(a) \cdot f(c) \le 0$.

В простейшем варианте цикл, уточняющий решение, можно записать в виде:

```
delta:=2*eps
нц пока b-a>delta
  c:=(a+b)/2
  если f(a)*f(c)<0 то
     b:=c
   иначе
     a:=c
```

```
delta:=2*eps;
while b-a>delta do begin
  c:=(a+b)/2;
  if f(a)*f(c)<0 then
     b:=c
  else a:=c;
end;
```

```
  все                    writeln('x = ', (a+b)/2:6:3);
кц
вывод 'x = ', (a+b)/2
```

Однако в приведённом алгоритме есть одна проблема, которая проявляется в том случае, когда на очередном шаге $f(a) = 0$ или $f(c) = 0$. Предположим, что рассматривается функция $f(x) = x - 1$, и мы выбрали для поиска отрезок [0, 2]. Тогда на первом же шаге получаем $c = 1$, так что $f(c) = 0$; это приводит к выполнению присваивания $a{:=}c$. Поэтому на втором шаге получаем $f(a) = 0$, а на всех последующих шагах — $f(a) \cdot f(c) > 0$, что, в конечном счёте, даёт неверный ответ $x = 2$.

Чтобы решить эту проблему, вернёмся к формулировке «на отрезке $[a, b]$ есть решение, если функция $f(x)$ имеет разные знаки на концах этого отрезка». В школьном алгоритмическом языке есть функция `sign`, которая возвращает знак числа, т. е. вычисляет функцию:

$$sign(x) = \begin{cases} -1, & \text{если } x < 0, \\ 0, & \text{если } x = 0, \\ 1, & \text{если } x > 0. \end{cases}$$

Тогда условный оператор в теле цикла можно заменить так:

```
если sign(f(a))<>sign(f(c))      if sign(f(a))<>sign(f(c))
  то    b:= c                      then b:= c
  иначе a:= c все                  else a:= c;
```

Теперь программа даёт правильное решение даже для рассмотренных выше особых случаев (проверьте это самостоятельно). К сожалению, во многих версиях Паскаля[1] функции `sign` нет, но вы легко можете её написать.

На каждом шаге этого цикла ширина отрезка уменьшается в 2 раза, за n шагов она уменьшится в 2^n раз. Таким образом, при выборе единичного начального отрезка для получения решения с точностью 0,001 достаточно 10 шагов цикла.

Метод деления отрезка пополам очень прост и надёжен, позволяет найти решение с заданной точностью (в пределах точности машинных вычислений). Однако для его применения нужно заранее *отделить корни уравнения*, т. е. найти отрезки, каждый из которых содержит только один корень. Для отделения корней

[1] В среде FreePascal функция `sign` входит в модуль `Math`, который нужно подключить командой `uses Math;`

можно использовать построенный график функции или метод перебора с некоторым шагом. Таким образом, решение уравнения проводится в два этапа — отделение корней и уточнение корней.

К сожалению, метод деления отрезка пополам неприменим для решения систем уравнений с несколькими неизвестными.

Пример: полёт мяча

Для иллюстрации рассмотрим такую задачу: Вася бросает мяч со скоростью 12 м/с. Под каким углом к горизонту Васе нужно бросить мяч, чтобы попасть в мишень на высоте 4 м на расстоянии 10 м? В момент броска мяч находится на высоте 2 м (рис. 9.7).

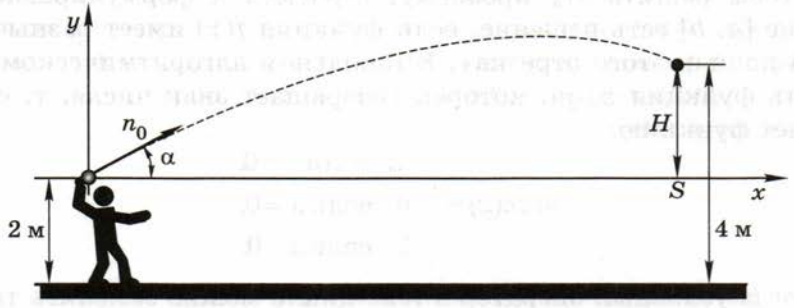


Рис. 9.7

Примем за начало координат 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). Будем считать мяч материальной точкой; поскольку его скорость невысока, сопротивлением воздуха можно пренебречь. Известные из физики уравнения движения запишутся в виде:

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

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 $y = H$:

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

Время t можно сразу выразить из первого уравнения:

$$t = \frac{S}{v_0 \cos\alpha}.$$

Подставляя этот результат во второе уравнение, получаем уравнение с одним неизвестным α, которое можно привести к стандартному виду $f(\alpha)=0$, где:

$$f(\alpha)=S\cdot tg\alpha-\frac{g\cdot S^2}{2v_0^2\cos^2\alpha}-H=0.$$

Решать его аналитически достаточно сложно[1], поэтому мы найдём приближённое решение. При вычислении тригонометрических функций угол измеряется в радианах, поэтому нужно искать решение в диапазоне углов α от 0 до $\pi/2$.

Мы не знаем, сколько решений имеет уравнение, поэтому изменим метод перебора так, чтобы найти все решения. Цикл while не будет останавливаться на первом найденном решении, а будет продолжаться, пока угол не станет больше $\pi/2$. Если в какой-то полосе есть решение, вычисляем угол в градусах и выводим его на экран. Приведём основные части программ на школьном алгоритмическом языке:

```
pi:=3.1415926
u:=0
delta:=2*eps
нц пока u<pi/2
  если f(u)*f(u+delta)<=0 то
    вывод 'Угол: ', (u+eps)*180/pi, ' градусов', нс
  все
  u:=u+delta
кц
```

и на Паскале:

```
u:=0;
delta:=2*eps;
while u<pi/2 do begin
  if f(u)*f(u+delta)<=0 then begin
    alpha:=(u+eps)*180/pi;
    writeln('Угол: ', alpha:4:1, ' градусов');
  end;
  u:=u+delta;
end;
```

[1] Хотя можно, например, с помощью замены $z=\frac{1}{\cos^2\alpha}$.

В переменной *u* хранится угол в радианах, а в переменной *alpha* — угол в градусах. Если запустить эту программу, мы увидим, что уравнение имеет два решение — углы примерно равны 35,6° и 65,8°.

Попробуйте применить в этой же задаче метод деления отрезка пополам. Подумайте, с какими проблемами мы здесь сталкиваемся, и почему они возникают.

Повторите вычисления для начальных скоростей 10 м/с и 20 м/с и объясните полученные результаты.

Использование табличных процессоров

Для численного решения уравнений можно использовать табличный процессор, например OpenOffice.org Calc (или LibreOffice Calc) или Microsoft Excel. Обычно сначала строится график функции, который позволяет определить количество решений уравнения и их примерное расположение; затем используется модуль «Поиск решения». Далее мы будем рассматривать программу Calc из пакета OpenOffice.org, указывая на незначительные отличия Excel.

Введём исходные данные для рассмотренной задачи бросания мяча, как показано на рис. 9.8. Для того чтобы формулы выглядели более привычно, дадим ячейкам B1, B2 и B3 имена *S*, *H* и *v* (их можно ввести в левом верхнем углу таблицы).

В столбце A заполним ряд значений углов от 0° до 85° с шагом 5°. Для этого введём два первых значения, выделим эти ячейки и «растянем» за маркер заполнения (квадратик в правом нижнем углу выделенной части, рис. 9.9).

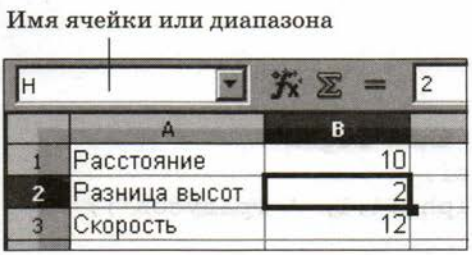


Рис. 9.8

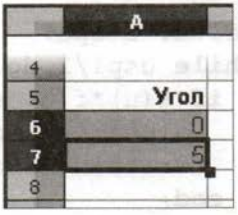


Рис. 9.9

Добавим столбцы, в которых для каждого угла будут вычисляться его значение в радианах (с помощью стандартной функции

RADIANS, в русской версии Excel — РАДИАНЫ), время полёта, координата y и значение функции $f(\alpha)$ (рис. 9.10).

	A	B	C	D	E
1	**Расстояние**	10			
2	**Разница высот**	2			
3	**Скорость**	12			
4					
5	**Угол**	**Угол (рад)**	**Время**	**y**	**f(x)**
6	0	=RADIANS(A6)	=S/v/COS(B6)	=v*SIN(B6)*C6-9,81*C6^2/2	=D6-H
7	5				
8	10				

Рис. 9.10

Обратите внимание, что в формулах мы используем имена ячеек S, H и v. Это абсолютные ссылки, не меняющиеся при копировании; например, вместо имени S можно было бы написать адрес $B$1, но это было бы менее понятно. Эти формулы можно просто «растянуть» (скопировать) вниз за маркер заполнения.

Теперь построим график функции $f(\alpha)$. Сначала нужно выделить данные в столбцах A и E, это можно сделать, если удерживать нажатой клавишу Ctrl. Затем строим диаграмму типа **Диаграмма XY** (в Excel — диаграмма **Точечная**) — рис. 9.11. График функции пересекает ось OX в двух точках, т. е. уравнение $f(\alpha) = 0$ имеет два решения, одно около 35°, второе — около 65°.

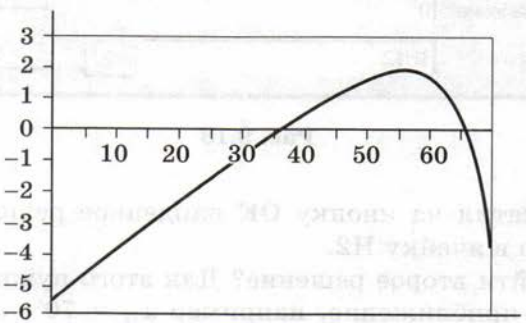


Рис. 9.11

Теперь уточним решение, используя возможности табличного процессора, в котором реализован один из приближённых мето-

дов решения уравнений. Для этого нужно знать **начальное приближение** α_0 — значение неизвестной величины, достаточно близкое к решению. По графику мы определили, что первый раз график пересекает ось *OX* для значения угла около 35°, поэтому можно взять $\alpha_0 = 35°$. Запишем это значение в свободную ячейку, например в H2, и добавим недостающие формулы так, чтобы получить значение функции $f(\alpha)$ в ячейке L2 (рис. 9.12).

	H	I	J	K	L
1	**Угол**	**Угол (рад)**	**Время**	**y**	**f(x)**
2	35				

Рис. 9.12

Задача подбора параметра формулируется так: «*установить в ячейке ... значение ..., изменяя значение ячейки ...*». Например, в нашем случае нужно установить в ячейке L2 значение 0, изменяя H2. Ячейка L2 называется **целевой**, потому что наша *цель* — получить в ней определённое значение (ноль). Ячейка H2 — это **изменяемая ячейка**. В главном меню выбираем пункт **Сервис**, **Подбор параметра** и вводим эти данные (рис. 9.13).



Рис. 9.13

После нажатия на кнопку **ОК** найденное решение уравнения будет записано в ячейку H2.

Как же найти второе решение? Для этого нужно выбрать другое начальное приближение, например $\alpha_0 = 70°$, в остальном порядок действий не меняется. Сделайте это самостоятельно.

Проверьте, что будет происходить при изменении начальной скорости до 10 м/с и до 20 м/с. Попробуйте объяснить эти результаты с точки зрения физики.

Вопросы и задания

1. Какие методы называются приближёнными? В каких случаях они используются?
2. Что такое итерационный метод?
3. Сравните приближённые и аналитические методы решения уравнений. В чём достоинства и недостатки каждого подхода?
4. Объясните, в чём заключается метод перебора. В чём его недостатки?
5. Как с помощью математических операций определяют, есть ли решение уравнения на заданном отрезке? В каких случаях такой подход не сработает?
6. Как избежать зацикливания в методе перебора?
7. Объясните, почему методом перебора при ширине полосы $\delta = 2\varepsilon$ можно найти решение с точностью ε.
8. Что такое отделение корней и уточнение корней?
9. Объясните изменения, сделанные в первоначальной программе для решения уравнения методом перебора, которые позволили в одном цикле найти все решения на заданном отрезке.
10. Объясните, как работает метод деления отрезка пополам. Сравните его с методом перебора.

Задачи

1. Решите уравнение $x^2 = 5\cos(x-1)$ методом перебора и методом деления отрезка пополам. Сравните количество шагов цикла при использовании каждого метода.
2. Решите уравнение $x^2 = 5\cos(x-1)$, используя табличный процессор.

*3. Попытайтесь улучшить программу для метода перебора, используя одно из значений функции, вычисленных на предыдущем шаге цикла.

*4. Попытайтесь улучшить программу для метода деления отрезка пополам, используя одно из значений функции, вычисленных на предыдущем шаге цикла.

5. Экспериментально (пробуя различные значения ε) определите, с какой точностью можно найти решение уравнения $x^2 = 5\cos(x-1)$ в вашей среде программирования.
6. Решите задачу из примера «Полёт мяча» с помощью собственной программы, а затем с помощью табличного процессора. Обсудите преимущества и недостатки этих способов решения.

*7. Используя замену $z = \dfrac{1}{\cos^2 \alpha}$, постройте аналитическое решение уравнения из примера «Полёт мяча». Для практических вычислений используйте электронные таблицы. Сравните точное и численное решения.

§ 71 Дискретизация

Вычисление длины кривой

Определим длину траектории L, по которой летит шарик, брошенный под углом к горизонту (см. задачу в § 70). Это не так просто, потому что траектория — кривая линия.

Постараемся как-то свести задачу к более простой, которую мы умеем решать. Если бы линия состояла из отдельных отрезков, её длину можно было бы точно вычислить с помощью теоремы Пифагора. Например, длина ломаной на рис. 9.14 равна:

$$L = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} + \sqrt{(x_3 - x_2)^2 + (y_3 - y_2)^2}.$$

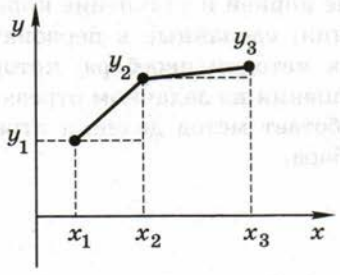


Рис. 9.14

Используя эту идею, разделим кривую линию на небольшие участки и заменим каждый участок отрезком так, чтобы получилась ломаная (штриховая линия на рис. 9.15).

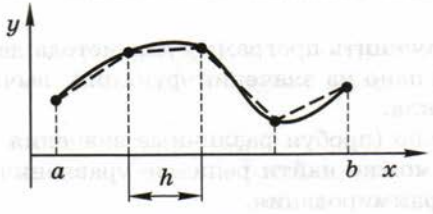


Рис. 9.15

Обычно разбивают исходный отрезок $[a, b]$ (на котором нужно определить длину кривой) на равные участки длины h. Конечно, длина ломаной отличается от длины кривой, но естественно предположить, что при уменьшении шага разбиения h эта разница будет уменьшаться.

Обратим внимание, что мы фактически выполнили **дискретизацию** — представили кривую в виде набора точек, при этом информация о поведении функции между этими точками была потеряна (вспомните оцифровку звука!). Величина h называется **шагом дискретизации**.

Подведём итоги:

- дискретизация позволяет представить задачу в виде, пригодном для компьютерных расчётов;
- при дискретизации часть информации теряется, поэтому все методы, основанные на дискретизации, — приближённые, они решают задачу с некоторой погрешностью;
- чтобы уменьшить погрешность, нужно уменьшать шаг дискретизации (увеличивать количество точек), но при этом возрастёт объём (и время) расчётов;
- при выборе малого шага дискретизации на результат могут сильно влиять погрешности вычислений, вызванные неточностью представления вещественных чисел в памяти компьютера (см. § 29).

Теперь можно составить программу. Будем считать, что константы (или переменные) a, b и h задают границы отрезка и шаг дискретизации. Тогда основная часть программы может выглядеть так на школьном алгоритмическом языке:

```
x:=a; L:=0
нц пока x<b
  y1:=f(x)
  y2:=f(x+h)
  L:=L+sqrt(h*h+(y1-y2)*(y1-y2))
  x:=x+h;
кц
вывод 'Длина кривой ', L
```

и на Паскале:

```
x:=a; L:=0;
while x<b do begin
  y1:=f(x);
  y2:=f(x+h);
  L:=L+sqrt(h*h+(y2-y1)*(y2-y1));
  x:=x+h
end;
writeln('Длина кривой ', L:10:3);
```

Возвращаясь к задаче с шариком, вспомним, что его движение описывается уравнениями:

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

Если выразить время из первого уравнения и подставить во второе, получается

$$y = f(\alpha) = x \cdot tg\alpha - \frac{g \cdot x^2}{2v_0^2 \cos^2\alpha}.$$

Это и есть функция, график которой нас интересует. Написать полную программу вы уже можете самостоятельно. Проверьте её работу для разных значений исходных данных (скорости и угла вылета, расстояния).

Вычисление площадей фигур

Применим метод дискретизации в другой достаточно сложной задаче — для вычисления площади фигуры. Пусть фигура, площадь которой нас интересует, ограничена графиками функций $y = f_1(x)$ и $y = f_2(x)$ (рис. 9.16).

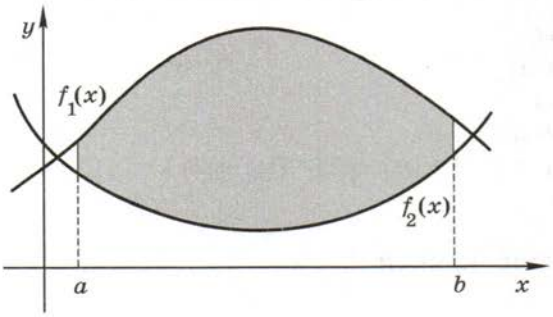


Рис. 9.16

В некоторых простых случаях (но далеко не всегда!) площадь такой фигуры можно вычислить аналитически с помощью методов высшей математики. Мы же будем использовать приближённые методы, применять которые значительно проще.

Как и при вычислении длины кривой, применим *дискретизацию* — разделим фигуру на отдельные полоски и заменим каж-

дую полоску на другую фигуру, для которой мы можем легко найти площадь, например на *прямоугольник* (рис. 9.17).

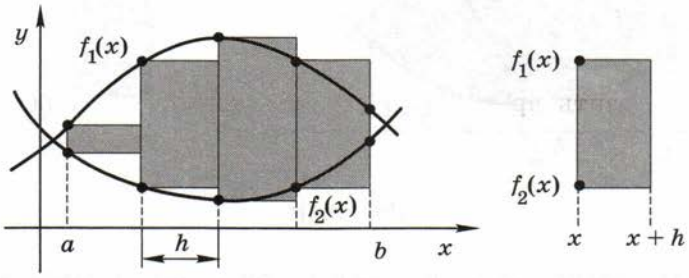


Рис. 9.17

Понятно, что площадь исходной фигуры в общем случае не совпадает с суммой площадей получившихся прямоугольников. Так и должно было быть, ведь при дискретизации информация о поведении функций между точками отсчёта была утеряна. Однако при уменьшении шага дискретизации h эта разница будет уменьшаться.

На рисунке 9.17 высота прямоугольника рассчитывается как разность значений функций на левой границе отрезка (можно использовать и правую границу). На практике обычно вычисляют высоту *в середине отрезка* (в точке $x + h/2$), тогда площадь прямоугольника будет более близка к площади полосы исходной фигуры. Заметим, что ширина каждого из прямоугольников равна h, поэтому в программе умножение на h можно выполнить в конце цикла:

```
S:=0; x:=a
нц пока x<b
  S:=S+f1(x+h/2)-f2(x+h/2)
  x:=x+h
кц
S:=h*S
вывод 'Площадь ', S
```

```
S:=0; x:=a;
while x<b do begin
  S:=S+f1(x+h/2)-f2(x+h/2);
  x:=x+h
end;
S:=h*S;
writeln('Площадь ', S:8:3);
```

Этот метод называется *методом прямоугольников*.

Вместо прямоугольника для замены удобно применять ещё одну известную фигуру, для которой легко считается площадь, — *трапецию* (рис. 9.18).

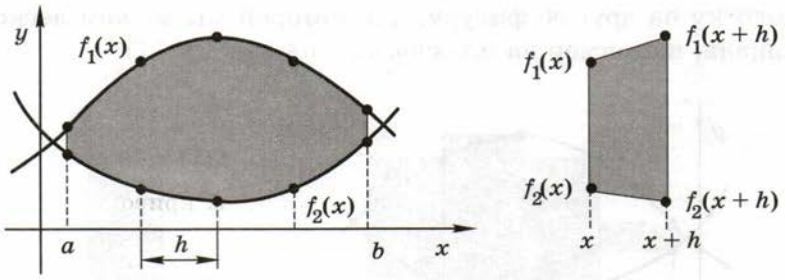


Рис. 9.18

Площадь трапеции равна произведению полусуммы её оснований на высоту, т. е. для элементарной трапеции с левой границей в точке x получаем выражение для площади:

$$\frac{h}{2}\cdot\left[f_1(x)-f_2(x)+f_1(x+h)-f_2(x+h)\right].$$

Простейшая программа выглядит так:

```
S:=0; x:=a
нц пока x<b
  S:=S+f1(x)-f2(x)
  S:=S+f1(x+h)-f2(x+h)
  x:=x+h
кц
S:=h*S/2
вывод 'Площадь ', S
```

```
S:=0; x:=a
while x<b do begin
  S:=S+f1(x)-f2(x)
      +f1(x+h)-f2(x+h);
  x:=x+h
end;
S:=h*S/2;
writeln('Площадь ', S:8:3);
```

Этот метод называется *методом трапеций*.

Заметим, что правое основание очередной трапеции совпадает с левым основанием следующей, поэтому можно немного изменить программу, чтобы на каждом шаге цикла вычислялась разность функций только в одной точке. Сделайте это самостоятельно.

Вопросы и задания

1. Что такое дискретизация?
2. Что даёт дискретизация в рассмотренных в параграфе задачах?
3. Что такое шаг дискретизации? Как должна быть связана его величина с длиной отрезка $[a, b]$?
4. Как зависят точность и время вычислений от выбора шага дискретизации?

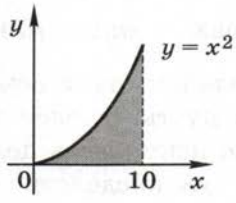
5. Почему методы, основанные на дискретизации, всегда дают приближённый результат?
6. Подумайте, можно ли выполнять дискретизацию с переменным шагом. Ответ обоснуйте.

Задачи

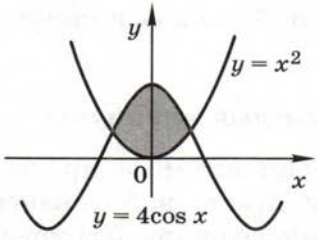
1. Измените программу для вычисления длины кривой так, чтобы на каждом шаге цикла вычислять только одно значение функции.
2. Найдите приближённо длину параболы $y = x^2$ на отрезке $x \in [0, 10]$.
3. Для примера, разобранного в § 70, вычислите длину траектории движения шарика для углов вылета 35,5° и 65,8°. Сравните полученные результаты. Постройте эти траектории с помощью табличного процессора.
4. Решите задачу 3 при разных значениях шага. Какой шаг вы рекомендуете выбрать для этого случая? Почему?
5. В чём заключается дискретизация при вычислении площади?

*6. Измените программу для вычисления площади методом трапеций так, чтобы повторно не вычислять одни и те же величины.

7. Найдите площадь фигуры, ограниченной параболой $y = x^2$ и осью OX, на отрезке $x \in [0, 10]$.



*8. Найдите площадь фигуры, ограниченной графиками функций $y = x^2$ и $y = 4\cos x$.



*9. Найдите площадь фигуры, ограниченной эллипсом

$$\frac{x^2}{4} + \frac{y^2}{9} = 1.$$

*10. Найдите с помощью приближённых методов площадь круга радиуса $R = 2$. Используя это значение, из формулы $S = \pi \cdot R^2$ определите приближенно число π.

§ 72 Оптимизация

Что такое оптимизация?

Оптимизация — это поиск наилучшего (оптимального) решения задачи в заданных условиях.

С точки зрения математики, цель оптимизации — выбрать неизвестную величину x (или несколько неизвестных величин — массив) наилучшим образом.

Чтобы задача оптимизации была корректной, нужно определить **целевую функцию** $f(x)$. Оптимальным называется такое решение, при котором целевая функция достигает максимума (если это «доходы», «прибыль») или минимума («расходы», «потери»):

$$f(x) \to \max \quad \text{или} \quad f(x) \to \min.$$

Чтобы задача оптимизации стала осмысленной, нужно ввести **ограничения**. Например, пусть человек хочет построить загородный дом за минимальную цену (здесь целевая функция — это общая цена строительства, нужно сделать её минимальной). Очевидно, что лучшее решение — не строить дом вообще, при этом расходы будут равны нулю. Такое «оптимальное» решение получено потому, что мы не задали ограничения (например, нужен двухэтажный дом с гаражом, балконом и камином).

Локальные и глобальный минимумы

По традиции в теории оптимизации рассматривают задачу поиска минимума. Если нужно найти максимум, просто меняют знак функции: значение функции $f(x)$ максимально там, где значение функции $(-f(x))$ минимально.

В математике различают **локальный** («местный») и **глобальный** («общий») **минимумы**. В точках x_1, x_2 и x_3 функция, график которой показан на рис. 9.19, имеет локальные минимумы, это значит, что слева и справа от этих точек функция возрастает.

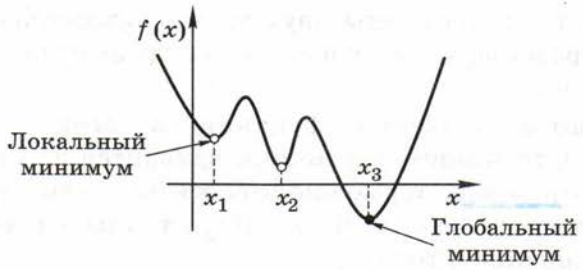


Рис. 9.19

Минимум в точке x_3 — глобальный, потому что здесь функция имеет наименьшее значение во всей рассматриваемой области.

Очевидно, что нас всегда интересует глобальный минимум. Однако большинство существующих методов оптимизации предназначено именно для поиска локальных минимумов[1] вблизи заданной начальной точки (начального приближения). Можно представить себе, что график функции — это срез поверхности, на которую устанавливается шарик в некоторой начальной точке; куда этот шарик скатится, такой минимум и будет найден.

Результат локальной оптимизации зависит от выбранного начального приближения.

Метод дихотомии

Дихотомия (греч. διχοτομια — деление надвое) — это метод поиска минимума функции, который очень напоминает метод деления отрезка пополам (бисекции). Пусть дана непрерывная функция $f(x)$, имеющая на отрезке $[a, b]$ один минимум в точке x^*. Нужно найти это значение с заданной точностью ε.

Как и в методе бисекции, мы последовательно «сжимаем» отрезок, пока его ширина не будет достаточно мала (меньше, чем 2ε). На каждом шаге (рис. 9.20):

1) вычисляется середина отрезка $c = \dfrac{a+b}{2}$;

[1] Для некоторых типов функций существуют методы глобальной оптимизации, но они сложны и выходят за рамки школьного курса.

2) вычисляются координаты двух точек, симметричных относительно середины: $x_1 = c - r$ и $x_2 = c + r$, где $0 < r < (b-a)/2$ — некоторое число;

3) сравниваются значения функции в этих точках: если $f(x_1) > f(x_2)$, то минимум функции находится на отрезке $[x_1, b]$, поэтому отрезок $[a, x_1]$ можно отбросить — переместить левую границу в точку x_1; если $f(x_1) < f(x_2)$, то правая граница отрезка перемещается в точку x_2.

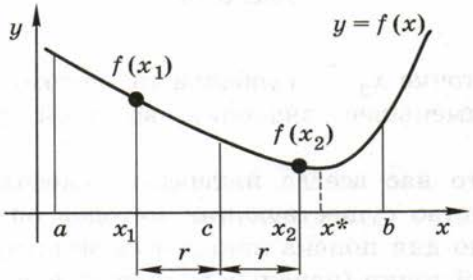


Рис. 9.20

Остаётся один вопрос: как выбирать r на каждом шаге? Проще всего вычислять его по формуле $r = k \cdot (b-a)$, где k — постоянный коэффициент $(0 < k < 0{,}5)$. Тогда ширина отрезка на следующем шаге будет равна

$$\frac{b-a}{2} + k \cdot (b-a) = (0{,}5 + k) \cdot (b-a),$$

т. е. составит $0{,}5 + k$ от первоначальной ширины. Для ускорения поиска выгодно, чтобы это отношение было как можно меньше, т. е. чтобы коэффициент k был возможно ближе к нулю (ноль выбирать нельзя, потому что при этом точки x_1 и x_2 совпадут, и метод не будет работать). Программа может выглядеть примерно так:

```
k:=0.01
delta:=2*eps
нц пока b-a>delta
  r:=k*(b-a)
  x1:=(a+b)/2-r
  x2:=(a+b)/2+r
  если f(x1)>f(x2) то
        a:=x1
```

```
k:=0.01;
delta:=2*eps;
while b-a>delta do begin
  r:=k*(b-a);
  x1:=(a+b)/2-r;
  x2:=(a+b)/2+r;
  if f(x1)>f(x2) then
        a:=x1
```

```
  иначе b:=x2                    else b:=x2
  все                          end;
кц                             writeln('x =', (a+b)/2:10:3);
вывод 'x =', (a+b)/2
```

В качестве упражнения можно исследовать работу этой программы при разных значениях k, подсчитав для каждого варианта количество шагов цикла, которое потребуется для получения решения с заданной точностью.

Существует вариант метода дихотомии, при котором на каждом шаге цикла нужно вычислять только одно значение функции (второе «переходит» с предыдущего шага). В этом случае нужно выбирать коэффициент k так, чтобы выполнялось равенство $0{,}5 + k = \frac{1}{\phi}$, где ϕ — отношение «золотого сечения»:

$$\phi = \frac{1+\sqrt{5}}{2} \approx 1{,}618\ldots$$

Пример: оптимальная раскройка листа

Рассмотрим пример практической задачи оптимизации. В углах квадратного листа железа, сторона которого равна 1 м, вырезают четыре квадрата со стороной x. Затем складывают получившуюся развёртку (по штриховым линиям на рис. 9.21), сваривают швы и таким образом получается бак.

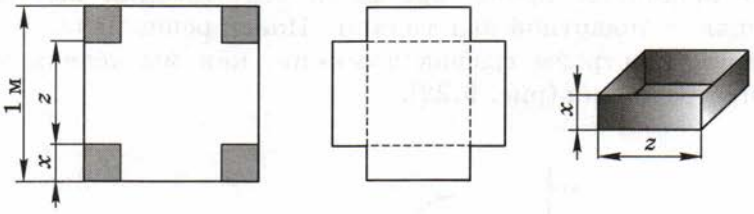


Рис. 9.21

Требуется выбрать размер выреза x так, чтобы получился бак наибольшего объёма.

Для того чтобы грамотно поставить задачу оптимизации, нужно:

1) определить целевую функцию: в данном случае выразить объём бака через неизвестную величину x;
2) задать ограничения на возможные значения x.

Легко видеть, что основание получившегося бака — это квадрат со стороной z (см. рис. 9.21), а его высота равна x. Величина z зависит от x и равна $z = 1 - 2x$, поэтому объём бака вычисляется по формуле $V = x(1 - 2x)^2$, это и есть целевая функция, для которой нужно найти максимум.

Понятно, что x не может быть меньше нуля. Вместе с тем x не может быть больше, чем половина стороны исходного листа (0,5 м), поэтому ограничения запишутся в виде двойного неравенства: $0 \le x \le 0{,}5$. Заметим, что при $x = 0$ и $x = 0{,}5$ объём бака равен нулю (в первом случае равна нулю высота, во втором — площадь основания).

Таким образом, нужно искать максимум целевой функции $f(x) = x(1 - 2x)^2$ на отрезке [0, 0,5]. Для этого можно использовать метод дихотомии (сделайте это самостоятельно). Не забудьте, что приведённый выше вариант программы рассчитан на поиск минимума, а в этой задаче нужно найти максимум.

Использование табличных процессоров

В стандартной поставке OpenOffice.org Calc модуль оптимизации работает только для линейных функций. Для того чтобы решить рассмотренную выше задачу с баком, нужно использовать модуль Solver for Nonlinear Programming (он входит в стандартную поставку последних версий LibreOffice) и в параметрах модуля оптимизации выбрать один из методов нелинейной оптимизации. В табличном процессоре Excel оптимизация выполняется с помощью стандартной надстройки «Поиск решения».

Сначала построим график функции, как мы делали это при решении уравнения (рис. 9.22).

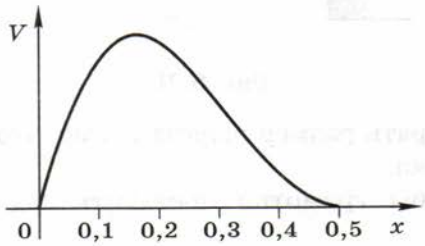


Рис. 9.22

По этому графику видно, что функция имеет единственный максимум в районе точки $x_0 = 0{,}2$. Это значение можно выбрать в качестве начального приближения для поиска.

Выделим в таблице две ячейки (например, E2 и F2), в первую запишем начальное приближение (0,2), а во вторую — формулу для вычисления объёма для этого значения x (рис. 9.23).

	E	F
1	x	Объем
2	0,200	0,072

Рис. 9.23

Задача оптимизации формулируется в виде «*найти максимум (или минимум) целевой функции в ячейке ..., изменяя значения ячеек ... при ограничениях ...*». В нашей задаче целевая ячейка — F2 (нужно найти ее максимум), изменяемая ячейка — E2.

Выбираем в главном меню пункт **Сервис**, **Поиск решения**, в появившемся окне вводим адреса целевой и изменяемой ячеек (для этого можно установить курсор в нужное поле ввода и щёлкнуть по ячейке) и ограничения (рис. 9.24).

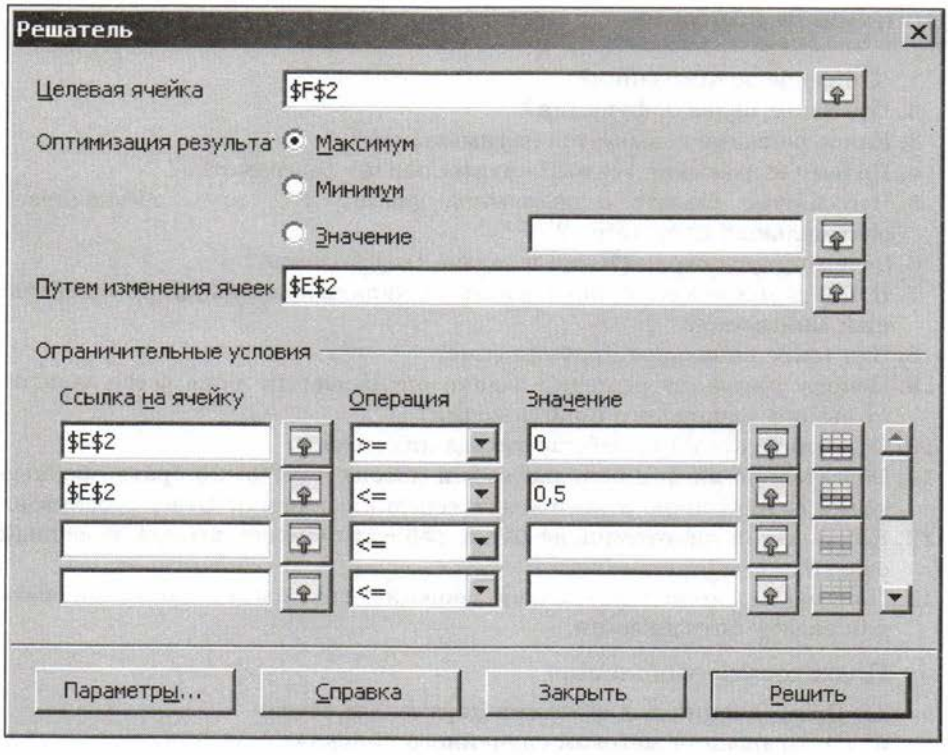


Рис. 9.24

После щелчка на кнопке **Решить** в ячейку E2 будет записано оптимальное значение x, а в целевую ячейку — соответствующее (максимальное) значение объёма.

Надстройка «Поиск решения» позволяет:

- находить максимум или минимум целевой функции;
- решать уравнения, задавая желаемое значение целевой функции;
- использовать несколько изменяемых ячеек и диапазонов, например запись A2:A6;B15 в списке изменяемых ячеек означает «изменять все ячейки диапазона A2:A6 и ячейку B15»;
- использовать ограничения типа «меньше или равно», «больше или равно», «равно», «целое» и «двоичное» (только 0 или 1).

Кнопка **Параметры** позволяет опытным пользователям менять настройки алгоритма оптимизации.

Вопросы и задания

1. Что такое оптимизация?
2. Что такое целевая функция?
3. Какое решение называется оптимальным?
4. Почему выражение «самый оптимальный» безграмотно?
5. Что можно сказать о рекламной фразе «Этот крем обеспечивает оптимальный цвет лица»?
6. Зачем нужны ограничения в задаче оптимизации?
7. В чём разница между понятиями «локальный минимум» и «глобальный минимум»?
8. Что такое начальное приближение?
9. Почему результат решения задачи оптимизации чаще всего зависит от выбора начального приближения?
10. Объясните принцип работы метода дихотомии.
11. Обязательно ли при использовании метода дихотомии брать пробные точки симметрично относительно середины отрезка? Ответ обоснуйте.
12. Когда метод дихотомии не будет работать (может выдать неверный ответ)?
13. Подумайте, можно ли задачу решения уравнения сформулировать как задачу оптимизации.

Подготовьте сообщение:

а) «Оптимизация для функции двух переменных»
б) «Оптимизация методом случайного поиска»
в) «Программное обеспечение для решения задач оптимизации»

Задачи

1. Примените метод дихотомии для решения задачи оптимальной раскройки, которая разобрана в параграфе. Решите задачу, используя разные значения коэффициента k, и постройте график зависимости количества шагов цикла от k.

*2. Напишите программу, которая реализует метод «золотого сечения» (на каждом шаге вычисляется только одно значение функции).

*3. Банка имеет форму цилиндра[1], полная площадь её поверхности (боковая поверхность и два круга-основания) равна 100 см^2. Определите радиус и высоту банки, которая при этих условиях имеет максимальный объём.

*4. Банка имеет форму цилиндра, её объём равен 500 см^3. Определите радиус и высоту банки, которая при этих условиях имеет минимальную площадь полной поверхности.

5. Некоторая фирма хочет провести рекламную кампанию в газетах. Данные о цене рекламного объявления и тиражах газет внесены в таблицу:

Газета	Тираж	Цена одного объявления, руб.	Кол-во объявлений	Расходы, руб.	Охват
Ведомости	10 000	1000	1	1000	10 000
Туризм	3500	570	2	1140	7000
Спорт	6000	700	3	2100	18 000
Правда	20 000	1250	4	5000	80 000
Всего				9240	115 000

В каждую газету нужно дать не менее одного и не более 6 объявлений. С помощью надстройки «Поиск решения» табличного процессора определите, сколько объявлений нужно дать в каждую газету, чтобы обеспечить общий охват не менее 200 000 человек и при этом израсходовать как можно меньше денег.

6. В условиях задачи 5 определите, сколько объявлений нужно дать в каждую газету, чтобы обеспечить наибольший общий охват и при этом израсходовать не более 15 000 рублей.

---

[1] Задачи 3 и 4 предложены В. Я. Лаздиным.

§ 73
Статистические расчёты

Статистика — это наука, которая изучает методы обработки и анализа больших массивов данных.

Табличные процессоры стали незаменимым инструментом для решения этих задач. И Excel, и Calc содержат несколько десятков встроенных статистических функций.

Свойства ряда данных

Простейшая задача статистики — изучить свойства одного ряда данных. **Ряд данных** X длиной n — это множество значений $x_1, x_2, \ldots, x_n$. Для ряда можно определить количество элементов, их сумму, среднее значение, минимальное и максимальное значения и т. д. Далее мы приводим как английские названия функций (для Calc), так и русские (для русской версии Excel).

Пусть ряд данных записан в ячейках A1:A20. Его сумма, среднее, минимальное и максимальное значения могут быть определены с помощью следующих формул:

сумма:	=SUM(A1:A20)	=СУММ(A1:A20)
среднее:	=AVERAGE(A1:A20)	=СРЗНАЧ(A1:A20)
минимальное:	=MIN(A1:A20)	=МИН(A1:A20)
максимальное:	=MAX(A1:A20)	=МАКС(A1:A20)

Функции SUM (русское название — СУММ) и AVERAGE (СРЗНАЧ) учитывают только числа в указанном диапазоне (без учёта пустых и текстовых ячеек). Количество числовых значений можно подсчитать с помощью функции COUNT (СЧЁТ), например:

=COUNT(A1:A20) =СЧЁТ(A1:A20)

С помощью функции COUNTIF (СЧЕТЕСЛИ) можно подсчитать число элементов ряда, удовлетворяющих некоторому условию. Например, если в диапазоне A1:A20 записаны школьные отметки, формула

=COUNTIF(A1:A20;"=5")

вычисляет число пятёрок, а формула

=COUNTIF(A1:A20;">3")

вычисляет общее число четвёрок и пятёрок (всех ячеек, значения которых больше 3).

Более сложная характеристика ряда — **среднеквадратическое отклонение** или **стандартное отклонение** σ_x, с помощью которого оценивается «разброс» значений $x_1, x_2, \ldots, x_n$ относительно среднего значения ряда $\overline{x}$:

$$\sigma_x = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \overline{x})^2}.$$

Среднеквадратическое отклонение — это неотрицательная величина (подумайте почему). Чем больше σ_x, тем больше разброс значений относительно среднего. Для вычисления стандартного отклонения в табличных процессорах используется функция STDEVP (СТАНДОТКЛОНП):

=STDEVP(A1:A20) =СТАНДОТКЛОНП(A1:A20)

Условные вычисления

Как вы знаете, в программировании важнейшую роль играют условные операторы (ветвления), позволяющие выбирать один из двух (или нескольких) вариантов обработки данных. В табличных процессорах тоже возможны **условные вычисления**, при которых в ячейку заносится то или иное значение в зависимости от выполнения какого-то условия.

Предположим, что в книжном интернет-магазине «Бука» доставка покупок бесплатна для тех, кто сделал заказ на сумму более 500 рублей, а для остальных доставка стоит 20% от суммы (рис. 9.25).

	A	B	C
1	**Заказ**	**Сумма**	**Доставка**
2	1234	256 руб.	51 руб.
3	1345	128 руб.	26 руб.
4	1456	1 024 руб.	0
5	1565	512 руб.	0
6	1576	345 руб.	69 руб.

Рис. 9.25

Таким образом, есть два варианта вычисления стоимости доставки, поэтому в формулах столбца C нужно использовать ветвление. Например, алгоритм вычисления значения ячейки C2 может выглядеть так: «если B2>500, записать в ячейку 0, иначе записать значение B2*0,2». В программе на Паскале мы бы записали:

```
if B2>500 then
     C2:=0
else C2:=B2*0.2;
```

В табличных процессорах для условных вычислений используют функцию IF (ЕСЛИ):

=IF(B2>500;0;B2*0,2) =ЕСЛИ(B2>500;0;B2*0,2)

У этой функции три аргумента, разделённые точками с запятой:

1) условие (B2>500);
2) значение ячейки в том случае, когда условие истинно (0);
3) значение ячейки в том случае, когда условие ложно (B2*0,2).

Первый аргумент может быть сложным условием, которое строится с помощью функций NOT (НЕ, отрицание), AND (И, логическое умножение) и OR (ИЛИ, логическое сложение). Например, если бесплатная доставка распространяется только на заказы, у которых номер меньше 1500 и сумма больше 500 рублей, в ячейку C2 нужно записать такую формулу:

=IF(AND(A2<1500;B2>500);0;B2*0,2)

Здесь использовано сложное условие AND(A2<1500;B2>500), которое истинно только при одновременном выполнении обоих простых условий: A2<1500 и B2>500.

Второй и третий аргументы функции IF могут содержать вложенные вызовы этой функции. Пусть, например, для заказов стоимостью более 200 рублей (но не больше 500) стоимость доставки составляет 10% от суммы:

```
if B2>500 then
  C2:=0
else
  if B2>200 then
     C2:=B2*0.1
 else C2:=B2*0.2;
```

В табличном процессоре этот алгоритм запишется в виде:

=IF(B2>500;0;IF(B2>200;B2*0,1;B2*0,2))

Связь двух рядов данных

Одна из задач обработки данных — установить взаимосвязь между величинами, процессами, явлениями. Пусть существуют два ряда данных одинаковой длины:

$$x_1, x_2, \ldots, x_n \quad \text{и} \quad y_1, y_2, \ldots, y_n.$$

Например, первый ряд — это температура воздуха за n последних дней, а второй ряд — значения атмосферного давления в те же дни. Требуется определить, есть ли связь между этими рядами, и оценить, насколько она сильная.

Для решения этой задачи чаще всего используется **коэффициент корреляции** (англ. *correlation* — взаимоотношение, связь):

$$\rho_{xy} = \frac{\frac{1}{n}\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sigma_x \cdot \sigma_y}.$$

Здесь $\overline{x}$ и $\overline{y}$ — средние значения рядов, а σ_x и σ_y — их среднеквадратические отклонения.

Величина ρ_{xy} — это безразмерный коэффициент[1], причём можно показать, что всегда $-1 \le \rho_{xy} \le 1$. Если $\rho_{xy} > 0$, то увеличение значения x (в среднем) приводит к увеличению y; если же $\rho_{xy} < 0$, то при увеличении x значение y чаще всего уменьшается.

Чем больше модуль ρ_{xy}, тем сильнее связь между двумя величинами. При $\rho_{xy} = -1$ или $\rho_{xy} = 1$ они строго связаны линейной зависимостью $y = kx + b$, где k и b — некоторые числа. В случае $\rho_{xy} = -1$ эта зависимость убывающая ($k < 0$), а при $\rho_{xy} = 1$ — возрастающая ($k > 0$).

Считается, что между x и y есть сильная связь, если $|\rho_{xy}| > 0{,}5$. При меньших значениях ρ_{xy} делать какие-то далеко идущие выводы не следует (связь слабая или не обнаружена).

Для вычисления коэффициента корреляции в табличных процессорах используется функция CORREL (КОРРЕЛ):

=CORREL(A1:A20;B1:B20) =КОРРЕЛ(A1:A20;B1:B20)

[1] Попробуйте доказать это самостоятельно.

Обратите внимание, что у этой функции два аргумента (два ряда данных одинаковой длины), адреса двух диапазонов отделяются точкой с запятой.

Нужно учитывать, что коэффициент корреляции лучше всего обнаруживает линейную зависимость. Если связь есть, но она далека от линейной, коэффициент корреляции может быть невысок. В таких случаях для установления связи нужно использовать более сложные методы, которые мы здесь рассматривать не будем.

Вопросы и задания

1. Что изучает статистика? Как вы думаете, в чём её задача?
2. Как влияют пустые ячейки в электронной таблице на результат работы функций COUNT и AVERAGE?
3. Чем отличаются функции COUNT и COUNTIF?
4. Что показывает среднеквадратическое отклонение?
5. Что показывает коэффициент корреляции?
6. Для двух рядов коэффициент корреляции равен (–0,5). Что можно сказать о возможной связи этих рядов между собой?
7. Для двух рядов коэффициент корреляции равен 0,1. Что можно сказать о возможной связи этих рядов между собой?

Задачи

1. В электронной таблице значение формулы =SUM(B1:B2) равно 5. Чему равно значение ячейки B3, если значение формулы =AVERAGE(B1:B3) равно 3?
2. В электронной таблице значение формулы =SUM(C3:E3) равно 12. Чему равно значение формулы =AVERAGE(C3:F3), если значение ячейки F3 равно 4?
3. В электронной таблице значение формулы =SUM(A2:D2) равно 12. Чему равно значение формулы =AVERAGE(A2:E2), если значение ячейки E2 равно 13?
4. В электронной таблице значение формулы =AVERAGE(A6:C6) равно (–2). Чему равно значение формулы =SUM(A6:D6), если значение ячейки D6 равно 6?
5. В электронной таблице значение формулы =AVERAGE(B5:E5) равно 10. Чему равно значение формулы =SUM(B5:F5), если значение ячейки F5 равно 1?
6. В электронной таблице значение формулы =AVERAGE(A1:C1) равно 6. Чему равно значение ячейки D1, если значение формулы =SUM(A1:D1) равно 7?

7. В электронной таблице значение формулы =AVERAGE(A3:D4) равно 6. Чему равно значение формулы =AVERAGE(A3:C4), если значение формулы =SUM(D3:D4) равно 6?
8. В электронной таблице значение формулы =AVERAGE(C2:D5) равно 4. Чему равно значение формулы =SUM(C5:D5), если значение формулы =AVERAGE(C2:D4) равно 6?
*9. Как изменится значение ячейки C3, если после ввода формул переместить содержимое ячейки B2 в B3?

	A	B	C
1	1	2	
2	2	6	=COUNT(A1:B2)
3			=AVERAGE(A1:C2)

10. Доставка товара в фирме «Рога и копыта» стоит 200 рублей, если в доме есть лифт. Если лифта нет, подъём на каждый этаж стоит 200 рублей:

	A	B	C	D
1	**Заказ**	**Этаж**	**Лифт**	**Доставка**
2	12	5	нет	1000
3	34	2	да	200
4	56	8	да	200

Какую формулу нужно записать в ячейку D2?

11. При приёме на работу претенденты проходят два тура собеседования. В каждом туре выставляется оценка от 0 до 100 баллов. На работу принимаются те, кто в каждом туре набрал не менее 80 баллов.

	A	B	C	D
1	**Фамилия**	**1-й тур**	**2-й тур**	**Принят**
2	Иванов	80	80	да
3	Петров	90	70	нет
4	Сидоров	85	90	да

Какую формулу нужно записать в ячейку D2?

12. Решите задачу 11 при условии, что на работу принимаются все, кто набрал 90 баллов хотя бы в одном туре собеседования.

13. Сниженный тариф на телефонные разговор — 2 рубля за минуту разговора — действует в рабочие дни после 20 часов и в выходные дни. Обычный тариф — 5 рублей за минуту. Дни в таблице нумеруются с 1 до 7 (1 — понедельник).

	A	B	C	D
1	**Код**	**Время**	**День недели**	**Тариф, руб.**
2	12	21:12:20	2	2
3	34	17:23:50	1	5
4	56	10:21:42	7	2

Какую формулу нужно записать в ячейку D2? Момент времени 20:00:00 в формуле нужно записывать как TIME(20;0;0).

14. При покупке на сумму более 1000 рублей в магазине владельцам дисконтных карт предоставляется скидка 5%.

	A	B	C	D
1	**Код**	**Цена, руб.**	**Дисконтная карта**	**Со скидкой, руб.**
2	12	1 200	нет	1 200
3	34	1 450	да	1 378
4	56	750	да	750

Какую формулу нужно записать в ячейку D2?

15. Во время уценки цена всех товаров, которых хранятся на складе более 6 месяцев, снижается на 20% (если цена товара больше 1000 рублей) или на 10% (если цена меньше 1000 рублей).

	A	B	C	D
1	**Код**	**Цена, руб.**	**Хранится (месяцев)**	**Новая цена, руб.**
2	12	1 200	3	1 200
3	34	1 450	7	1 160
4	56	750	12	675

Какую формулу нужно записать в ячейку D2?

§ 74
Обработка результатов эксперимента

Зачем это нужно?

Многие практические задачи связаны с проведением эксперимента, в результате которого исследователь с помощью измерительных приборов получает массивы данных. Затем эти данные необходимо обработать, для того чтобы выявить закономерности, подтвердить или опровергнуть выводы теории и т. п.

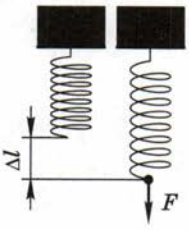


Рис. 9.26

Например, с помощью динамометра и линейки можно экспериментально определить жёсткость пружины (рис. 9.26). Для этого используется закон Гука, связывающий приложенную силу F, жёсткость пружины k и её удлинение Δl линейной зависимостью:

$$F = k \cdot \Delta l.$$

Определив жёсткость пружины k, мы сможем вычислять её удлинение при любой нагрузке (в пределах действия закона Гука), не выполняя новых измерений.

Величина k зависит от материала пружины и её размеров. Для определения жёсткости k на нижний конец пружины подвешивают груз известной массы m (так что сила $F = mg$ тоже известна) и измеряют удлинение пружины Δl. Тогда $k = F/\Delta l$.

Обычно такой эксперимент проводится несколько раз, в результате получается серия значений F_i (для $i = 1, 2, \ldots, n$) и соответствующих им удлинений Δl_i. Для каждой пары рассчитанная жёсткость $k_i = F_i/\Delta l_i$ будет, скорее всего, различной. Конечно, можно принять за k среднее значение по всем полученным измерениям. Однако такой подход не очень хорошо обоснован с научной точки зрения. Поэтому были разработаны другие методы, один из которых рассматривается далее.

Метод наименьших квадратов

Предположим, что есть два ряда данных одинаковой длины: $x_1, x_2, \ldots, x_n$ и $y_1, y_2, \ldots, y_n$. Предполагается, что они связаны линейной зависимостью $y = k \cdot x$, где k — неизвестный коэффициент. Требуется найти *оптимальное* значение k, которое лучше всего соответствует исходным данным.

Поскольку речь идёт о задаче оптимизации, нужно определить функцию, которая позволяет оценить, насколько хорошо выбранная зависимость соответствует исходным данным. Предположим, что выбран некоторый коэффициент k, так что для каждого x_i можно найти соответствующее ему значение функции $Y_i = k \cdot x_i$.

В идеале график функции должен проходить через все точки (x_1, y_1), (x_2, y_2), ..., (x_n, y_n), т. е. при всех i должно выполняться условие $Y_i = y_i$. Однако на практике этого, скорее всего, не будет (рис. 9.27).

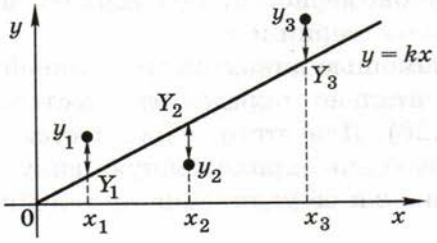


Рис. 9.27

Отклонение полученной линии от исходных данных определяется разностями $Y_i - y_i$: чем они меньше (по модулю), тем лучше соответствие. Однако сумма этих разностей даёт неправильную оценку точности — слагаемые с разными знаками могут скомпенсировать друг друга. Чтобы решить эту проблему, можно сложить квадраты этих величин и выбрать k так, чтобы эта сумма

$$E = \sum_{i=1}^{n} (Y_i - y_i)^2 = (Y_1 - y_1)^2 + (Y_2 - y_2)^2 + \ldots + (Y_n - y_n)^2$$

была минимальной. Такой подход называется **методом наименьших квадратов**. Впервые его предложил немецкий математик *К. Ф. Гаусс*.

Как найти коэффициент k наилучшим образом, т. е. так, чтобы сумма квадратов E была минимальной? Для этого заменим Y_i на $k \cdot x_i$ и раскроем скобки в формуле для вычисления E:

$$(Y_i - y_i)^2 = (k \cdot x_i - y_i)^2 = k^2 x_i^2 - 2k x_i y_i + y_i^2.$$

Группируя слагаемые, содержащие k^2 и k, получаем:

$$E = Ak^2 - Bk + C,$$

где $A = \sum_{i=1}^{n} x_i^2$, $B = 2\sum_{i=1}^{n} x_i y_i$ и $C = \sum_{i=1}^{n} y_i^2$. График зависимости E от k — это парабола, причём её ветви направлены вверх, потому что

$A > 0$ (это сумма квадратов). Вершина параболы (и минимум функции!) находится в точке $k = \frac{B}{2A}$, это и будет оптимальное решение.

Таким образом, алгоритм для определения оптимального значения k приобретает вид:

1) вычислить коэффициенты параболы $A = \sum_{i=1}^{n} x_i^2$ и $B = 2\sum_{i=1}^{n} x_i y_i$;

2) вычислить $k = \frac{B}{2A}$.

Решение получилось простым только потому, что мы выбрали очень простую функцию, линейную с нулевым свободным членом. В более сложных случаях строгое решение задачи оптимизации требует знания высшей математики.

Если исходные данные записаны в массивах $x[1..N]$ и $y[1..N]$, программа для рассмотренного случая выглядит так:

```
A:=0; B:=0
нц для i от 1 до N
  A:=A+x[i]*x[i]
  B:=B+x[i]*y[i]
кц
k:=B/A
```

```
A:=0; B:=0;
for i:=1 to N do begin
  A:=A+x[i]*x[i];
  B:=B+x[i]*y[i];
end;
k:=B/A;
```

Чтобы избавиться от лишних операций, умножение на 2 при вычислении B и деление на 2 при вычислении k не выполняется (применение двух этих операций не меняет результат).

Для решения задачи методом наименьших квадратов можно использовать табличные процессоры с модулем поиска решения. Пусть в результате измерений получены точки (1; 1,1), (2; 1,8) и (3; 3,5). Занесём эти координаты в столбцы A и B, в ячейку B1 запишем начальное приближение для k, а в столбец C — значения функции $y = k \cdot x$ для значений x из столбца A (рис. 9.28).

	A	B	C
1	k	1,000	
2	E	=SUMXMY2(B5:B7;C5:C7)	
3			
4	x	y	Y
5	1	1,1	=$B$1*A5
6	2	1,8	=$B$1*A6
7	3	3,5	=$B$1*A7

Рис. 9.28

Величина E — это сумма квадратов разностей двух рядов, которая вычисляется с помощью функции SUMXMY2 (СУММКВРАЗН). У этой функции два аргумента — ряд y (измеренные значения в столбце B) и ряд Y (вычисленные значения функции в столбце C). Задача оптимизации — найти минимальное значение E (в ячейке B2), изменяя значение k в ячейке B1 — решается с помощью надстройки «Поиск решения».

Восстановление зависимостей

Пусть заданы пары значений x и y, и предполагается, что они связаны некоторой зависимостью $y = f(x)$, которую нужно найти для того, чтобы вычислять значение функции в других точках, для которых значения y неизвестны. Такая задача называется **задачей восстановления зависимости**.

Если вид функции не задан, эта задача некорректна, потому что через заданные точки можно провести сколько угодно различных линий (графиков функций), и невозможно сказать, какая из них лучше подходит (рис. 9.29).

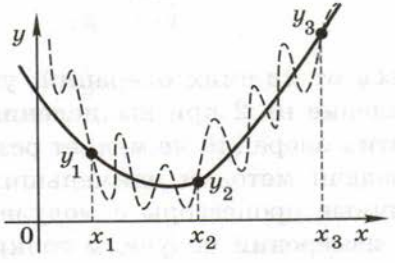


Рис. 9.29

Поэтому для того, чтобы сделать задачу осмысленной, нужно заранее задать вид функции, так что останется только найти её неизвестные коэффициенты.

Откуда взять вид функции? В некоторых случаях он известен из физических законов, описывающих явление (так было в примере с исследованием закона Гука). Иногда вид зависимости можно определить по внешнему виду расположения точек. Также можно попробовать функции разного типа и выбрать лучший вариант. Часто используют следующие типы функций (рис. 9.30):

- линейную $y = a \cdot x + b$;
- логарифмическую $y = a \cdot \ln x + b$;
- показательную (экспоненциальную) $y = a \cdot b^x$;
- степенную $y = a \cdot x^b$.

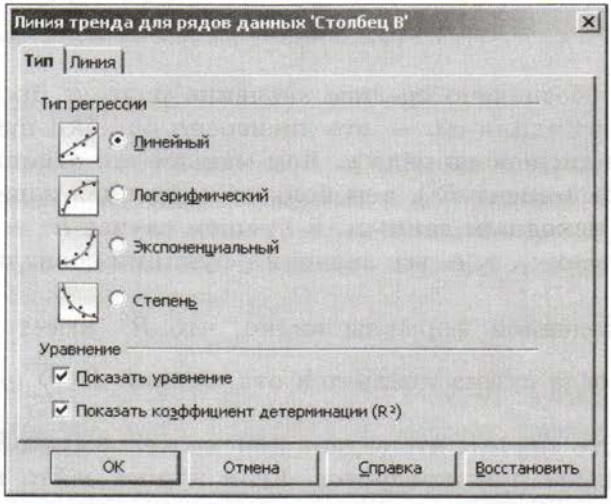


Рис. 9.30

Задача сводится к тому, чтобы выбрать коэффициенты a и b наилучшим образом. Для её решения «вручную» нужно применять методы вычислительной математики, выходящие за рамки школьного курса. Однако в современных табличных процессорах есть встроенные возможности для решения задачи восстановления зависимостей. Полученные графики оптимальных функций называются **линиями тренда** (англ. *trend* — основное направление развития).

Сначала нужно ввести исходные данные (координаты точек) в таблицу и построить по ним диаграмму типа **Диаграмма XY** (в Excel — диаграмма **Точечная**). Лучше оставить на диаграмме только точки, не соединяя их линией.

Для того чтобы построить линию тренда, надо щёлкнуть правой кнопкой мыши на одной из точек и выбрать пункт **Вставить линию тренда** из контекстного меню. В появившемся окне можно выбрать вид зависимости. Если установить флажок **Показать уравнение**, уравнение линии тренда будет показано на диаграмме. Флажок **Показать коэффициент детерминации (R^2)** позволяет

увидеть, насколько точно полученная линия соответствует исходным данным. Коэффициент R^2 вычисляется по формуле

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - Y_i)^2}{\sum_{i=1}^{n}(y_i - \overline{y})^2},$$

где через $\overline{y}$ обозначено среднее значение ряда y. Дробь, которая вычитается из единицы, — это дисперсия ошибки приближения, делённая на дисперсию ряда y. Чем меньше это отношение (и чем больше коэффициент R^2), тем больше найденная зависимость соответствует исходным данным. В лучшем случае $R^2 = 1$, при этом $y_i = Y_i$ для всех i, т. е. все значения функции совпадают с заданными.

Из приведённой формулы видно, что R^2 имеет наибольшее значение, когда сумма квадратов отклонений $E = \sum_{i=1}^{n}(y_i - Y_i)^2$ минимальна. Это значит, что задача поиска максимума R^2 решается методом наименьших квадратов. Если нужно найти неизвестные коэффициенты функции, которая не входит в стандартный набор (например, $y = a \cdot \sin bx + c$), можно применить метод наименьших квадратов с помощью надстройки «Поиск решения».

Прогнозирование

Во многих задачах (например, в экономике) в результате обработки данных нужно сделать **прогноз**. Если найдена зависимость $y = f(x)$, эта задача решается просто — нужно найти значения функции для тех значений x, которые нас интересуют. Однако может получиться так, что функция, которая очень хорошо соответствует имеющимся данным (см. функцию $y = f(x)$ на рис. 9.31), оказывается непригодна для прогноза.

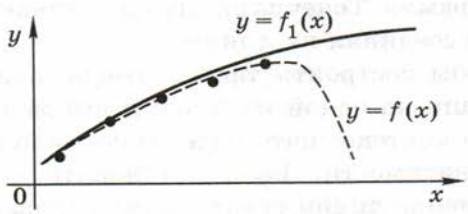


Рис. 9.31

В таких случаях для решения задачи прогнозирования нужно выбирать другую функцию, которая дает меньшее значение R^2, но лучше показывает закономерность изменения величины (например, возрастающий характер функции).

Вопросы и задания

1. Объясните, почему экспериментальные исследования требуют специальных методов обработки данных.
2. Объясните суть метода наименьших квадратов. Почему можно считать такой подход решением задачи оптимизации?

Подготовьте сообщение

а) «Интерполяция»
б) «Экстраполяция»
в) «Аппроксимация»

Задачи

1. Результаты серии измерений, сделанных для определения жёсткости пружины на основе закона Гука, записаны в таблицу:

F, Н	1	3	6	10
Δl, м	0,028	0,070	0,170	0,260

Используя метод наименьших квадратов, определите жёсткость пружины. Решите задачу разными способами (с помощью собственной программы и табличного процессора), сравните результаты.

*2. Используя те же данные, что и в задаче 1, найдите жёсткость пружины другим способом. Введём величину, обратную жёсткости: $\eta = 1/k$, тогда $\Delta l = \eta \cdot F$. Сначала, применив метод наименьших квадратов, найдите η, а затем рассчитайте $k = 1/\eta$. Сравните результат с тем, что получилось в задаче 1. Объясните расхождение.

3. Почему задача восстановления зависимости некорректна, если не задан вид функции?
4. Доходы начинающей фирмы (в тысячах рублей) за первые 5 лет работы приведены в таблице:

Год	1	2	3	4	5
Доход	93	187	270	321	350

С помощью табличного процессора найдите зависимость дохода от года работы (выберите лучший из стандартных вариантов, с наибольшим значением R^2). С помощью этой зависимости сделайте прогноз развития фирмы на 2 года вперед.

***5.** При изучении волн измерили отклонение уровня поверхности воды y от «нулевого» уровня в одной точке в разные моменты времени t:

t	0	0,2	0,4	0,6	0,8	1
y	1	2	1,2	–0,6	–2	–1,5

Предполагается, что волна описывается формулой $y = a \cdot \sin(b \cdot t + c)$, где a, b и c — некоторые числа. Определите неизвестные коэффициенты методом наименьших квадратов с помощью табличного процессора. В качестве начального приближения можно выбрать $a \approx 1$; $b \approx 4$; $c \approx 1$.

Практические работы к главе 9

Работа № 62 «Решение уравнений методом перебора»
Работа № 63 «Решение уравнений методом деления отрезка пополам»
Работа № 64 «Решение уравнений в табличных процессорах»
Работа № 65 «Вычисление длины кривой»
Работа № 66 «Вычисление площади фигуры»
Работа № 67 «Оптимизация. Метод дихотомии»
Работа № 68 «Оптимизация с помощью табличных процессоров»
Работа № 69 «Статистические расчеты»
Работа № 70 «Условные вычисления»
Работа № 71 «Метод наименьших квадратов»
Работа № 72 «Линии тренда»

ЭОР к главе 9 на сайте ФЦИОР (http//fcior.edu.ru)

- Точность вычислений
- Особенности выполнения вычислений на ЭВМ. Потеря точности
- Уравнения с одной переменной. Корни уравнения. Линейные уравнения
- Дискретизация сигналов
- Задачи оптимизации. Динамическое программирование
- Основные алгоритмы работы со структурами данных

Самое важное в главе 9

- Погрешность вычислений — это разница между вычисленным значением величины и её точным (истинным) значением. Погрешность бывает абсолютная и относительная (в процентах). Для оценки точности измерений и расчётов более полезна относительная погрешность.
- Точность вычислений определяется точностью исходных данных и погрешностями метода.
- Компьютер позволяет легко находить приближённые решения многих задач, которые трудно решаются аналитически (или не решаются вообще).
- Многие приближённые методы решения уравнений — итерационные, т. е. основаны на многократном повторении некоторого алгоритма, который позволяет на каждом шаге приближаться к точному решению. Процедура заканчивается, когда отклонение полученного приближённого решения от точного становится меньше заданной величины ε.
- Дискретизация — необходимый этап подготовки вычислительных задач для решения на компьютере. Чтобы повысить точность, нужно уменьшать шаг дискретизации, но это увеличивает объём вычислений.
- Оптимизация — это поиск наилучшего решения в заданных условиях. В задачах оптимизации необходимо ввести целевую функцию, для которой требуется найти минимум или максимум. Кроме того, в большинстве практических случаев нужно задать ограничения на допустимые значения переменных.
- Большинство приближённых методов оптимизации позволяет искать только локальный минимум (максимум), в этом случае результат оптимизации зависит от выбора начального приближения.
- Цель статистических расчётов — выявить закономерности путём математической обработки больших массивов данных.
- Для того чтобы задача восстановления зависимости по точкам была корректной, необходимо заранее выбрать вид функции, так что останется определить лишь её неизвестные коэффициенты.

Глава 10
Информационная безопасность

§ 75
Основные понятия

Зависимость современных организаций от компьютерных технологий стала настолько сильной, что вывод из строя компьютерной сети или программного обеспечения может остановить работу предприятия. Чтобы этого не произошло, нужно соблюдать правила информационной безопасности.

Информационная безопасность — это защищённость информации от любых действий, в результате которых владельцам или пользователям информации может быть нанесён недопустимый ущерб. Причиной такого ущерба может быть искажение или утеря информации, а также неправомерный доступ к ней.

Прежде всего в защите нуждаются государственная и военная тайны, а также коммерческая, юридическая и врачебная тайны. Необходимо защищать личную информацию: паспортные данные, данные о банковских счетах, пароли на сайтах, а также любую информацию, которую можно использовать для шантажа, вымогательства и т. п.

Конечно, невозможно защититься от любых потерь, поэтому задача состоит в том, чтобы исключить именно *недопустимый* ущерб. С точки зрения экономики, средства защиты не должны стоить больше, чем возможные потери.

Защита информации — это меры, направленные на то, чтобы не потерять информацию, не допустить её искажения, а также не допустить, чтобы к ней получили доступ люди, не имеющие на это права. В результате нужно обеспечить:

- доступность информации — возможность получения информации за приемлемое время;
- целостность (отсутствие искажений) информации;
- конфиденциальность информации (недоступность для посторонних).

Доступность информации нарушается, например, когда оборудование выходит из строя или веб-сайт не отвечает на запросы пользователей в результате массовой атаки вредоносных программ через Интернет.

Нарушения **целостности информации** — это кража или искажение информации, например подделка сообщений электронной почты и других цифровых документов.

Конфиденциальность информации нарушается, когда информация становится известной тем людям, которые не должны о ней знать (происходит перехват секретной информации).

В компьютерных сетях защищённость информации снижается в сравнении с отдельным компьютером, потому что:

- в сети работает много пользователей, их состав меняется;
- есть возможность незаконного подключения к сети;
- существуют уязвимости в сетевом программном обеспечении;
- возможны атаки взломщиков и вредоносных программ через сеть.

В России вопросы, связанные с защитой информации, регулирует **закон «Об информации, информационных технологиях и о защите информации»**.

Технические средства защиты информации — это замки, решётки на окнах, системы сигнализации и видеонаблюдения, другие устройства, которые блокируют возможные каналы утечки информации или позволяют их обнаружить.

Программные средства обеспечивают доступ к данным по паролю, шифрование информации, удаление временных файлов, защиту от вредоносных программ и др.

Организационные средства включают:

- распределение помещений и прокладку линий связи таким образом, чтобы злоумышленнику было сложно до них добраться;
- политику безопасности организации.

Серверы, как правило, находятся в отдельном (охраняемом) помещении и доступны только администраторам сети. Важная информация должна периодически копироваться на резервные носители (диски или магнитную ленту) для сохранения её в случае сбоев. Обычные сотрудники (не администраторы):

- имеют право доступа только к тем данным, которые им нужны для работы;

- не имеют права устанавливать программное обеспечение;
- раз в месяц должны менять пароли.

Самое слабое звено любой системы защиты — это человек. Некоторые пользователи могут записывать пароли на видном месте (чтобы не забыть) и передавать их другим, при этом возможность незаконного доступа к информации значительно возрастает. Поэтому очень важно обучить пользователей основам информационной безопасности.

Большинство утечек информации связано с *инсайдерами* (англ. *inside* — внутри) — недобросовестными сотрудниками, работающими в фирме. Известны случаи утечки закрытой информации не через ответственных сотрудников, а через секретарей, уборщиц и другой вспомогательный персонал. Поэтому ни один человек не должен иметь возможности причинить непоправимый вред (в одиночку уничтожить, украсть или изменить данные, вывести из строя оборудование).

Вопросы и задания

1. Что такое информационная безопасность?
2. Что входит в понятие «защита информации»?
3. На какие группы делятся средства защиты информации?
4. Какие меры безопасности обычно применяются в организациях?
5. Почему при объединении компьютеров в сеть безопасность снижается?
6. Кто такие инсайдеры?

§ 76
Вредоносные программы

Что такое компьютерный вирус?

Компьютерный вирус — это программа, способная создавать свои копии (не обязательно совпадающие с оригиналом) и внедрять их в файлы и системные области компьютера. При этом копии могут распространяться дальше.

Как следует из этого определения, основная черта компьютерного вируса — это способность распространяться при запуске.

Вирус — это один из типов вредоносных программ. Однако очень часто вирусами называют любые вредоносные программы (англ. *malware*).

Вредоносные программы — это программы, предназначенные для незаконного доступа к информации, для скрытого использования компьютера или для нарушения работы компьютера и компьютерных сетей.

Зачем пишут такие программы? Во-первых, с их помощью можно получить управление компьютером пользователя и использовать его в своих целях. Например, через заражённый компьютер злоумышленник может взламывать сайты и незаконно переводить на свой счёт деньги. Некоторые программы блокируют компьютер и для продолжения работы требуют отправить платное SMS-сообщение.

Зараженные компьютеры, подключенные к сети Интернет, могут объединяться в сеть специального типа — **ботнет** (от англ. *robot* — робот и *network* — сеть). Такая сеть часто состоит из сотен тысяч компьютеров, обладающих в сумме огромной вычислительной мощностью. По команде «хозяина» ботнет может организовать атаку на какой-то сайт. В результате огромного количества запросов сервер не справляется с нагрузкой, сайт становится недоступен, и бизнесмены несут большие денежные потери. Такая атака называется **DoS-атакой**[1] (англ. **DoS** — *Denial of Service* — отказ в обслуживании). Кроме того, ботнеты могут использоваться для подбора паролей, рассылки **спама** (рекламных электронных сообщений) и другой незаконной деятельности.

Во-вторых, некоторые вредоносные программы предназначены для **шпионажа** — передачи по Интернету секретной информации с вашего компьютера: паролей доступа к сайтам, почтовым ящикам, учётным записям в социальных сетях, банковским счетам и электронным платёжным системам. В результате таких краж пользователи теряют не только данные, но и деньги.

В-третьих, иногда вирусы пишутся ради самоутверждения программистами, которые по каким-то причинам не смогли при-

[1] Здесь речь идёт, строго говоря, о распределённой DoS-атаке (англ. **DDoS** — *Distributed DoS*), которая проводится сразу со многих компьютеров.

менить свои знания для создания полезного ПО. Такие программы нарушают нормальную работу компьютера: время от времени перезагружают его, вызывают сбои в работе операционной системы и прикладных программ, уничтожают данные.

Наконец, существуют вирусы, написанные ради шутки. Они не портят данные, но приводят к появлению звуковых или зрительных эффектов (проигрывание мелодии; искажение изображения на экране; кнопки, убегающие от курсора и т. п.).

Создание и распространение компьютерных вирусов и вредоносных программ — это уголовные преступление, которое предусматривает (в особо тяжких случаях) наказание до 7 лет лишения свободы (Уголовный кодекс РФ, статья 273).

Признаки заражения вирусом:

- замедление работы компьютера;
- уменьшение объема свободной оперативной памяти;
- зависание, перезагрузка или блокировка компьютера;
- ошибки при работе ОС или прикладных программ;
- изменение длины файлов, появление новых файлов (в том числе скрытых);
- рассылка сообщений по электронной почте без ведома автора.

Для того чтобы вирус смог выполнить какие-то действия, он должен оказаться в памяти в виде программного кода и получить управление компьютером.

Поэтому **вирусы заражают** не любые данные, а только **программный код**, который может выполняться. Например:

- исполняемые программы (с расширениями exe, com);
- загрузочные секторы дисков;
- пакетные командные файлы (bat);
- драйверы устройств;
- библиотеки динамической загрузки (dll), функции из которых вызываются из прикладных программ;
- документы, которые могут содержать **макросы** — небольшие программы, выполняющиеся при нажатии на клавиши или выборе пункта меню; например, макросы нередко используются в документах пакета Microsoft Office;
- веб-страницы (в них можно внедрить программу-**скрипт**, которая выполнится при просмотре страницы на компьютере пользователя).

В отличие от кода программ файлы с данными (например, тексты, рисунки, звуковые и видеофайлы) только обрабатываются, но не выполняются, поэтому заложенный в них код никогда не должен получить управление компьютером. Однако из-за ошибок в программном обеспечении может случиться так, что специально подобранные некорректные данные вызовут сбой программы обработки и выполнение вредоносного кода[1]. Таким образом, существует некоторый шанс, что вредоносная программа, внедрённая в рисунок или видеофайл, все-таки запустится.

Сейчас существуют два основных источника заражения вредоносными программами — флэш-диски и компьютерные сети. **Компьютер может быть заражён при:**

- запуске заражённого файла;
- загрузке с заражённого диска CD/DVD или флэш-диска;
- автозапуске заражённого диска CD/DVD или флэш-диска (вирус автоматически запускается из файла autorun.inf в корневом каталоге диска);
- открытии заражённого документа с макросами;
- открытии сообщения электронной почты с вирусом или запуске заражённой программы, полученной в приложении к сообщению;
- открытии веб-страницы с вирусом;
- установке активного содержимого для просмотра веб-страницы.

Кроме того, есть вирусы-черви, которые распространяются по компьютерным сетям без участия человека. Они могут заразить компьютер даже тогда, когда пользователь не сделал никаких ошибочных действий.

Типы вредоносных программ

К вредоносным программам относятся компьютерные вирусы, черви, троянские программы и др. По **«среде обитания»** обычно выделяют следующие типы вирусов:

- **файловые** — внедряются в исполняемые файлы, системные библиотеки и т. п.;

[1] В 2002 г. был обнаружен вирус, который внедрялся в рисунки формата JPEG. Однако он получал управление только из-за ошибки в системной библиотеке Windows, которая была быстро исправлена.

- **загрузочные** — внедряются в загрузочный сектор диска или в главную загрузочную запись жёсткого диска (англ. **MBR** — *Master Boot Record*); опасны тем, что загружаются в память раньше, чем ОС и антивирусные программы;
- **макровирусы** — поражают документы, в которых могут быть макросы;
- **скриптовые вирусы** — внедряются в командные файлы или в веб-страницы (записывая в них код на языке VBScript или JavaScript);
- **сетевые вирусы** — распространяются по компьютерным сетям.

Некоторые вирусы при создании новой копии немного меняют свой код, для того чтобы их было труднее обнаружить. Такие вирусы называют **полиморфными** (от греч. πολυ — много, μορφη — форма, внешний вид).

Червь — это вредоносная программа, которая распространяется по компьютерным сетям. Наиболее опасны **сетевые черви**, которые используют «дыры» (ошибки в защите, уязвимости) операционных систем и распространяются очень быстро без участия человека. Червь посылает по сети специальный пакет данных — **эксплойт** (англ. *exploit* — эксплуатировать), который позволяет выполнить код на удалённом компьютере и внедриться в систему.

Как правило, вскоре после обнаружения уязвимости выпускается обновление программного обеспечения («заплатка», «патч»); если его установить, то червь становится неопасен. К сожалению, системные администраторы не всегда вовремя устанавливают обновления. Это приводит к эпидемиям сетевых червей, которые по статистике вызывают наибольшее число заражений. Заражённые компьютеры используются для рассылки спама или массовых DoS-атак на сайты в Интернете.

Почтовые черви распространяются как приложения к сообщениям электронной почты. Они представляют собой программы, которые при запуске заражают компьютер и рассылают свои копии по всем адресам из адресной книги пользователя. Из-за этой опасности многие почтовые серверы (например, mail.google.com) не разрешают пересылку исполняемых файлов.

Чтобы заставить пользователя запустить червя, применяются методы **социальной инженерии**: текст сообщения составляется так, чтобы заинтересовать человека и спровоцировать его на

запуск программы, приложенной к письму. В некоторых случаях программа-вирус упакована в архив и защищена паролем, но находится немало людей, которые распаковывают его (пароль указывается в письме) и запускают программу. Часто в почтовых сообщениях содержится только ссылка на сайт, содержащий вирус.

Иногда файл, пришедший как приложение к письму, имеет двойное расширение, например:

```
СуперКартинка.jpg                .exe
```

В самом деле это программа (расширение имени файла exe), но пользователь может увидеть только первые две части имени и попытаться открыть такой «рисунок».

Существуют черви, которые могут распространяться через файлообменные сети, чаты и системы мгновенных сообщений (например, ICQ), но они мало распространены.

Ещё одна группа вредоносных программ — **троянские программы**, или «троянцы» (трояны). Троянский конь — это огромный деревянный конь, которого древние греки подарили жителям Трои во время Троянской войны. Внутри него спрятались воины, которые ночью выбрались, перебили охрану и открыли ворота города. Троянские программы проникают на компьютер под видом «полезных» программ, например кодеков для просмотра видео или экранных заставок. В отличие от вирусов и червей они не могут распространяться самостоятельно и часто «путешествуют» вместе с червями. Среди «троянцев» встречаются:

- **клавиатурные шпионы** — передают «хозяину» все данные, вводимые с клавиатуры (в том числе коды доступа к банковским счетам и т. п.);
- **похитители паролей** — передают пароли, запомненные, например, в браузерах;
- **утилиты удалённого управления** — позволяют злоумышленнику управлять компьютером через Интернет (например, загружать и запускать любые файлы);
- **логические бомбы** — при определённых условиях (дата, время, команда по сети) уничтожают информацию на дисках.

Большинство существующих вирусов написано для ОС Windows, которая установлена более чем на 90% персональных компьютеров.

Известны также вирусы для Mac OS и Linux, но не каждому удается их запустить. Дело в том, что обычный пользователь (не администратор) в этих операционных системах не имеет права на изменение системных файлов, поэтому Mac OS и Linux считают защищёнными от вирусов. Кроме того, вирусы часто полагаются на то, что системные функции размещаются в памяти по определённым адресам. При сборке ядра Linux из исходных кодов эти адреса могут меняться, поэтому вирус, работающий на одном дистрибутиве, может не работать на других.

Вопросы и задания

1. Что такое компьютерный вирус? Чем он отличается от других программ?
2. Что такое вредоносные программы? Какие вредоносные программы вы знаете?
3. Перечислите признаки заражения компьютера вирусом.
4. Какие вредные действия могут совершать вредоносные программы?
5. Какие объекты могут быть заражены вирусами?
6. Какие объекты не заражаются вирусами?
7. При каких действиях пользователя возможно заражение вирусом?
8. Является ли создание и распространение вирусов уголовным преступлением?
9. Какие типы вирусов вы знаете?
10. Что означает сокращение MBR?
11. Чем опасны загрузочные вирусы?
12. Что такое макровирусы? Какие файлы они поражают?
13. Что могут заражать скриптовые вирусы?
14. Что такое полиморфные вирусы? Почему их сложно обнаруживать?
15. Что такое сетевой червь?
16. Что такое эксплойт?
17. Почему необходимо сразу устанавливать обновления для операционных систем?
18. С какими целями могут быть использованы компьютеры, заражённые сетевым червем?
19. Почему многие почтовые серверы запрещают пересылку исполняемых файлов?
20. Что такое социальная инженерия? Как она используется авторами вирусов?
21. Что такое троянские программы? Какие типы троянских программ вы знаете?
22. Какие операционные системы лучше защищены от вирусов? Почему?

Подготовьте сообщение:

а) «Сетевые черви»
б) «Социальная инженерия»
в) «Троянские программы»
г) «Вредоносные программы для Linux и MacOS»
д) «Вредоносные программы и закон»

§ 77
Защита от вредоносных программ

Антивирусные программы

Антивирус — это программа, предназначенная для борьбы с вредоносными программами.

Антивирусы выполняют три основные задачи:

1) не допустить заражение компьютера вирусом;
2) обнаружить присутствие вируса в системе;
3) удалить вирус без ущерба для остальных данных.

Код большинства вирусов содержит характерные цепочки байтов — **сигнатуры** (от лат. *signare* — подписать). Если в файле обнаруживается сигнатура какого-то вируса, можно предположить, что файл заражён. Такой подход используется всеми антивирусными программами. Сигнатуры известных вирусов хранятся в базе данных антивируса, которую нужно регулярно обновлять через Интернет.

Современные антивирусы — это программные комплексы, состоящие из нескольких программ. Чаще всего они включают антивирус-сканер (иногда его называют антивирус-доктор) и антивирус-монитор.

Для того чтобы **антивирус-сканер** начал работу, пользователь должен его запустить и указать, какие файлы и папки нужно проверить. Это «защита по требованию». Сканеры используют два основных метода поиска вирусов:

- **поиск в файлах сигнатур вирусов**, которые есть в базе данных; после обнаружения файл с вирусом можно вылечить, а если это не получилось — удалить;

- **эвристический анализ** (греч. ευρηκα — «нашёл!»), при котором программа ищет в файле код, похожий на вирус.

Эвристический анализ часто позволяет обнаруживать полиморфные вирусы (изменяющие код с каждым новым заражением), но не гарантирует это. Кроме того, случаются ложные срабатывания, когда «чистый» файл попадает под подозрение.

Главный недостаток сканеров состоит в том, что они не могут предотвратить заражение компьютера, потому что начинают работать только при ручном запуске.

Антивирусы-мониторы — это программы постоянной защиты, они находятся в памяти в активном состоянии. Их основная задача — не допустить заражения компьютера и получения заражённых файлов извне. Для этого мониторы:

- проверяют «на лету» все файлы, которые копируются, перемещаются или открываются в различных прикладных программах;
- проверяют используемые флэш-диски;
- перехватывают действия, характерные для вирусов (форматирование диска, замена и изменение системных файлов) и блокируют их;
- проверяют весь поток данных, поступающий из Интернета (сообщения электронной почты, веб-страницы, сообщения ICQ).

Мониторы ведут непрерывное наблюдение, блокируют вирус в момент заражения. Иногда они могут перехватить и неизвестный вирус (сигнатуры которого нет в базе), обнаружив его подозрительные действия.

Главный недостаток антивирусов-мониторов — значительное замедление работы системы, особенно на маломощных компьютерах. Кроме того, мониторы фактически встраиваются в операционную систему, поэтому ошибки разработчиков антивируса могут привести к печальным последствиям (вплоть до удаления системных библиотек и вывода ОС из строя). Бывает и так, что при запущенном мониторе некоторые программы работают неправильно или вообще не работают. Тем не менее не рекомендуется отключать монитор, особенно если вы работаете в Интернете или переносите файлы с помощью флэш-дисков.

Современные антивирусы частично защищают компьютер ещё и от:

- **фишинга** — выманивания паролей для доступа на сайты Интернета с помощью специально сделанных веб-страниц, которые внешне выглядят так же, как «официальные» сайты;
- **рекламных баннеров** и **всплывающих окон** на веб-страницах;
- **спама** — рассылки нежелательных рекламных сообщений по электронной почте.

Большинство антивирусных программ — условно-бесплатные (англ. *shareware*), пробные версии с ограниченным сроком действия можно свободно загрузить из Интернета. Наиболее известны антивирусы AVP (www.kaspersky.ru), DrWeb (www.drweb.com), Nod32 (www.eset.com), McAfee (home.mcafee.com).

На многих сайтах (www.kaspersky.ru, www.freedrweb.com) доступны для скачивания лечащие программы-сканеры, которые бесплатны для использования на домашних компьютерах. В отличие от полных версий в них нет антивируса-монитора, и базы сигнатур не обновляются.

Существуют антивирусы, бесплатные для использования на домашних компьютерах, например Microsoft Security Essentials (www.microsoft.com), Avast Home (www.avast.com), Antivir Personal (free-av.com), AVG Free (free.grisoft.com). Антивирус ClamAV (www.clamav.net) распространяется свободно с исходным кодом.

На сайтах некоторых компаний можно найти **онлайновые антивирусы** (например, http://www.kaspersky.ru/virusscanner). Они устанавливают на компьютер специальный сканирующий модуль и проверяют файлы и оперативную память. Как правило, онлайновые антивирусы могут обнаружить вирусы, но не удаляют их, предлагая приобрести коммерческую версию.

Брандмауэры

Для защиты отдельных компьютеров и сетей от атак из Интернета (в том числе и вирусных) используются **брандмауэры** (*нем.* Brandmauer — стена между зданиями для защиты от распространения огня). Их также называют **сетевыми экранами** или **фаейрволами** (от англ. *firewall* — противопожарная стена). Брандмауэры запрещают передачу данных по каналам связи, которые часто используют вирусы и программы для взлома сетей.

На рисунке 10.1 показана защита одного компьютера с помощью брандмауэра, точно так же защищаются от угроз из Интернета локальные сети.



Рис. 10.1

Брандмауэр входит в состав современных версий ОС Windows, в ядро Linux также включён встроенный брандмауэр Netfilter. Иногда устанавливают дополнительные брандмауэры, например Agnitum Outpost (www.agnitum.com), Kerio Winroute Firewall (kerio.ru) или бесплатную программу Comodo Personal Firewall (www.personalfirewall.comodo.com).

Меры безопасности

Главный вред, который могут нанести вредоносные программы, — это потеря данных или паролей доступа к закрытой информации.

Чтобы уменьшить возможный ущерб, рекомендуется регулярно делать резервные копии важных данных на дисках CD/DVD или флэш-дисках.

Если вы работаете в сети, желательно включать антивирус-монитор и брандмауэр. Монитор сразу сообщит об опасности, если вставленный флэш-диск содержит вирус. Все новые файлы (особенно программы!) нужно проверять с помощью антивируса-сканера.

Не рекомендуется открывать подозрительные сообщения электронной почты, полученные с неизвестных адресов, особенно файлы-приложения (помните про методы социальной инженерии — заинтересовать жертву и заставить запустить программу). Опасно также переходить по ссылкам в тексте писем, с большой вероятностью они ведут на сайты, зараженные вирусами.

Если компьютер заражён, нужно отключить его от компьютерной сети и запустить антивирус-сканер. Очень часто это позволяет удалить вирус, если его сигнатура есть в базе данных. Если антивирус не был установлен раньше, можно попробовать установить его на заражённый компьютер, но это не всегда приводит к успеху (вирус может блокировать установку антивируса).

Если антивирус-сканер не обнаруживает вирус или не может его удалить, можно попытаться (желательно с другого компьютера) найти в Интернете бесплатную утилиту для лечения с новыми базами сигнатур. Например, утилита CureIt (www.freedrweb.com) не требует установки и может быть запущена с флэш-диска. Даже если удалить вирус не удастся, скорее всего, он будет обнаружен, и программа покажет его название. Следующий шаг — искать в Интернете утилиту для удаления именно этого вируса (например, ряд утилит можно найти на сайте support.kaspersky.ru).

В особо тяжёлых случаях для уничтожения вирусов приходится полностью форматировать жёсткий диск компьютера, при этом все данные теряются.

Вопросы и задания

1. Что такое антивирус? Какие задачи он решает?
2. Что такое сигнатура?
3. Почему нужно регулярно обновлять базы сигнатур антивирусов?
4. Чем отличается антивирус-сканер от антивируса-монитора?
5. Что значит «защита по требованию»?
6. Что такое эвристический анализ? В чём его достоинства и недостатки?
7. Какие функции у антивируса-монитора? Каковы его недостатки?
8. Что такое фишинг?
9. Что такое спам?
10. Какие ограничения есть у пробных версий коммерческих антивирусов?
11. Что такое онлайновый антивирус?
12. Что такое брандмауэр? Зачем он нужен?
13. В чём заключается основной вред, наносимый вирусами? Как можно уменьшить возможные потери?
14. Как можно улучшить безопасность компьютера при работе в сети Интернет?
15. Какие меры безопасности необходимы при работе с электронной почтой?
16. Какие действия можно предпринять, если компьютер заражен вирусом?

Подготовьте сообщение

а) «Бесплатное антивирусное программное обеспечение»
б) «Онлайновые антивирусы»
в) «Настройка брандмауэра»

§ 78 Шифрование

Один из методов защиты информации от неправомерного доступа — это шифрование, т. е. кодирование специального вида.

Шифрование — это преобразование (кодирование) открытой информации в зашифрованную, недоступную для понимания посторонними.

Шифрование применяется в первую очередь для передачи секретной информации по незащищённым каналам связи. Шифровать можно любые данные — тексты, рисунки, звук, базы данных и т. д.

Человечество применяет шифрование с того момента, как появилась секретная информация, которую нужно было скрыть от врагов. Первое известное науке шифрованное сообщение — египетский текст, в котором вместо принятых тогда иероглифов были использованы другие знаки.

Методы шифрования и расшифровывания сообщения изучает наука **криптология**, история которой насчитывает около четырех тысяч лет. Она состоит из двух ветвей: криптографии и криптоанализа.

Криптография — это наука о способах шифрования информации.
Криптоанализ — это наука о методах и способах вскрытия шифров.

Обычно предполагается, что сам алгоритм шифрования известен всем, но неизвестен его ключ, без которого сообщение невозможно расшифровать. В этом заключается отличие шифрования от простого кодирования, при котором для восстановления сообщения достаточно знать только алгоритм кодирования.

Ключ — это параметр алгоритма шифрования (шифра), позволяющий выбрать одно конкретное преобразование из всех вариантов, предусмотренных алгоритмом. Знание ключа позволяет свободно зашифровывать и расшифровывать сообщения.

Все шифры (системы шифрования) делятся на две группы — симметричные и несимметричные (с открытым ключом).

Симметричный шифр означает, что и для шифрования, и для расшифровывания сообщений используется один и тот же ключ. В системах **с открытым ключом** используются два ключа — открытый и закрытый, которые связаны друг с другом с помощью некоторых математических зависимостей. Информация шифруется с помощью открытого ключа, который доступен всем желающим, а расшифровывается с помощью закрытого ключа, известного только получателю сообщения.

Криптостойкость шифра — это устойчивость шифра к расшифровке без знания ключа.

Стойким считается алгоритм, который для успешного раскрытия требует от противника недостижимых вычислительных ресурсов, недостижимого объёма перехваченных сообщений или такого времени, что по его истечении защищённая информация будет уже не актуальна.

Шифр Цезаря[1] — один из самых известных и самых древних шифров. В этом шифре каждая буква заменяется на другую, расположенную в алфавите на заданное число позиций k вправо от неё. Алфавит замыкается в кольцо, так что последние символы заменяются на первые. На рисунке 10.2 — пример шифра Цезаря (со сдвигом 3)[2].

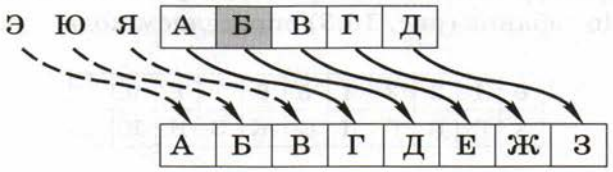


Рис. 10.2

[1] Назван в честь римского императора Гая Юлия Цезаря, использовавшего его для секретной переписки.

[2] Будем считать, что буквы «Е» и «Ё» совпадают.

Знаменитая фраза «ПРИШЕЛ УВИДЕЛ ПОБЕДИЛ» при использовании шифра Цезаря со сдвигом 3 будет закодирована так:

ТУЛЫИО ЦЕЛЗИО ТСДИЗЛО

Если первая буква алфавита имеет код 0, вторая — код 1 и т. д., алгоритм шифрования может быть выражен формулой

$$y = (x + k) \bmod n,$$

где x — код исходного символа, k — величина сдвига, y — код символа-замены, n — количество символов в алфавите, а запись $(x + k) \bmod n$ обозначает остаток от деления $x + k$ на n. Операция взятия остатка от деления необходима для того, чтобы замкнуть алфавит в кольцо. Например, при использовании русского алфавита (32 буквы, без буквы «Ё») для буквы «Я» (код 31) получаем код заменяющего символа $(31 + 3) \bmod 32 = 2$, это буква «В».

Ключом для шифра Цезаря служит сдвиг k, если его знать, то сообщение легко расшифровать. Для этого используется формула

$$x = (y - k + n) \bmod n.$$

Шифр Цезаря относится к шифрам **простой подстановки**, так как каждый символ исходного сообщения заменяется на другой символ из того же алфавита. Такие шифры легко раскрываются с помощью частотного анализа, потому что в каждом языке частоты встречаемости букв примерно постоянны для любого достаточно большого текста.

Значительно сложнее сломать **шифр Виженера**[1], который стал естественным развитием шифра Цезаря. Для использования шифра Виженера используется ключевое слово, которое задает переменную величину сдвига. Например, пусть ключевое слово — «ЗАБЕГ». По таблице (рис. 10.3) определяем коды букв.

0	1	2	3	4	5	6	7	8	9	
А	Б	В	Г	Д	Е	Ж	З	И	К	...

Рис. 10.3

Получаем: «З» — 7, «А» — 0, «Б» — 1, «Е» — 5, «Г» — 3. Это значит, что для кодирования первой буквы любого текста используется сдвиг 7, для кодирования второй — 0 (символ не меняется) и т. д. Для пятой буквы используется сдвиг 3, а для шес-

[1] Назван по имени *Блеза Виженера*, швейцарского дипломата XVI века.

той — снова 7 (начинаем «проходить» кодовое слова с начала). Фраза «ПРИШЕЛ УВИДЕЛ ПОБЕДИЛ» при использовании шифра Виженера с ключом «ЗАБЕГ» будет закодирована в виде «ЦРЙЭИТ УГНЗМЛ РУДМДЙР».

Шифр Виженера обладает значительно более высокой криптостойкостью, чем шифр Цезаря. Это значит, что его труднее раскрыть — подобрать нужное ключевое слово. Теоретически, если длина ключа равна длине сообщения и каждый ключ используется только один раз, шифр Виженера обладает абсолютной криптостойкостью — взломать его невозможно.

Вопросы и задания

1. Чем различаются понятия «шифрование» и «кодирование»?
2. Что такое ключ?
3. Как называется наука, изучающая методы шифрования?
4. Что такое симметричный шифр? Какая проблема возникает при использовании симметричного шифра, если участники переписки находятся в разных странах?
5. Что такое несимметричные шифры? На чём основана их надёжность?
6. Что такое криптостойкость алгоритма? Какой алгоритм считается криптостойким?

Подготовьте сообщение

а) «Полиграммные шифры подстановки»
б) «Шифрование и закон»
в) «Криптостойкость шифров»
г) «Частотный анализ»

Задачи

1. Зашифруйте с помощью шифра Цезаря со сдвигом 6 высказывание «ЛЮДИ ОХОТНО ВЕРЯТ ТОМУ, ЧЕМУ ЖЕЛАЮТ ВЕРИТЬ».
2. Напишите программу, которая выполняет шифрование строки с помощью шифра Цезаря.

*3. Попытайтесь расшифровать сообщение, закодированное шифром Цезаря с неизвестным сдвигом: «ХШЖНПУТ ФКХКОЙКТ». Для этого можно написать программу.

4. Используя шифр Виженера с ключом «ЛЕНА», зашифруйте сообщение «НЕЛЬЗЯ ОБИЖАТЬ ГОСТЯ».

*5. Измените программу для шифрования с помощью шифра Цезаря так, чтобы учитывать букву Ё.

*6. Измените программу для шифрования с помощью шифра Цезаря так, чтобы учитывать и букву Ё, и пробел.

§ 79
Хэширование и пароли

В современных информационных системах часто используется вход по паролю. Если при этом где-то хранить пароли всех пользователей, система становится очень ненадёжной, потому что «утечка» паролей позволит сразу получить доступ к данным. Вместе с тем кажется, что пароли обязательно где-то нужно хранить, иначе пользователи не смогут войти в систему. Однако это не совсем так. Можно хранить не пароли, а некоторые числа, полученные в результате обработки паролей. Простейший вариант — сумма кодов символов, входящих в пароль. Для пароля «A123» такая сумма равна

215 = 65 (код «A») + 49 (код «1») + 50 (код «2») + 51 (код «3»).

Фактически мы определили функцию $H(M)$, которая сообщение M *любой* длины превращает в короткий код m. Такую функцию называются **хэш-функцией** (от англ. *hash* — мешанина, крошить), а само полученное число — **хэш-кодом**, **хэш-суммой** или просто **хэшем** исходной строки. Важно, что, зная хэш-код, невозможно восстановить исходный пароль! В этом смысле **хэширование** — это *необратимое* шифрование.

Итак, вместо пароля «A123» мы храним число 215. Когда пользователь вводит пароль, мы считаем сумму кодов символов этого пароля и разрешаем вход в систему только тогда, когда она равна 215. И вот здесь возникает проблема: существует очень много паролей, для которых наша хэш-функция даёт значение 215, например «B023». Такая ситуация — совпадение хэш-кодов различных исходных строк — называется **коллизией** (англ. *collision* — столкновение). Коллизии будут всегда — ведь мы «сжимаем» длинную цепочку байтов до числа. Казалось бы, ничего хорошего не получилось: если взломщик узнает хэш-код, то, зная алгоритм его получения, он сможет легко подобрать пароль с таким же хэшем и получить доступ к данным. Однако это произошло потому, что мы выбрали плохую хэш-функцию.

Математики разработали надёжные (но очень сложные) хэш-функции, обладающие особыми свойствами:

1) хэш-код очень сильно меняется при малейшем изменении исходных данных;

2) при известном хэш-коде m невозможно за приемлемое время найти сообщение M с таким хэш-кодом ($H(M) = m$);
3) при известном сообщении M невозможно за приемлемое время найти сообщение M_1 с таким же хэш-кодом ($H(M) = H(M_1)$).

Здесь выражение «невозможно за приемлемое время» (или «вычислительно невозможно») означает, что эта задача решается только перебором вариантов (других алгоритмов не существует), а количество вариантов настолько велико, что на решение могут уйти сотни и тысячи лет. Поэтому даже если взломщик получил хэш-код пароля, он не сможет за приемлемое время получить сам пароль (или пароль, дающий такой же хэш-код).

Чем длиннее пароль, тем больше количество вариантов. Кроме длины для надёжности пароля важен используемый набор символов. Например, очень легко подбираются пароли, состоящие только из цифр. Если же пароль состоит из 10 символов и содержит латинские буквы (заглавные и строчные) и цифры, перебор вариантов (англ. *brute force* — метод «грубой силы») со скоростью 10 млн паролей в секунду займет более 2000 лет.

Надёжные пароли должны состоять не менее чем из 7–8 символов; пароли, состоящие из 15 символов и более, взломать методом «грубой силы» практически невозможно. Не используйте пароли типа «12345», «qwerty», свой день рождения, номер телефона. Плохо, если пароль представляет собой известное слово, для этих случаев взломщики используют подбор по словарю. Сложнее всего подобрать пароль, который представляет собой случайный набор заглавных и строчных букв, цифр и других знаков[1].

Сегодня для хэширования в большинстве случаев применяют алгоритмы MD5, SHA1 и российский алгоритм, изложенный в ГОСТ Р34.11-94 (он считается одним из самых надёжных). В криптографии хэш-коды чаще всего имеют длину 128, 160 и 256 битов.

Хэширование используется также для проверки правильности передачи данных: различные контрольные суммы — это не что иное, как хэш-коды.

[1] Однако такой пароль сложно запомнить.

Вопросы и задания

1. Что такое хэширование? Хэш-функция? Хэш-код?
2. Какую хэш-функцию вы используете, когда начинаете искать слово в словаре?
3. Что такое коллизии? Почему их должно быть как можно меньше?
4. Какие требования предъявляются к хэш-функциям, которые используются при хранении паролей?
5. Что значит «вычислительно невозможно»?
6. Взломщик узнал хэш-код пароля администратора сервера. Сможет ли он получить доступ к секретным данным на сервере?
7. Какие свойства пароля влияют на его надёжность?
8. Как выбрать надёжный пароль?
9. Какие алгоритмы хэширования сейчас чаще всего применяются?
10. Предложите какой-нибудь свой метод хэширования. Подумайте, как часто при его использовании могут происходить коллизии.

Подготовьте сообщение

а) «Где используют хэши?»
б) «Хэширование и передача данных»
в) «Хэширование с "солью"»

Задачи

1. Напишите программу, которая запрашивает длину пароля, количество используемых символов и скорость перебора (например, в миллионах вариантов в секунду) и находит время, необходимое для подбора пароля методом «грубой силы».
2. Компьютер выполняет перебор со скоростью 10 млн паролей в секунду. С помощью программы (см. задачу 1) определите, какова должна быть длина пароля, чтобы время его взлома составило не менее 1 месяца, если:
   а) пароль состоит только из цифр;
   б) пароль состоит только из заглавных латинских букв и цифр;
   в) пароль состоит только из латинских букв (заглавных и строчных) и цифр.

§ 80
Современные алгоритмы шифрования

Государственным стандартом шифрования в России является алгоритм, зарегистрированный как ГОСТ 28147-89. Он является **блочным шифром**, т. е. шифрует не отдельные символы, а 64-бит-

ные блоки. В алгоритме предусмотрены 32 цикла преобразования данных с 256-битным ключом, за счёт этого он очень надёжен (обладает высокой криптостойкостью). На современных компьютерах раскрытие этого шифра путём перебора ключей («методом грубой силы») займёт не менее сотен лет, что делает такую атаку бессмысленной.

В США в качестве стандарта принят блочный шифр AES (*англ.* Advanced Encryption Standard, передовой стандарт шифрования), выбранный в 2001 г. по результатам проведённого конкурса. Шифр AES используется также в защищённых беспроводных сетях (WiFi).

В Интернете популярен алгоритм RSA, названный так по начальным буквам фамилий его авторов — Р. Райвеста (R. Rivest), А. Шамира (A. Shamir) и Л. Адлемана (L. Adleman). Это алгоритм *с открытым ключом*, стойкость алгоритма основана на том, что перемножить два очень больших простых числа достаточно просто, а вот разложить такое произведение на простые сомножители очень трудно (эту задачу сейчас умеют решать только перебором вариантов). Поскольку количество вариантов огромно, для раскрытия шифра требуется много лет работы современных компьютеров.

Для применения алгоритма RSA требуется построить открытый и секретный ключи следующим образом.

1. Выбрать два больших простых числа, p и q.
2. Найти их произведение $n = p \cdot q$ и значение $\varphi = (p-1)\cdot(q-1)$.
3. Выбрать число e $(1 < e < \varphi)$, которое не имеет общих делителей с φ.
4. Найти число d, которое удовлетворяет условию $d \cdot e = k\varphi + 1$ для некоторого целого k.
5. Пара значений (e, n) — это открытый ключ RSA (его можно свободно публиковать), а пара (d, n) — это секретный ключ.

Передаваемое сообщение нужно сначала представить в виде последовательности чисел в интервале от 0 до $n - 1$. Для шифрования используют формулу

$$y = x^e \bmod n,$$

где x — исходное сообщение (число), (e,n) — открытый ключ, y — закодированное сообщение (число), а запись $x^e \bmod n$ обозначает остаток от деления x^e на n. Расшифровка сообщения выполняется по формуле

$$x = y^d \bmod n.$$

Это значит, что зашифровать сообщение может каждый (открытый ключ общеизвестен), а прочитать его — только тот, кто знает секретный показатель степени d.

Для лучшего понимания мы покажем работу алгоритма RSA на простом примере. Возьмём $p=3$ и $q=7$, тогда находим $n=p\cdot q=21$ и $\varphi=(p-1)\cdot(q-1)=12$. Выберем $e=5$, тогда равенство $d\cdot e=k\varphi+1$ выполняется, например, при $d=17$ (и $k=7$). Таким образом, мы получили открытый ключ (5, 21) и секретный ключ (17, 21).

Зашифруем сообщение «123» с помощью открытого ключа (5, 21). Получаем:

$$1\Rightarrow 1^5 \bmod 21 = 1,\quad 2\Rightarrow 2^5 \bmod 21 = 11,\quad 3\Rightarrow 3^5 \bmod 21 = 12,$$

т. е. зашифрованное сообщение состоит из чисел 1, 11 и 12. Зная секретный ключ (17, 21), можно его расшифровать:

$$1\Rightarrow 1^{17} \bmod 21 = 1,\quad 11\Rightarrow 11^{17} \bmod 21 = 2,\quad 12\Rightarrow 12^{17} \bmod 21 = 3.$$

Мы получили исходное сообщение.

Конечно, вы заметили, что при шифровании и расшифровке приходится вычислять остаток от деления очень больших чисел (например, 12^{17}) на n. Оказывается, само число 12^{17} в этом случае находить не нужно. Достаточно записать в обычную целочисленную переменную, например x, единицу, а потом 17 раз выполнить преобразование $x = 12 \cdot x \bmod 21$. После этого в переменной x будет значение $12^{17} \bmod 21 = 3$. Попробуйте доказать правильность этого алгоритма.

Чтобы взломать шифр, злоумышленнику надо узнать секретный показатель степени d. А для этого необходимо найти большие простые числа p и q, такие что $n=p\cdot q$. Если n велико, это очень сложная задача, её решение перебором вариантов на современном компьютере займёт сотни лет. В 2009 г. группа учёных из разных стран в результате многомесячных расчётов на сотнях компьютеров смогла расшифровать сообщение, зашифрованное алгоритмом RSA с 768-битным ключом. Поэтому сейчас надёжными считаются ключи с длиной 1024 бита и более. Если будет построен работающий квантовый компьютер, взлом алгоритма RSA будет возможен за очень небольшое время.

При использовании симметричных шифров всегда возникает проблема: как передать ключ, если канал связи ненадёжный? Ведь, получив ключ, противник сможет расшифровать все дальнейшие сообщения. Для алгоритма RSA этой проблемы нет, сто-

ронам достаточно обменяться открытыми ключами, которые можно показывать всем желающим.

У алгоритма RSA есть ещё одно достоинство: его можно использовать для цифровой подписи сообщений. Она служит для доказательства авторства документов, защиты сообщений от подделки и умышленных изменений.

Цифровая подпись — это набор символов, который получен в результате шифрования сообщения с помощью личного секретного кода отправителя.

Отправитель может передать вместе с исходным сообщением такое же сообщение, зашифрованное с помощью своего секретного ключа (это и есть цифровая подпись). Получатель расшифровывает цифровую подпись с помощью открытого ключа. Если она совпала с незашифрованным сообщением, можно быть уверенным, что его отправил тот человек, который знает секретный код. Если сообщение было изменено при передаче, оно не совпадёт с расшифрованной цифровой подписью. Так как сообщение может быть очень длинным, для сокращения объёма передаваемых данных чаще всего шифруется не всё сообщение, а только его хэш-код.

Во многих современных программах есть возможность шифровать данные с паролем. Например, офисные пакеты OpenOffice.org и Microsoft Office позволяют шифровать все создаваемые документы (для их просмотра и/или изменения нужно ввести пароль). При создании архива (например, в архиваторах 7ZIP, WinRAR, WinZip) также можно установить пароль, без которого извлечь файлы невозможно.

В простейших задачах для шифрования файлов можно использовать бесплатную программу Шифровальщик (www.familytree.ru/ru/cipher.htm), версии которой существуют для Linux и Windows. Программы TrueCrypt (www.truecrypt.org), BestCrypt (www.jetico.com) и FreeOTFE (freeotfe.org) создают логические диски-контейнеры, информация на которых шифруется. Свободно распространяемая программа DiskCryptor (diskcryptor.net) позволяет шифровать разделы жёстких дисков и даже создавать шифрованные флэш-диски и диски CD/DVD.

Программа GnuPG (gnupg.org) также относится к свободному программному обеспечению. В ней поддерживаются симметричные и несимметричные шифры, а также различные алгоритмы электронной цифровой подписи.

Вопросы и задания

1. Какой алгоритм шифрования принят в России в качестве государственного стандарта?
2. Что такое блочный алгоритм шифрования?
3. К какому типу относится алгоритм RSA? На чем основана его криптостойкость?
4. Что такое цифровая подпись?
5. Как можно использовать алгоритм RSA для цифровой подписи?

Подготовьте сообщение

а) «Стандарты шифрования разных стран»
б) «Шифрование с открытым ключом: за и против»
в) «Алгоритм Эль-Гамаля»
г) «Цифровая подпись в Российской Федерации»

Задачи

*1. Напишите программу, которая строит открытый и секретный ключи RSA для небольших множителей p и q.
*2. Напишите программу, которая шифрует и расшифровывает сообщения с помощью алгоритма при небольших значениях открытого и секретного ключей.

§ 81 Стеганография

При передаче сообщений можно не только применять шифрование, но и скрывать сам факт передачи сообщения.

Стеганография — это наука о скрытой передаче информации путём скрытия самого факта передачи информации.

Древнегреческий историк Геродот описывал, например, такой метод: на бритую голову раба записывалось сообщение, а когда его волосы отрастали, он отправлялся к получателю, который брил его голову и читал сообщение.

Классический метод стеганографии — *симпатические* (невидимые) *чернила*, которые проявляются только при определенных условиях (нагрев, освещение, химический проявитель). Например, текст, написанный молоком, можно прочитать при нагреве.

Сейчас стеганография занимается *скрытием информации в текстовых, графических, звуковых и видеофайлах* с помощью программного «внедрения» в них нужных сообщений.

Простейшие способ — заменять младшие биты файла, в котором закодировано изображение. Причём это нужно сделать так, чтобы разница между исходным и полученным рисунками была неощутима для человека. Например, в чёрно-белом рисунке (256 оттенков серого) яркость каждого пикселя кодируется 8 битами. Если поменять 1–2 младших бита этого кода, «встроив» туда текстовое сообщение, фотография, в которой нет чётких границ, почти не изменится. При замене одного бита каждый байт исходного текстового сообщения хранится в младших битах кодов 8 пикселей. Например, пусть первые 8 пикселей рисунка имеют такие коды:

10101101	10010100	00101010	01010010	10101010	10101010	10101011	10101111

Чтобы закодировать в них код буквы «И» (11001000_2), нужно изменить младшие биты кодов:

10101101	10010101	00101010	01010010	10101011	10101010	10101010	10101110
1	1	0	0	1	0	0	0

Получателю нужно взять эти младшие биты и «собрать» их вместе в один байт.

Для звуков используются другие методы стеганографии, основанные на добавлении в запись коротких условных сигналов, которые обозначают 1 и 0 и не воспринимаются человеком на слух. Возможна также замена одного фрагмента звука на другой.

Для подтверждения авторства и охраны авторских прав на изображения, видео и звуковые файлы применяют *цифровые водяные знаки* — внедрённую в файл информацию об авторе. Они получили своё название от старых водяных знаков на деньгах и документах. Для того чтобы установить авторство фотографии, достаточно расшифровать скрытую информацию, записанную с помощью водяного знака.

Иногда цифровые водяные знаки делают видимыми (текст или логотип компании на фотографии или на каждом кадре видеофильма). На многих сайтах, занимающихся продажей цифровых фотографий, видимые водяные знаки размещены на фотографиях, предназначенных для предварительного просмотра.

Вопросы и задания

1. Что такое стеганография?
2. Какие методы стеганографии существовали до изобретения компьютеров?
3. Как можно добавить текст в закодированное изображение?
4. На чем основаны методы стеганографии для звуковых данных и видеоданных?
5. Что такое цифровые водяные знаки? Зачем они используются?

Подготовьте сообщение

«Водяные знаки в цифровую эпоху»

§ 82
Безопасность в Интернете

Угрозы безопасности

Если компьютер подключён к Интернету, появляются дополнительные угрозы безопасности. Атаку через сеть могут проводить злоумышленники и **боты** (программы-роботы), находящиеся в других городах и странах. Можно выделить три основные **цели злоумышленников**:

- **использование вашего компьютера** для взлома других компьютеров, атак на сайты, рассылки спама, подбора паролей и т. п.;
- **кража секретной информации** — данных о банковских картах, имён и паролей для входа на почтовые серверы, в социальные сети, платёжные системы;
- **мошенничество** — хищение чужого имущества путём обмана.

Первые две угрозы связаны, главным образом, с вредоносными программами: вирусами, червями и «троянцами», которые позволяют злоумышленнику управлять компьютером через сеть и получать с него данные.

Мошенничество процветает потому, что многие пользователи Интернета очень доверчивы и неосторожны. Классический пример мошенничества — так называемые **нигерийские письма**, приходящие по электронной почте. Пользователя от имени какого-то бывшего высокопоставленного лица просят принять участие в переводе крупных денежных сумм за границу, обещая выплачивать большие проценты. Если получатель соглашается, мошенники постепенно выманивают у него деньги.

Фишинг (англ. *phishing*, искажение слова *fishing* — рыбная ловля) — это выманивание паролей. Для этого чаще всего используются сообщения электронной почты, рассылаемые якобы от имени администраторов банков, платёжных систем, почтовых служб, социальных сетей. В сообщении говорится, что ваш счёт (или учётная запись) заблокирован, и даётся ссылка на сайт, который внешне выглядит как настоящий, но расположен по другому адресу (это можно проверить в адресной строке браузера). Неосторожный пользователь вводит своё кодовое имя и пароль, с помощью которых мошенник получает доступ к данным или банковскому счёту.

Антивирусы и последние версии браузеров содержат специальные модули для обнаружения подозрительных сайтов (*«антифишинг»*) и предупреждают о заходе на такой сайт. Кроме того, нужно помнить, что администраторы сервисов никогда не просят пользователя сообщить свой пароль по электронной почте.

Мошенничество может быть связано и с вредоносными программами. В 2010 г. несколько миллионов компьютеров в России было заражено троянской программой Winlock, которая блокировала компьютер и требовала отправить платное SMS-сообщение для снятия блокировки.

Правила личной безопасности

Вредоносные программы, распространяющиеся через Интернет, представляют серьёзную угрозу безопасности данных. Нужно помнить, что многих проблем можно избежать, если работать в Интернете только из-под ограниченной учётной записи (без прав администратора). Кроме того, желательно своевременно обновлять программное обеспечение; особенно важно устанавливать «заплатки», связанные с безопасностью.

Чтобы ваши пароли не украли, лучше не запоминать их в браузере (иногда они хранятся в открытом виде и могут быть украдены троянской программой). Заходя под своим именем в закрытую зону сайта с другого компьютера, нужно отмечать флажок **Чужой компьютер**, иначе тот, кто после вас откроет эту страницу на вашем компьютере, сможет получить доступ к вашим данным.

На многих сайтах предусмотрена возможность восстановления пароля по секретному вопросу. Этот вопрос нужно выбирать так, чтобы никто другой не знал ответа на него и, самое важное, не мог его выведать. Например, ответы на вопросы «Как звали вашу первую собаку?», «Какое ваше любимое блюдо?» и т. п. часто можно найти на персональных страничках авторов в социальных

сетях (в заметках, подписях к фотографиям и т. п.). Если мама автора имеет свою страничку, на ней, скорее всего, можно найти её девичью фамилию, поэтому вопрос «Какова девичья фамилия вашей матери?» тоже лучше не использовать.

Нужно понимать, что, размещая какую-то информацию в Интернете, вы делаете её доступной для широкого круга лиц, включая работодателей, милицию, официальные органы и даже преступников. Возможны ситуации, когда эта информация (личные данные, фотографии, высказывания на форумах и в блогах) может быть использована против вас, даже если она находится в закрытом разделе сайта.

Для передачи информации, которую необходимо сохранить в тайне, лучше применять шифрование (например, упаковать данные в архив с паролем).

Наибольший уровень безопасности обеспечивается при денежных расчётах через Интернет: вместо протокола HTTP используют защищенный протокол HTTPS (англ. *Hypertext Transfer Protocol Secure* — безопасный HTTP), который предусматривает шифрование данных (например, с помощью алгоритма RSA). Поэтому нужно проверять, чтобы адрес на странице ввода пароля в таких системах начинался с https://, а не с http://.

Современные молодые люди часто общаются в чатах, форумах и т. п., в том числе с теми, кого они не знают лично. Продолжение такого *виртуального* (компьютерного, электронного) знакомства в реальной жизни весьма опасно, потому что нередко участники чатов и форумов представляются не теми, кем они являются на самом деле.

Вопросы и задания

1. Какие угрозы безопасности существуют при подключении к Интернету?
2. Какие схемы интернет-мошенничества вам известны?
3. Какие меры безопасности нужно соблюдать при работе в Интернете?
4. Как обеспечивается безопасность обмена данными при денежных расчётах в Интернете?

Подготовьте сообщение

а) «Нигерийские письма»
б) «Фишинг»
в) «Безопасность финансовых расчётов в Интернете»

Практические работы к главе 10

Работа № 73 «Использование антивирусных программ»
Работа № 74 «Простые алгоритмы шифрования данных»
Работа № 75 «Современные алгоритмы шифрования и хэширования»
Работа № 76 «Использование стеганографии»

ЭОР к главе 10 на сайте ФЦИОР (http//fcior.edu.ru)

- Организация защиты при работе в сети
- Компьютерные вирусы и антивирусные программы
- Методы и средства защиты программных продуктов
- Сетевые и интернет-технологии. Компьютерная безопасность
- Информационная безопасность

Самое важное в главе 10

- Информационная безопасность — это защищённость информации от любых действий, в результате которых может быть искажена или утеряна информация либо нарушена её конфиденциальность, а владельцам или пользователям информации нанесён недопустимый ущерб.
- Основные угрозы информационной безопасности — выход оборудования из строя, неправомерный доступ к информации, вредоносные программы.
- Слабое звено любой системы информационной защиты — это человек.
- Шифрование — это преобразование открытой информации в зашифрованную, недоступную для понимания посторонних.
- Существуют системы шифрования с открытым ключом, при использовании которых все данные могут передаваться по незащищённому каналу связи.
- Цифровая подпись — это набор символов, который получен в результате шифрования сообщения с помощью личного секретного кода отправителя.

Навигационные значки

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) электронные образовательные ресурсы (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%2000/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.

Для заметок

Для заметок

Для заметок

Для заметок