К. Ю. Поляков, Е. А. Еремин

ИНФОРМАТИКА

10 класс

(базовый и углублённый уровни)

(в 2 частях)

Учебник

Часть 2

Рекомендовано
к использованию при реализации имеющих государственную аккредитацию образовательных программ начального общего, основного общего, среднего общего образования

Москва
БИНОМ. Лаборатория знаний
2019

УДК 004.9
ББК 32.97
П54

Поляков К. Ю.

П54 Информатика (базовый и углублённый уровни) (в 2 частях). 10 класс. Ч. 2 : учебник / К. Ю. Поляков, Е. А. Еремин. — М. : БИНОМ. Лаборатория знаний, 2019. — 352 с. : ил.

ISBN 978-5-9963-4589-2 (Ч. 2)
ISBN 978-5-9963-4590-8

Учебник предназначен для изучения информатики на базовом и углублённом уровнях в 10 классах общеобразовательных организаций. 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, включающий также учебник для 11 класса, методическое пособие и задачник.

Соответствует федеральному государственному образовательному стандарту среднего общего образования и примерной основной образовательной программе среднего общего образования.

УДК 004.9
ББК 32.97

ISBN 978-5-9963-4589-2 (Ч. 2)
ISBN 978-5-9963-4590-8

Глава 6
ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ (§§ 39–43)

§ 39
Пакеты прикладных программ

Ключевые слова:

- пакет прикладных программ
- офисный пакет
- графический редактор
- 3D-моделирование
- рендеринг
- настольно-издательская система
- вёрстка
- система автоматизированного проектирования

Пакет прикладных программ (ППП) — это набор программ для решения некоторого класса задач.

Офисные пакеты

Самые известные офисные пакеты — *Microsoft Office* и *OpenOffice* (а также его вариант *LibreOffice*). Пакет *Microsoft Office* — *коммерческий*, а *OpenOffice* можно установить и использовать бесплатно. Кроме того, *OpenOffice* — это *кроссплатформенное программное обеспечение*: существуют версии этого пакета для операционных систем *Windows*, *Linux* и *macOS*.

В состав любого офисного пакета входит **текстовый процессор**. В пакетах *OpenOffice* и *LibreOffice* он называется *Writer*, а в пакете *Microsoft Office* — *Word*.

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* включён табличный процессор *Excel*, а в пакете *OpenOffice* есть близкая по возможностям программа *OpenOffice Calc*.

Компьютерная презентация (лат. *praesentatio* — представление) — это набор изображений (*слайдов*), который предназначен для иллюстрации доклада или выступления. Задача презентации — улучшить восприятие информации. Для создания презентаций в пакете *Microsoft Office* используют программу *Microsoft PowerPoint*, а в пакете *OpenOffice* — программу *OpenOffice 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 *Access*, а в пакет *OpenOffice* — программа *OpenOffice Base*, которая работает с СУБД *HSQLDB*.

Программы для управления предприятием

В работе предприятия, особенно крупного, очень важно обеспечить надёжность работы и совместимость используемых программ и баз данных. В современных компаниях используются пакеты прикладных программ для:

- управления предприятием;
- бухгалтерского учёта;
- управления персоналом (кадрового учёта);
- управления перевозками сырья и товаров и других задач.

В России разработкой высококачественных программ этого класса занимается компания *1С* (**1с.ru**).

Пакеты для решения научных задач

Для учёных, работающих в области точных наук (математиков, физиков, астрономов), компьютер стал незаменимым помощником: без него выполнить обработку огромных массивов данных было бы практически нереально.

Пакет прикладных программ *MATLAB* (от англ. *Matrix Laboratory* — «матричная лаборатория») используют более миллиона инженеров и научных работников во всем мире. Этот пакет работает на большинстве современных операционных систем, включая *Windows, macOS и Linux*. Он позволяет не только использовать готовые алгоритмы, но и программировать на

специальном языке в удобной интегрированной среде разработки программ. Первоначально *MATLAB* разрабатывался как система проектирования систем управления, но сейчас используется в самых разных областях.

Пакет *MATLAB* — это коммерческое программное обеспечение. Но существуют бесплатные пакеты, относящиеся к свободному программному обеспечению, которые решают практически те же задачи: *GNU Octave* и *Scilab*. На рисунке 6.37 показано окно системы *Scilab*. Пакеты *MATLAB*, *GNU Octave* и *Scilab* предназначены для выполнения численных расчётов.

Рис. 6.37

Если нужно получить решение математической задачи в аналитическом виде (в виде формулы, «в буквах»), применяют системы символьных вычислений (системы компьютерной алгебры): *Matematica*, *MathCAD*, *Maple*. На рисунке 6.38 показано окно системы *Maple*.

Рис. 6.38

Программы для дизайна и вёрстки

Графические редакторы — это программы для создания и редактирования изображений. Существуют отдельные программы для редактирования растровых и векторных рисунков, которые часто называют просто растровыми и векторными графическими редакторами.

Растровые редакторы предназначены для:

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

Растровый редактор *Paint* — стандартное приложение *Windows*, но для сложной обработки (например, для улучшения качества фотографий) его возможностей недостаточно.

Среди профессионалов популярен растровый редактор *Adobe Photoshop*. Он работает в операционных системах *Windows* и *macOS*.

Бесплатный графический редактор *Gimp* (рис. 6.39) — кроссплатформенный, разработаны его версии для *Windows*, *Linux* и *macOS*.

Рис. 6.39

Существуют бесплатные онлайновые редакторы — специальные сайты в Интернете, на которых можно редактировать рисунки, не устанавливая себе на компьютер никаких программ.

Векторные редакторы используются для подготовки:

- художественных иллюстраций;
- чертежей, схем, графиков;
- логотипов (эмблем), визиток, плакатов;
- небольших изображений для сайтов в Интернете (иконок, кнопок).

Самые известные профессиональные векторные редакторы — *Adobe Illustrator* и *CorelDraw*. Бесплатно можно использовать редактор *Draw* из пакета *OpenOffice* и кроссплатформенную программу *Inkscape* (рис. 6.40).

Рис. 6.40

Иногда «плоских» рисунков недостаточно, и нужно представить объект в пространстве. Такие задачи возникают при проектировании машин и самолётов, в архитектуре, кино, телевидении и компьютерных играх. Для работы с трёхмерными объектами используют **программы 3D-моделирования** (трёхмерного моделирования), которые позволяют:

- определить форму (геометрию) объектов;
- задать материалы для объектов;
- установить источники света;
- определить точки наблюдения (виртуальные камеры);
- создать анимацию с трёхмерными объектами;
- выполнить **рендеринг**, т. е. построить «плоскую» картинку или анимацию, выбрав какую-нибудь точку наблюдения.

Среди программ 3D-моделирования наиболее популярны коммерческая программа *3ds Max* и бесплатная кроссплатформенная программы *Blender* (рис. 6.41). Модели, построенные в программах 3D-моделирования, можно использовать для печати на 3D-принтерах.

Рис. 6.41

Алгоритмы работы с трёхмерной графикой достаточно сложны и требуют большого объёма вычислений. Поэтому для нормальной работы программ 3D-моделирования, особенно для выполнения *рендеринга*, при котором просчитывается ход многих тысяч лучей от источников света и их влияние на изображение, требуется быстродействующий процессор и большой объём оперативной памяти.

В любом современном издательстве для подготовки к печати книг и журналов используется специальное программное обеспечение — **настольно-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 правилами типографского дела. В настольно-издательских системах готовят **оригинал-макет** (изображение, точно совпадающее с будущим отпечатком) и отправляют его в типографию.

Долгое время для подготовки оригинал-макетов использовались настольно-издательские системы *QuarkXPress* (**www.quark.com**), *Corel Ventura* и *Adobe Page Maker*; сейчас конкуренцию им составляют *Adobe InDesign* (**www.adobe.com**) и бесплатная свободно распространяемая программа *Scribus*, окно которой показано на рис. 6.42.

Рис. 6.42

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

Системы автоматизированного проектирования

Современный инженер для подготовки чертежей уже почти никогда не использует традиционные инструменты: кульман, линейку, лекала, перья. Всё делается на компьютере в **системах автоматизированного проектирования (САПР)**. САПР (англ. **CAD**: *Computer-Aided Design*) применяются в машиностроении, строительстве, электронике.

Все САПР используют векторное кодирование двумерных (плоских) и трёхмерных моделей. Главная задача большинства простых (двумерных) САПР — качественная подготовка чертежей. В 3D-системах можно вращать модель и строить растровую картинку, показывающую, как будет выглядеть готовое изделие.

Наиболее совершенные (но и очень дорогие!) САПР позволяют выполнять математическое моделирование, рассчитывать дета-

ли на прочность и температурные нагрузки, определять допуски (максимально допустимые отклонения размеров от заданных), готовить данные для работы станков с числовым программным управлением. Это даёт возможность не строить дорогие натурные модели, а просчитывать многие варианты на компьютере и выбирать лучший из них.

Наиболее известные САПР:

- *AutoCAD* (**autodesk.com**) — самая популярная система автоматизированного проектирования и черчения;
- *ArchiCAD* (**graphisoft.ru**) — для проектирования зданий, ландшафтов и мебели;
- *OrCAD* (**cadence.com**) — для проектирования электронных схем;
- *КОМПАС 3D* — российская САПР, позволяющая оформлять чертежи в соответствии с российскими стандартами.

Рассмотрим подробнее возможности САПР *КОМПАС 3D*. Её главная задача — облегчить создание чертежей и другой стандартной документации.

В программе удобно создавать обычные («плоские», двумерные) чертежи деталей (виды с разных сторон, разрезы и т. д.) и сборочные чертежи.

Сначала строятся вспомогательные направляющие линии, все размеры задаются в виде чисел. Затем остаётся просто обвести контур каждого вида по нужным направляющим и построить выносные размерные линии, причём значения размеров будут проставлены автоматически с учётом выбранного масштаба. Таким образом, всё, что раньше инженеры рисовали вручную, теперь можно значительно быстрее и аккуратнее сделать с помощью компьютерной программы.

Однако есть и другой вариант, который даёт ещё бóльшие возможности. Сначала строится трёхмерная модель нужной детали, а затем программа по ней автоматически создаёт нужные виды на чертеже. Причём при любых изменениях модели чертёж также автоматически перестраивается (рис. 6.43).

Чтобы создать простейшую трёхмерную фигуру в *КОМПАС 3D*, необходимо сначала нарисовать плоское сечение фигуры — **эскиз**. Сама фигура строится по этому сечению как результат выдавливания (рис. 6.44) или вращения (рис. 6.45). Например, выдавливанием из прямоугольника получается параллелепипед, а из окружности — цилиндр; при выдавливании внутрь тела образуется отверстие. Вращением прямоугольного треугольника вокруг катета можно получить конус, а вращением прямоугольника вокруг его стороны — цилиндр.

Рис. 6.43

Рис. 6.44

Рис. 6.45

При построении трёхмерных моделей сложных объектов приходится многократно добавлять к детали или «вычитать» из неё некоторые тела. При добавлении мы получаем различные выступы, а при вычитании — отверстия или канавки.

Программа *КОМПАС 3D* работает с файлами нескольких форматов. Чертежи сохраняются в файлах с расширением *cdw*, трёхмерные модели деталей — в файлах с расширением *m3d*, сборки (несколько трёхмерных моделей, между которыми установлены связи) — в файлах с расширением *a3d*. Программа умеет открывать файлы, созданные в программе *AutoCAD* с расширениями *dxf* и *dwg*, возможен экспорт модели из *КОМПАС 3D* в некоторые другие форматы САПР.

В *КОМПАС 3D* можно не только готовить чертежи, но и выполнять инженерные расчёты. Например, можно задать материал детали и изучить её реальные физические свойства — объём, плотность, массу, центр тяжести и т. д. Дополнительные библиотеки позволяют рассчитывать прочность деталей, напряжения, которые возникают при действии на них различных сил (рис. 6.46), выполнять теплофизические расчёты.

Рис. 6.46

КОМПАС 3D — это коммерческая программа, но существует её «облегчённая» версия *КОМПАС 3D LT*, которая бесплатна для домашнего использования и учебных целей. Главное ограничение этой версии — в ней можно работать только с одной деталью.

Выводы

- Пакет прикладных программ (ППП) — это набор программ для решения некоторого класса задач.
- Офисные пакеты обычно включают текстовый процессор, табличный процессор, программу для подготовки компьютерных презентаций, систему управления базами данных.
- ППП для управления предприятием содержат программы для бухгалтерского учёта, управления персоналом, управления перевозками и др.
- Пакеты для решения научных задач используют численные методы и символьные вычисления (алгоритмы компьютерной алгебры).
- Программы для дизайна — это растровые и векторные графические редакторы, программы 3D-моделирования.
- Настольно-издательские системы служат для подготовки книг и других изданий к печати в типографии.
- Системы автоматизированного проектирования применяют для подготовки чертежей и математического моделирования различных деталей, приборов, зданий и др.

Нарисуйте в тетради интеллект-карту этого параграфа.

Вопросы и задания

1. Какие два типа пакетов используют для научных расчётов? Почему нельзя обойтись только одним?
2. Сравните задачи, которые решаются с помощью растровых и векторных графических редакторов.
3. Почему рендеринг трёхмерных сцен требует больших вычислительных ресурсов?
4. Сравните возможности настольно-издательских систем и текстовых процессоров.
5. Выделите три типа САПР, которые упоминаются в тексте параграфа.
6. Используя дополнительные источники, выясните, какие пакеты прикладных программ существуют в других областях.

Подготовьте сообщение

а) «Бесплатные математические пакеты»
б) «Бесплатные САПР»

Проекты

а) Сравнение офисных пакетов
б) Сравнение пакетов для научных исследований
в) Сравнение САПР

Интересные сайты

mathworks.com — пакет *MATLAB* для научных и инженерных расчётов
scilab.org — бесплатный пакет *Scilab* для научных и инженерных расчётов
gnu.org/software/octave — бесплатный математический пакет *GNU Octave*
wolfram.com/mathematica — система символьных вычислений *Wolfram Mathematica*
ptc.com/products/mathcad/ — система компьютерной алгебры *Mathcad*
maplesoft.com — система символьных вычислений *Maple*
maxima.sourceforge.net — бесплатная система компьютерной алгебры *Maxima*
gimp.org — свободный растровый графический редактор *Gimp*
inkscape.org — свободный векторный графический редактор *Inkscape*
blender.org — свободная программа для 3D-моделирования *Blender*
scribus.net — свободная программа для компьютерной вёрстки *Scribus*
kompas.ru — программа *КОМПАС 3D*
veselowa.ru — «Черчение для всех» — уроки по программе *КОМПАС 3D*
saprblog.ru — уроки по различным САПР
2d-3d.ru/samouchiteli/ — уроки по 3D-моделированию и САПР
mysapr.com — уроки по программе *КОМПАС 3D*

§ 40
Обработка мультимедийной информации

Ключевые слова:

- мультимедиа
- АЦП
- ЦАП
- аудиокодек
- битрейт
- MIDI
- сэмпл
- распознавание речи
- синтез речи
- аудиоредактор
- удаление шума
- видеокодек
- видеоредактор

Как вы знаете, компьютеры были изобретены в первую очередь для того, чтобы ускорить сложные вычисления — обработку числовых данных. Однако текст, рисунки, звуки и видео тоже можно представить как числа. В результате в наше время компьютеры обрабатывают самые различные виды информации, причём в основном именно нечисловые.

Мультимедиа (от лат. *multum* — много, *media* — средства) — это использование различных форм представления информации (текст, графика, анимация, звук, видео и т. д.) в одном документе.

Часто при использовании мультимедиа человек может влиять на показ материалов: перейти вперёд или вернуться назад, изменить настройки, выбрать один из предложенных вариантов и т. п. Такое взаимодействия человека и компьютера называют *интерактивностью* (взаимной активностью).

Слово «мультимедиа» также используется в словосочетаниях мультимедиа-компьютер, мультимедиа-носитель, устройства мультимедиа, технологии мультимедиа. К **устройствам мультимедиа** относят устройства, предназначенные для работы с графикой, звуками, видео:

- дисководы для работы с оптическими дисками (CD, DVD, Blu-ray);
- видеокарты, содержащие мощные процессоры и оперативную память;
- звуковые карты;
- звуковые колонки;
- микрофон;

- MIDI-клавиатуру для записи музыки в виде нот через специальный разъём звуковой карты;
- *тюнер* — устройство, с помощью которого можно принимать, просматривать и сохранять на компьютере телевизионные сигналы и радиосигналы;
- цифровые фотокамеры и видеокамеры.

Чтобы работать с устройствами, надо знать, как это делается. Тут на помощь приходят технологии.

Технология — способ изготовления некоторого продукта из исходных материалов.

Вот некоторые **технологии мультимедиа**:

- *приём и обработка телевизионного сигнала*;
- *видеозахват* — ввод, сохранение в цифровом виде и обработка видеосигнала;
- *анимация* — «оживление» изображения на экране;
- *звуковые эффекты* (созданные на компьютере или записанные с помощью микрофона);
- *трёхмерная графика (3D-графика)*;
- *виртуальная реальность*[1], позволяющая пользователю погрузиться в мир, созданный с помощью компьютера, и действовать в нём.

К **программным средствам мультимедиа** относятся:

- *мультимедийные приложения* — энциклопедии, интерактивные обучающие курсы, компьютерные игры, тренажёры, рекламные ролики, компьютерные презентации и др.;
- *средства создания мультимедийных приложений* — редакторы изображений, звука и видеофильмов, программы для создания презентаций.

Обработка звуковой информации

В первых персональных компьютерах единственным звуковым устройством был небольшой встроенный динамик. В современных моделях звук обрабатывает звуковая карта. Материнские платы содержат встроенные (интегрированные) звуковые карты, однако для качественного проигрывания и записи звука их возможностей

[1] Слово «виртуальный» означает «действующий и проявляющий себя, как настоящий».

не хватает. В этом случае нужно использовать дополнительную звуковую карту, которая устанавливается в разъём материнской платы или подключается через USB-порт.

Как вы знаете из главы 2, устройства для ввода и вывода звука (микрофоны, наушники, звуковые колонки) — это аналоговые приборы. В то же время компьютер — это цифровое устройство, в котором данные хранятся и обрабатываются в двоичном коде. Поэтому нужны преобразователи, которые связывают цифровой компьютер с аналоговыми устройствами ввода и вывода. Такие преобразователи есть в звуковой карте — это **аналого-цифровой преобразователь (АЦП)**, который переводит аналоговый сигнал микрофона в цифровой код, и **цифро-аналоговый преобразователь (ЦАП)**, который преобразует цифровой код звука в аналоговый сигнал для наушников или звуковых колонок.

Устройство, которое кодирует и декодирует звуковые данные с помощью АЦП и ЦАП, называется **аппаратным аудиокодеком** (кодировщик/декодировщик). В современных звуковых картах используют кодеки, соответствующие стандартам AC97 или HD Audio (англ. *High Definition Audio* — звук высокой чёткости).

Оцифрованные звуковые данные занимают много места в памяти, поэтому для уменьшения размера файлов их сжимают с помощью специальных программ — **программных кодеков**. Для проигрывания звука программа-кодек распаковывает данные и передаёт их звуковой карте. Самый известный аудиокодек — *MP3* (*MPEG-1 Layer III*, файлы с расширением *mp3*). Этот кодек выполняет сжатие с потерями, учитывая особенности восприятия звука человеком. При сжатии можно регулировать **битрейт** — количество бит, используемых для хранения одной секунды звука. Например, при использовании битрейта 128 Кбит/с удаётся сжать звук примерно в 11 раз (по сравнению с оцифровкой без сжатия). Кроме кодека *MP3* часто применяются также кодек *AAC* (англ. *Advanced Audio Codec* — передовой аудиокодек) и свободный кодек *Ogg Vorbis* (*OGG*). Они также сжимают звук с потерями.

Существует и ещё один способ кодирования звука — стандарт **MIDI** (англ. *Musical Instrument Digital Interface* — цифровой интерфейс музыкальных инструментов). В MIDI-файле (с расширением *mid*) хранится не оцифрованный звук, а программа для проигрывания электронной музыки: ноты, код музыкального инструмента, громкость, тембр, темп, тональность. Когда звуковая карта получает эти данные, она ищет нужный образец звука (**сэмпл**, от англ. *sample* — образец) в волновой таблице, которая хранится в её памяти, и проигрывает его с помощью встроенного синтезатора (**секвенсора**). Поэтому на разных звуковых картах один и тот же MIDI-файл может звучать по-разному.

Принцип записи музыки в MIDI-файле напоминает принцип хранения векторной графики: запоминается не конечный результат, а данные, которые позволяют его получить.

Размер MIDI-файлов значительно меньше, чем размер файлов, получающихся при оцифровке того же звука. Однако с помощью этого метода кодируются только такие звуки, которые можно сыграть на музыкальных инструментах, включённых в стандарт MIDI. Речь человека, шум листвы, плеск волн таким образом закодировать не удастся.

Для прослушивания музыки используются **программы-плееры** (от англ. *play* — играть). Как правило, плеер входит в состав любой операционной системы. Среди современных программ этого класса можно отметить *Media Player Classic*, *Quick TIme Player*, *VLC Player*.

Компьютеры (точнее, компьютерные программы) научили весьма успешно **распознавать речь** человека на разных языках. Например, *Голосовой блокнот* на сайте **speechpad.ru** позволяет наговаривать текст с помощью микрофона. У компьютеров компании *Apple* есть функция голосового ввода «Диктовка». В состав операционной системы *iOS* для мобильных устройств включён персональный помощник *Siri* (англ. *Speech Interpretation and Recognition Interface* — интерфейс распознавания и интерпретации речи). Эта программа обрабатывает речь с микрофона, распознаёт смысл сказанного, выполняет звуковые команды и даёт рекомендации. Голосовой поиск есть также в приложении *Google* на мобильных устройствах с операционными системами *Android* и *iOS*. Такой способ общения с компьютером очень помогает людям с ограниченными возможностями.

Существуют системы **синтеза речи**, которые позволяют автоматически озвучивать текст. Основные области их применения — информационные сообщения в аэропортах и на вокзалах, автоответчики, роботы-консультанты и т. п. Во всех этих случаях системы синтеза речи успешно заменяют людей-дикторов.

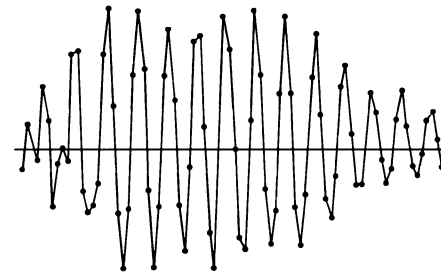
При оцифровке звука неизбежны некоторые потери информации (см. главу 2), но взамен мы получаем значительные преимущества: цифровой звук копируется без потери качества и обрабатывается с помощью специальных программ — **аудиоредакторов**. С их помощью можно:

- загружать, редактировать и сохранять звуковые файлы разных форматов;
- записывать звук с микрофона;
- вырезать фрагменты из файла;
- соединять звуковые фрагменты в один файл;
- изменять громкость и темп звука;
- удалять шумы.

Самые известные профессиональные аудиоредакторы — *Adobe Audition* и *Sound Forge*. Для простой обработки звука можно использовать бесплатную кроссплатформенную программу *Audacity* (рис. 6.47).

Рис. 6.47

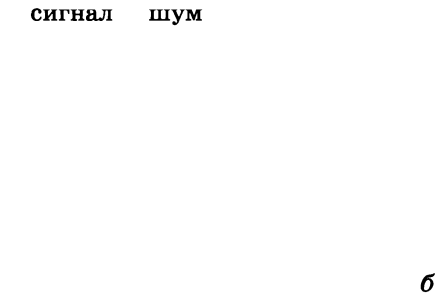
После загрузки звукового файла в аудиоредактор вы увидите график, представляющий собой оцифрованный звук (рис. 6.48, *а*). Если сильно увеличить масштаб, можно увидеть отдельные точки — хранящиеся в памяти значения (рис. 6.48, *б*). Сигнал между этими точками был потерян при оцифровке, поэтому программа соединяет их отрезками — это самый простой способ восстановления сигнала.



а *б*

Рис. 6.48

Звук, записанный с микрофона, почти всегда содержит **шум** — посторонние звуки, которые желательно удалить. Для этого нужно выделить в записи участки, где есть только шум, но нет полезного сигнала (рис. 6.49, *а*).



а *б*

Рис. 6.49

Предполагают, что шум действует постоянно и его характеристики не меняются. Выделив участок без полезного сигнала, нужно построить модель шума (аудиоредакторы умеют это делать!), а затем программа использует эту модель для того, чтобы «вырезать» шум из остальной части звуковых данных (рис. 6.49, *б*). Алгоритм такой обработки достаточно сложен. Дело в том, что звуковой сигнал — это смесь звуковых волн разных частот. Обычно полезный сигнал и шум имеют разные частотные диапазоны: шум содержит более высокие частоты. Поэтому программа просто удаляет из сигнала высокочастотную часть. Конечно, при этом немного искажается и полезный сигнал — ведь он тоже может содержать высокие частоты.

Аудиоредакторы позволяют работать с несколькими звуковыми дорожками, которые проигрываются одновременно. На рисунке 6.50 показано, как можно объединять две дорожки: сначала работает только вторая дорожка (музыкальное сопровождение), затем её громкость плавно уменьшается и включается первая дорожка (диктор читает текст). На последнем этапе, когда чтение текста завершено, музыкальное сопровождение на второй дорожке вновь становится громче.

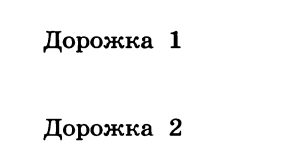


Рис. 6.50

Для изменения громкости звука в аудиоредакторах используют **огибающие** — векторные кривые, ограничивающие громкость; форму огибающей можно изменять как угодно, перетаскивая её узлы мышью.

Обработка видеоинформации

Цифровое видео в памяти компьютера — это изменяющееся изображение и оцифрованный звук, которые проигрываются синхронно (одновременно). Изображение меняется не реже 25 раз в секунду, звук обычно оцифрован с частотой 48 кГц.

При записи цифрового видео обычно используются форматы стандартной чёткости:

- 768 × 576 пикселей (соотношение сторон кадра 4:3)

или высокой чёткости:

- 1280 × 720 пикселей (*HD*, от англ. *High Definition*);
- 1920 × 1080 пикселей (*Full HD*).

Поскольку несжатые видеоданные имеют огромный объём (за 1 минуту записи в формате *Full HD* нужно сохранить более 7 Гбайт данных!), все видеоформаты используют сжатие с потерями. Эту операцию выполняют **видеокодеки** — программы для сжатия и распаковки видеоданных. Чаще всего используют кодеки *H.264*, *DivX*, *XviD* и *Theora* (последние два — свободное программное обеспечение). При очень сильном сжатии с потерями появляются заметные искажения (*артефакты*), например становится явно видно, что изображение разбито на блоки размером 8 × 8 пикселей.

Для обработки цифрового видео используют **видеоредакторы**. С их помощью можно:

- вводить данные с видеокамеры;
- корректировать цвет кадров;
- добавлять, переставлять и удалять фрагменты фильма;
- добавлять и редактировать звуки и титры;
- сохранять фильм в различных цифровых видеоформатах;
- создавать DVD-диски.

Среди коммерческих видеоредакторов популярны *Adobe Premier*, *Pinnacle Studio*, *VideoStudio Pro*, *Sony Vegas Pro*. На компьютерах фирмы *Apple* используют программу *iMovie*, а для профессиональной работы — программу *Final Cut Pro X*.

Существуют и бесплатные видеоредакторы, например программа *Kdenlive* для операционной системы *Linux*, программы *VirtualDub* и *VSCD Free Video Editor* для *Windows*, кроссплатформенная программа *Avidemux*.

Современные программы позволяют редактировать видео даже на планшетных компьютерах и смартфонах. В операционной системе *iOS* (на смартфонах *iPhone* и планшетах *iPad*) можно использовать программу *iMovie*. Программа *Adobe Premier Clip* работает как в *iOS*, так и на устройствах с операционной системой *Andriod*. Есть и бесплатные видеоредакторы для *Android*, такие как *KineMaster, WeVideo, VideoPad*.

Наиболее совершенные видеоредакторы позволяют накладывать друг на друга несколько видео- и аудиодорожек (рис. 6.51), применять спецэффекты, выполнять плавные переходы между фрагментами, обрезку видео и редактирование звука, не выходя из редактора.

Рис. 6.51

Выводы

- Мультимедиа — это использование различных форм представления информации в одном документе. Примеры применения технологии мультимедиа — видеофильмы, компьютерные презентации, игры.
- Существует два способа кодирования звука: оцифровка и инструментальное кодирование (стандарт MIDI). С помощью оцифровки можно закодировать любой звук, а с помощью стандарта MIDI — только нотную запись.
- Устройства и программы для кодирования/декодирования звука и видео называются кодеками.
- При кодировании оцифрованного звука и видео применяют сжатие с потерями.

- Современные видеоредакторы позволяют работать с несколькими дорожками, содержащими звуковые данные и видеоданные.

Нарисуйте в тетради интеллект-карту этого параграфа.

Вопросы и задания

1. Какую роль выполняют АЦП, ЦАП и синтезатор в составе звуковой карты?
2. Почему MIDI-звук может по-разному звучать на разных устройствах?
3. Используя дополнительные источники, выясните различия между стандартам *AC97* и *HD Audio*.
4. Почему не всякий звук можно закодировать с помощью стандарта *MIDI*?
5. Почему задачу распознавания речи относят к области искусственного интеллекта?
6. В каких задачах используется синтез речи?
7. Объясните принцип удаления шумов из звукового фрагмента.
8. Какие преимущества даёт использование нескольких дорожек при обработке звука и видео?

Подготовьте сообщение

а) «Аудиоредакторы для мобильных устройств»
б) «Видеоредактор *VirtualDub*»
в) «Видеоредактор *Avidemux*»
г) «Видеоредакторы для *iOS*»
д) «Видеоредакторы для *Android*»

Проекты

а) Монтаж звуковой дорожки
б) Монтаж видеоролика на выбранную тему

Интересные сайты

sourceforge.net/projects/audacity/ — аудиоредактор *Audacity*
audacity.ru/p1aa1.html — учебник по программе *Audacity*
virtualdub.org — видеоредактор *VirtualDub*
videosoftdev.com/ru/free-video-editor — видеоредактор *VSCD Free Video Editor*
avidemux.sourceforge.net — видеоредактор *Avidemux*

§ 41
Программы для создания презентаций

Ключевые слова:

- презентация
- слайд
- дизайн
- тема
- палитра
- цветовой круг
- макет
- шрифт
- фон
- контраст
- переходы
- сортировщик слайдов

Что такое презентация?

Компьютерная презентация — это набор изображений (слайдов), которые сменяют друг друга по команде человека или через заданные промежутки времени.

Компьютерная презентация служит для иллюстрации устного выступления.

Для подготовки презентаций часто используется программа *PowerPoint* из пакета *Microsoft Office*, бесплатная программа *OpenOffice Impress* или сервис *Google Docs*.

Слайд в компьютерной презентации может содержать самую разную информацию: текст, рисунки, таблицы, диаграммы, анимацию, видеофильм. В презентацию можно добавлять звук, который проигрывается во время показа одного или нескольких слайдов.

К каждому слайду можно добавить заметки, где записывают примерный текст выступления или комментарии. Во время показа презентации заметки не выводятся на экран.

Слайды можно распечатать (в том числе и в уменьшенном размере, по несколько слайдов на странице), тогда у вас будет раздаточный материал для слушателей. Им будет удобнее следить за выступлением и задавать вопросы.

В последние годы активно используются онлайн-инструменты для подготовки презентаций в браузере, самый известный из таких сервисов — Prezi (**prezi.com**).

Содержание презентаций

Цель любого докладчика — сообщить какую-то важную информацию и убедить слушателей в том, что это действительно важно.

Для этого нужно подать материал так, чтобы слушателям было интересно, чтобы они всё поняли и сделали нужные выводы.

Презентация не заменит электронный учебник, справочник, энциклопедию, веб-сайт, потому что это другой жанр. Не нужно выводить на слайды весь текст выступления. Если вы хотите, чтобы те, кто открыл презентацию, смогли прочитать текст, добавьте его в заметки к каждому слайду. Люди читают текст на слайдах быстрее, чем вы говорите, и если информация будет дублироваться, они перестанут вас слушать. Добавлять в презентацию много текста (как на рис. 6.52) бессмысленно, его всё равно никто не будет читать.

Лисица

Лиса, или лисица, — общее название нескольких видов млекопитающих семейства псовых. Лишь 11 видов этой группы относят к роду собственно лисиц (лат. *Vulpes*). Наиболее известный и распространённый представитель — обыкновенная лисица (*Vulpes vulpes*). Лисицы встречаются в фольклоре многих народов по всему миру. Согласно современным представлениям о филогении псовых, группа лисиц — полифилетическая, следовательно, непригодная в качестве таксона.

Рис. 6.52

Не пытайтесь включить в презентацию много информации. Наоборот, всё лишнее, что не «работает» на вашу цель, нужно убрать. Если вы думаете, что какая-то информация может понадобиться во время ответов на вопросы, сделайте дополнительные скрытые слайды. Они не будут показаны во время вашего доклада, но вы всегда сможете перейти к ним, если это будет необходимо.

Дизайн презентации

Для презентации важно не только содержание, но и оформление. На первых порах можно воспользоваться готовыми **стилями (темами)** оформления, которые разработаны профессиональными дизайнерами. В *PowerPoint* меню для выбора темы находится на вкладке *Дизайн* (рис. 6.53).

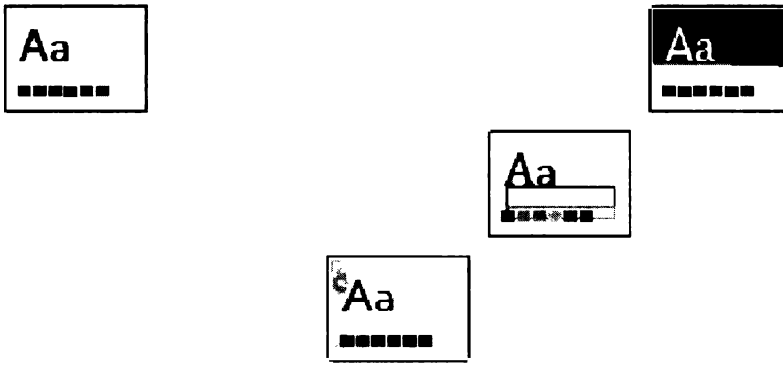


Рис. 6.53

Для каждой темы вы можете выбрать цветовую палитру, набор шрифтов, фон.

Опытные авторы презентаций редко используют готовые темы, а обычно создают свои. Одна из важных задач, которую при этом приходится решать, — выбор цветов для фона и текста. Гармоничные сочетания цветов изучает теория цвета. Для выбора палитры применяют цветовой круг (см. рис. 6.54 и цветной рисунок на форзаце).

Жёлтый

Зелёный Красный

Голубой Пурпурный

Синий

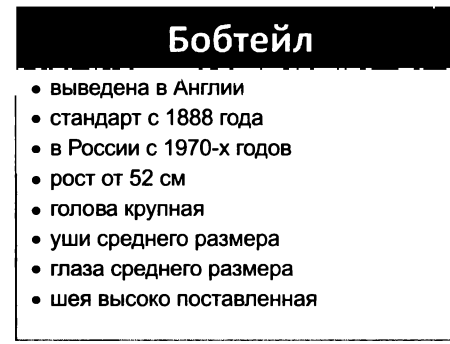
Рис. 6.54

Если нужно выбрать два цвета, обычно берут цвета, расположенные друг напротив друга, например красный и голубой, жёлтый и синий. Три цвета выбирают в вершинах равностороннего треугольника, например красный, синий и зелёный. Можно выбирать цвета в вершинах квадрата и других равносторонних фигур (пятиугольника, шестиугольника и т. п.).

Не забывайте, что есть ещё чёрный и белый цвета, которые очень хорошо сочетаются со всеми остальными. Дизайнер Роджер Блэк даже сформулировал такую мысль: «*Первый цвет — белый, второй — чёрный, третий — красный*». Действительно, чёрный и белый — это самый тёмный и самый яркий цвета, с ними отлично сочетается красный, его можно использовать для выделения. Хороший контраст также получается при использовании чёрного цвета рядом с жёлтым. Так, например, обозначается опасная зона — см. цветной рисунок на форзаце).

Слайд не должен выглядеть разноцветным, как попугай (рис. 6.55, *а* и цветной рисунок на форзаце). Желательно использовать не более трёх-четырёх цветов (для фона, заголовков, обычного и выделенного текста).

Неправильно: Правильно:



а *б*

Рис. 6.55

Макеты

При создании слайда можно использовать готовые макеты или делать всё вручную. Макет — это один из готовых вариантов размещения информации на слайде.

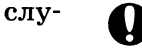
В программах для создания презентаций есть готовые макеты, разработанные опытными специалистами. Вот такие макеты предлагает программа *Impress* (рис. 6.56).

Рис. 6.56

Первый слева макет в верхнем ряду — пустой, если выбрать его, все элементы придётся добавлять вручную. На остальных макетах явно выделен заголовок и одна или несколько областей для элементов слайда. В каждую область можно вставить текст, рисунок, диаграмму, видео. Размеры областей можно менять, перетаскивая маркеры на углах рамок.

Размещение элементов слайда

Главная задача презентации — донести информацию до слушателей.



Вы должны сделать всё, чтобы облегчить вашим слушателям получение информации. Лишние элементы слайда — это информационный шум, он не улучшает восприятие, а только усложняет его.

Человек не может воспринимать много информации одновременно, поэтому обычно на слайд добавляют не более 7 элементов. Мелкие детали заставляют людей напрягать зрение, поэтому лучше их не использовать. Хорошо, если на слайде будут только три ведущих объекта. Если на слайде оказалось больше 7—9 объектов, лучше подумайте о том, чтобы сделать один дополнительный слайд.

По краям слайда нужно оставлять поля, самую важную информацию размещать ближе к центру (рис. 6.57 и цветной рисунок на форзаце).

Неправильно:

Правильно:

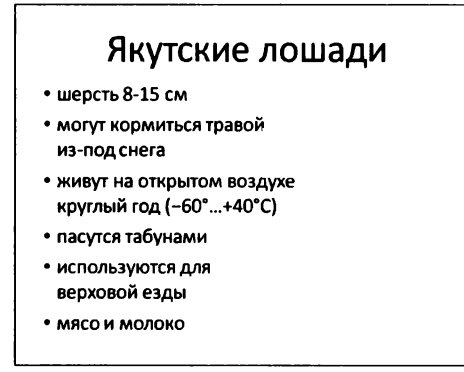


Рис. 6.57

Элементы на слайде не должны быть разбросаны в произвольном порядке, их нужно зрительно связывать друг с другом — выравнивать по вертикали и горизонтали. Выровненные элементы образуют в сознании человека единое целое: невидимая линия «связывает» их, даже если они находятся на некотором расстоянии друг от друга (рис. 6.58 и цветной рисунок на форзаце). Для выравнивания вы можете использовать встроенные возможности программ для подготовки презентаций:

- горизонтальное выравнивание: по левой или правой границе объектов, по центру;
- вертикальное выравнивание: по верхней или нижней границе объектов, по середине;
- распределение по горизонтали и вертикали (на равных расстояниях друг от друга).

Неправильно:

Правильно:

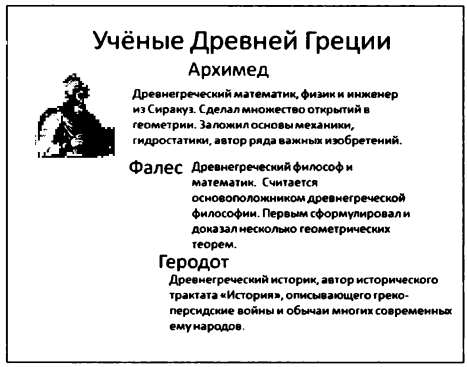


Рис. 6.58

Оформление текста

Для того чтобы текст был виден с задних рядов, его размер делают не менее 24 пунктов (напомним, что 1 пункт = 1/72 дюйма, а 1 дюйм = 2,54 см). Заголовки слайдов делают крупнее, чем подзаголовки и основной текст слайда, ведь их должны заметить в первую очередь.

Размер шрифта для каждого типа элементов должен быть одинаковым на всех слайдах, например 32 пункта для заголовков и 24 пункта для текста.

Для презентаций обычно выбирают шрифты без засечек (рубленые), например Arial, Calibri, Helvetica. Дело в том, что текст, набранный шрифтом с засечками (чёрточками на верхних и нижних концах букв), на расстоянии сливается в одну массу, и отдельные буквы трудно разобрать. В презентации обычно используют не более двух названий шрифтов (**гарнитур**).

В книгах текст выравнивается по ширине (т. е. выравниваются левая и правая границы). Этот приём хорошо работает, если строки достаточно длинные. В презентациях мы используем более крупный шрифт, и при выравнивании по ширине получаются очень большие «дырки» между словами, поэтому нужно использовать выравнивание по левой границе (рис. 6.59).

Неправильно:

Задачи

Задача 1
В бублике 1 дырка, а в кренделе в два раза больше. На сколько меньше дырок в 7 бубликах, чем в 12 кренделях?

Задача 2
Площадь одного уха слона равна 10 000 кв. см. Узнай в кв. м площадь ушей 12 одинаковых слонов.

Правильно:

Задачи

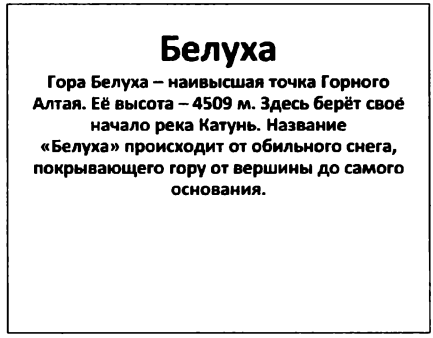
Задача 1
В бублике 1 дырка, а в кренделе в два раза больше. На сколько меньше дырок в 7 бубликах, чем в 12 кренделях?

Задача 2
Площадь одного уха слона равна 10 000 кв. см. Узнай в кв. м площадь ушей 12 одинаковых слонов.

Рис. 6.59

Выравнивание по центру используют только для заголовков. Недопустимо выравнивать по центру длинные тексты и тем более списки (рис. 6.60 и 6.61). Дело в том, что ровная левая граница и маркеры служат для того, чтобы легко найти начало следующей строки. При выравнивании по центру в каждой строке текст начинается в разных местах, и найти начало строки намного сложнее.

Неправильно:



Правильно:

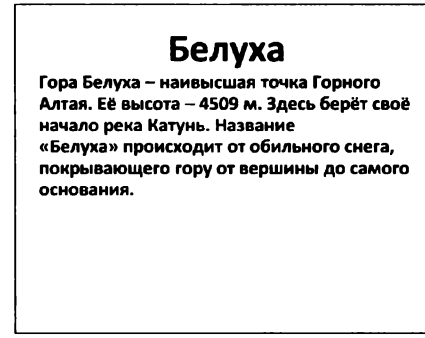
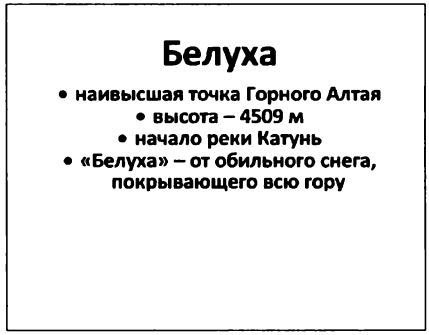


Рис. 6.60

Неправильно:



Правильно:

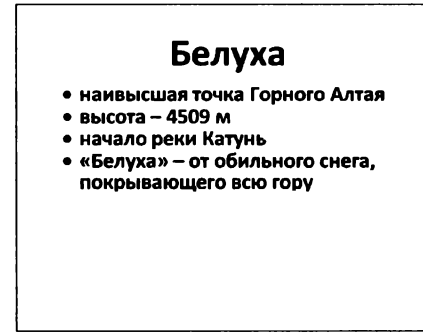


Рис. 6.61

Цвета фона и текста нужно выбирать так, чтобы они были **контрастными**, т. е. резко отличались друг от друга. Для того чтобы проверить контрастность цветов, можно перевести слайд в чёрно-белый вариант, тогда мы сравним тон пикселей текста и фона. Часто проектор искажает цвета и снижает контраст, поэтому получается, что на экране монитора текст виден хорошо, а на большом экране — плохо. Чтобы этого не произошло, контраст нужно выбирать «с запасом» (рис. 6.62).

Неправильно: Правильно:

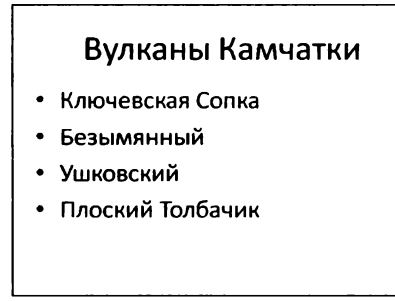


Рис. 6.62

Авторы многих презентаций любят делать фоном какую-нибудь картинку, и часто она мешает читать текст (рис. 6.63, *а*). Поэтому профессиональные дизайнеры, как правило, выбирают одноцветный фон (рис. 6.63, *б*). В крайнем случае, если для чего-то очень нужно оставить фоновый рисунок, можно подложить под текст так называемые «плашки» — одноцветные прямоугольники (рис. 6.63, *в*).

Неправильно: Правильно

а *б*

Правильно

в

Рис. 6.63

Как проверить, правильно ли оформлен ваш слайд? Задайте себе несколько вопросов и убедитесь, что на все эти вопросы ответ — «да».

— На слайде не более 7–9 объектов?
— На слайде есть поля?
— Элементы на слайде выровнены по вертикали и горизонтали?
— Текст хорошо читается издалека? Даже при показе через проектор?
— Рисунки и фон не мешают воспринимать информацию?

Добавление объектов

Часто элементов, которые уже есть в выбранном макете слайда, не хватает. Тогда приходится добавлять новые. В *PowerPoint* для этого используется вкладка *Вставка* (рис. 6.64).

Рис. 6.64

С её помощью можно добавить на слайд таблицы, рисунки, клипы из коллекции, надписи, векторные фигуры, диаграммы, звуки, видео. В программе *Impress* для этой цели служит меню *Вставка*.

С **таблицами** работают так же, как и в текстовом процессоре. Таблицы, как и все объекты на слайде, можно перемещать за рамку в любое нужное место. Их размеры изменяются с помощью маркеров на рамке.

При **вставке рисунка** программа предложит выбрать файл на диске. Кроме того, можно вставить рисунок (а также текст и таблицу) через буфер обмена из другой программы.

При **добавлении диаграммы** в программе *PowerPoint* появляется окно программы *Excel*, где нужно ввести данные, по которым строится диаграмма. В программе *Impress* для изменения данных нужно щёлкнуть правой кнопкой мыши на диаграмме и выбрать пункт *Таблица данных диаграммы* в контекстном меню.

При **вставке звука** вы выбираете файл на диске и настраиваете свойства звука:

- способ запуска: *автоматически* (при показе слайда) или *по щелчку*;
- момент окончания: *по щелчку*, *после этого слайда* или *после другого слайда* (последний вариант позволяет сделать в *PowerPoint* музыкальное сопровождение на несколько слайдов).

В современных версиях *PowerPoint* можно сделать обрезку звукового фрагмента.

Возможности программы *Impress* значительно скромнее, там звук всегда запускается автоматически и заканчивается при переходе к новому слайду.

При **добавлении видео** можно настроить размеры области показа и переместить её в нужное место. В *PowerPoint* можно установить режим показа *по щелчку* и развернуть видео на весь экран, в программе *Impress* видео всегда запускается автоматически.

Если вы используете звук и видео, нужно помнить про две особенности. Во-первых, для того чтобы прослушать звук и просмотреть видеофайлы, нужно установить используемые в них **программы-кодеки** (кодировщики/декодировщики). Может случиться так, что на вашем компьютере нужные кодеки есть, а на компьютере, где вы будете показывать презентацию, — нет, и поэтому слушатели ничего не услышат и не увидят. С этой точки зрения лучше всего использовать форматы *WAV* и *MP3* для звука и форматы *WMV* и *MPEG* для видео. Эти кодеки установлены на большинстве компьютеров.

Во-вторых, звуковые и видеофайлы не всегда сохраняются внутри файла с презентацией[1)]. Вместо этого программа устанавливает ссылки на их текущее место на диске и загружает их в память тогда, когда это необходимо. Поэтому при переносе презентации на другой компьютер необходимо скопировать не только файл с презентацией, но и все мультимедийные файлы. Лучше всего, если все эти файлы будут находиться в одной папке.

Переходы между слайдами

Вы можете определить эффекты, которые происходят при замене одного слайда другим. Они называются **переходами**. В программе *PowerPoint* для этого используется вкладка *Переходы* (рис. 6.65), а в *Impress* — панель *Смена слайдов*.

Рис. 6.65

---

1) В современных версиях *PowerPoint* звуковые и видеофайлы внедряются в презентацию.

Как видно из рис. 6.65, для выделенного слайда можно выбрать один из многочисленных эффектов перехода, сопровождающий звук, скорость перехода. Кнопка *Применить ко всем* позволяет установить этот режим перехода для всех слайдов.

Анимация в презентациях

Существует всего несколько ситуаций, когда анимация в презентациях оправданна.

Последовательное появление элементов. Если слайд содержит много элементов, нет смысла показывать все элементы сразу. Лучше, если они будут появляться последовательно, как будто вы пишете и рисуете на доске во время выступления. Появляется новый объект, и вы начинаете про него рассказывать.

Представьте себе, что вы просите слушателей ответить на какие-то вопросы, и на экран выводятся по одному варианты ответов. После того как вы вместе выяснили, что ответ неправильный, его можно перечеркнуть красной линией или крестом (рис. 6.66).

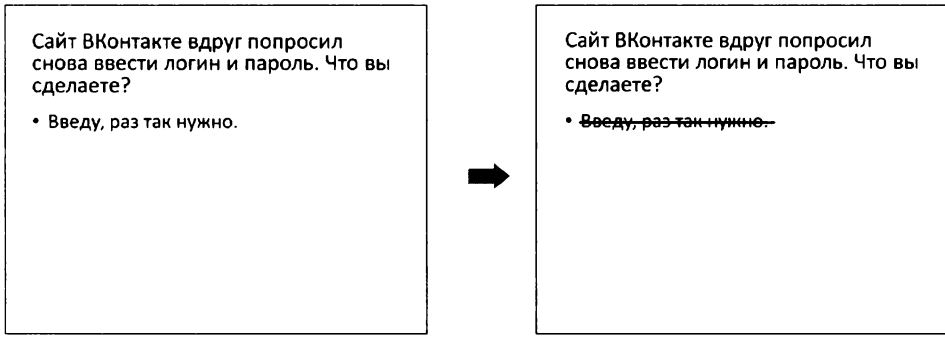


Рис. 6.66

Иногда нужно временно вывести на экран какой-то вопрос или пример, а потом его убрать (рис. 6.67).

Установка элемента на своё место. Допустим, вы хотите крупно показать несколько рисунков на одну тему и затем оставить на экране их уменьшенные копии. В этом случае можно сначала выводить крупный рисунок, а затем перемещать его на постоянное место, одновременно уменьшая размеры (рис. 6.68).

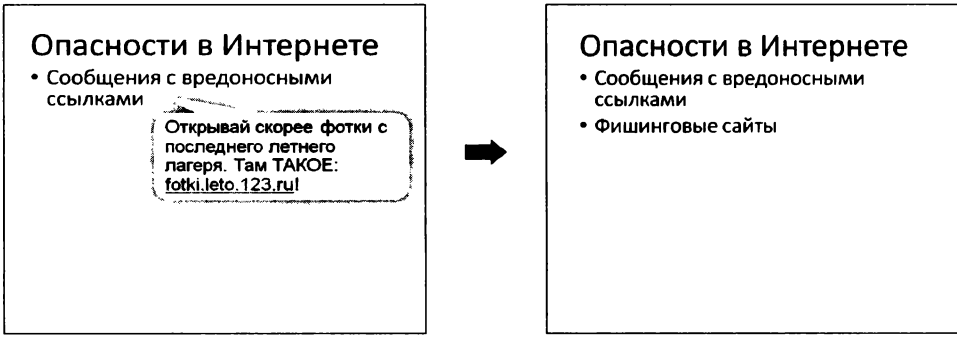


Рис. 6.67

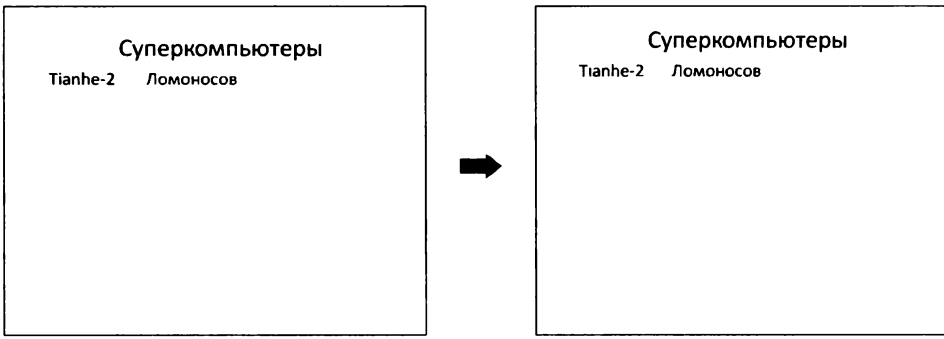


Рис. 6.68

Иллюстрация процесса. Если вам нужно наглядно показать какие-то изменения, анимация поможет решить эту задачу. Например, вы рассказываете о том, как бильярдный шар отскакивает от стенки поля. Можно, конечно, просто нарисовать его путь в виде линий, но анимация позволит слушателям легче понять, что происходит.

В программе *PowerPoint* анимация выполняется с помощью панели *Настройка анимации* (вкладка ленты *Анимация*), а в программе *Impress* — с помощью панели *Эффекты*.

К каждому объекту можно применить несколько типов анимации (и по несколько раз!):

- *вход* (появление объекта на экране);
- *выход* (исчезновение объекта);
- *выделение* (изменение свойств объекта);
- *перемещение* (по прямой или по нарисованной траектории).

На рисунке 6.69 показана панель настройки анимации в *PowerPoint*.

Рис. 6.69

Для каждого типа анимации предусмотрено множество эффектов. Для входа объектов часто используют эффекты *Растворение* (на месте) или *Появление* (постепенное прорисовывание в одном направлении, например слева направо).

Для текста можно установить анимацию по абзацам, например, чтобы выводить элементы списка по одному. Любой объект может участвовать в анимации несколько раз.

Выводы

- Компьютерная презентация — это набор изображений (слайдов), которые сменяют друг друга по команде человека или автоматически через заданные промежутки времени.
- Компьютерная презентация служит для иллюстрации устного выступления. Главная задача презентации — донести информацию до слушателей.
- В презентации обычно используют не более 3–4 цветов. Один из лучших трёхцветных наборов — чёрный, белый и красный.
- Лучше всего размещать на слайде от 3 до 7 элементов.
- Со всех сторон слайда нужно оставлять поля.
- Элементы слайда должны быть выровнены по вертикали и/или по горизонтали.
- Для презентаций обычно выбирают шрифты без засечек (рубленые), потому что они лучше читаются издалека.
- Текст в узких колонках выравнивают по левой границе (не по ширине).

- Выравнивание по центру используют только для заголовков. Недопустимо выравнивать по центру длинные тексты и списки.
- Цвета фона и текста нужно выбирать так, чтобы они были контрастными. Фоновый рисунок не должен мешать чтению текста.
- Все слайды презентации должны быть оформлены в одном стиле.
- Переходы — это эффекты при замене одного слайда другим.

Нарисуйте в тетради интеллект-карту этого параграфа.

Вопросы и задания

1. Как вы думаете, можно ли использовать компьютерные презентации как самостоятельные документы? Почему? Обсудите этот вопрос в классе.
2. Как вы считаете, нужно ли писать план (сценарий) презентации? Ответ обоснуйте.
3. Почему не рекомендуют размещать на слайдах много текста? Согласны ли вы с этим мнением?
4. Как можно использовать заметки к слайдам?
5. Предложите различные варианты использования скрытых слайдов.
6. Проверьте, как готовые темы оформления в вашей программе подготовки презентаций будут смотреться через проектор. Нравится ли вам результат?
7. Найдите в Интернете бесплатные презентации на интересующую вас тему, обсудите в классе, как они оформлены.
8. Выберите тему, на которую вы хотели бы сделать презентацию. Составьте план презентации из 5–6 слайдов, записав названия слайдов.
9. Предложите какой-нибудь алгоритм выбора макета для слайда. Он будет линейным, разветвляющимся или циклическим?
10. Обсудите достоинства и недостатки пустого макета, в котором всё приходится делать вручную. Почему его достаточно часто используют?
11. Как вы понимаете выражение «информационный шум»? Почему от него нужно избавляться?
12. Какие проблемы могут возникнуть при использовании в презентациях звука и видео? Как их решать?

Подготовьте сообщение

а) «Эмоциональный и деловой стиль в презентациях»
б) «Презентации, которые делал Стив Джобс»
в) «Польза и вред презентаций»
г) «Типичные ошибки в оформлении презентаций»
д) «Как люди воспринимают информацию на слайде?»
е) «Сервисы для создания презентаций»
ж) «Использование *Google Презентаций*»
з) «Сервисы для размещения презентаций в Интернете»
и) «Когда анимация не нужна?»
к) «Триггеры в презентациях»

Проекты

а) Сравнение программ для создания презентаций
б) Презентация видеоролика на выбранную тему
в) Презентация школьной научно-технической конференции

Интересные сайты

office.live.com/start/PowerPoint.aspx — онлайн-версия программы *PowerPoint*
artlebedev.ru/kovodstvo/ — проект «Ковództво» А. Лебедева (заметки о дизайне)
blog.powerlexis.ru — блог, посвящённый дизайну презентаций
color.adobe.com/ru/create/color-wheel/ — подбор цветовой палитры
paletton.com — подбор цветовой палитры
slidecast.ru — онлайн-сервис для создания анимированных презентаций
ru.calameo.com — сервис для размещения презентаций
slideshare.net — сервис для размещения презентаций
slideboom.com — сервис для размещения презентаций
ispring.ru — инструменты для создания мультимедийных презентаций на основе *PowerPoint*
prezi.com — онлайн-сервис для создания анимированных презентаций на основе технологии *Adobe Flash*
xplainto.me — программа «Объясняшки» для создания рисованных анимационных видеороликов на *iPad*

§ 42
Системное программное обеспечение

Ключевые слова:

- операционная система
- многозадачность
- терминал
- начальный загрузчик
- командный процессор
- утилита
- ОС реального времени
- драйвер
- ядро ОС
- файловая система
- кластер
- сектор
- журналирование

Что такое операционная система?

Команды, которые умеет 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 пользователю и прикладным программам удобный *интерфейс* (способ обмена данными) с аппаратными средствами компьютера.

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 (времени работы процессора, памяти, внешних устройств) между несколькими одновременно работающими программами.

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 простейшие 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 полнее использовать мощность компьютера, разработали **пакетный режим**: в ОС добавили команды, с помощью которых можно вводить в компьютер сразу несколько задач и выполнять их в автоматическом режиме. Это позволило максимально загрузить все устройства компьютера, например, пока идёт печать результатов одной задачи, выполнять вычисления по следующей.

На компьютерах третьего поколения часто применялся **многопользовательский режим (режим разделения времени)**. С большим компьютером (**мэйнфреймом**) было связано несколько **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* (*Microsoft Disk Operating System* — дисковая операционная система фирмы *Microsoft*).

Все современные ОС — многозадачные. ОС распределяет время работы процессора между запущенными программами, выделяя каждой *кванты* (небольшие интервалы) времени, так что создаётся 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 — запустить процесс загрузки ОС, который после нескольких подготовительных этапов (например, загрузки программ обслуживания файловой системы) завершается чтением в память ядра (основной части) ОС;
- *система управления памятью*;

- *система ввода/вывода*, которая управляет внешними устройствами и файлами; она использует программы для обмена данными с дисководами, клавиатурой, монитором и принтером; эти программы хранятся в постоянном запоминающем устройстве (ПЗУ) микросхемы BIOS, расположенной на материнской плате (см. главу 5);
- *командный процессор* — программа, которая выполняет команды пользователя, введённые в командной строке, и *командные файлы* — текстовые файлы, содержащие списки команд и даже программы на специальном языке программирования;
- *утилиты* (лат. *utilitas* — польза) — служебные программы 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 запускается при включении питания. Например, микроконтроллеры, встроенные в бытовые устройства, могут обходиться без операционной системы. Однако такой компьютер сложно программировать 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*, *macOS* и *Linux*. Все они используют графический интерфейс: окна программ, управление с помощью мыши, кнопки, переключатели и т. п.

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

Система **macOS** компании *Apple* установлена примерно на 5% компьютеров. Её часто используют профессионалы в области дизайна, компьютерной графики, полиграфии, видеомонтажа.

Около 1% компьютеров работают под управлением ОС **Linux**. Её начал разрабатывать в 1991 году 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* — свободное программное обеспечение, она распространяется бесплатно вместе с исходными кодами, так что каждый (при желании и умении) может её усовершенствовать.

Поэтому большинство современных суперкомпьютеров (до 80%) работает под управлением собственных ОС, основанных на ядре *Linux*. Множество серверов в локальных сетях и в Интернете, встроенные компьютеры в банкоматах, терминалах оплаты и даже беспилотных военных аппаратах управляются с помощью *Linux*.

На основе ядра *Linux* построено много различных **дистрибутивов** (распространяемых сборок). В дистрибутивы входит не только сама операционная система, но и программное обеспечение, состав которого зависит от конкретной сборки.

Программное обеспечение для *Linux* распространяется в виде **пакетов** — готовых к установке файлов специального формата. Наибольшее количество пакетов входит в дистрибутив *Debian* (пакеты с расширением *deb*), который стал основой для целого ряда популярных дистрибутивов — *Linux Mint* (**linuxmint.com**), *Ubuntu* (**ubuntu.com**), *Edubuntu* (**www.edubuntu.org**) и др.

Дистрибутивы второй группы являются «наследниками» *Red Hat Linux*, которая использовала пакеты с расширением *rpm*. Самые известные из них — *Fedora* (**fedoraproject.org**), *OpenSUSE* (**opensuse.org**), *CentOS* (**centos.org**), *Scientific Linux* (**www.scientificlinux.org**). Дистрибутивы *ALT Linux* (**www.altlinux.org**) и *Rosa Linux* (**rosalab.ru**) выпускаются российскими компаниями.

Появление карманных персональных компьютеров (КПК), смартфонов и планшетов 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*), *Windows Phone*, *Symbian*, *BlackBerry*. Портативные компьютеры фирмы *Apple* (*iPhone*, *iPad*) работают под управлением операционной системы *iOS*. Во всех этих ОС основной способ установки программного обеспечения — загрузка приложений из онлайн-магазинов.

По сравнению с ОС для персональных компьютеров мобильные операционные системы обеспечивают работу с сенсорным экраном, датчиками, сотовой связью, беспроводными сетями (*Bluetooth*, *Wi-Fi*), средствами GPS-навигации, фото- и видеокамерами, диктофоном, а также распознавание речи. Для планшетов и смартфонов важны не только характеристики быстродействия, но и уровень комфорта для пользователя. Разработчики прилагают большие усилия для того, чтобы уменьшить потребление электроэнергии и продлить время работы мобильного устройства без подзарядки.

Существует ещё один класс операционных систем, от которых требуется не просто решать задачи, а делать это за определённый промежуток времени. Такие ОС называются **операци-**

онными 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.lynx.com**).

Иногда говорят о том, что смартфоны фактически содержат две операционные системы. Основную ОС (*Android, iOS*) дополняет вторая, низкоуровневая операционная система реального времени, обслуживающая оборудование для радиосвязи с базовыми станциями сотовых операторов. Именно эта вторая ОС считается довольно уязвимой с точки зрения безопасности.

Многие современные операционные системы, включая *Linux*, *macOS*, *iOS*, *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 (в формате *только текст*);
- программы часто используют текстовый ввод данных и вывод результатов;
- широко применяются утилиты, запускаемые из командной строки;
- 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-накопители, принтеры, сканеры) рассматриваются как файлы.

Все *UNIX*-подобные системы считаются очень надёжными с точки зрения безопасности. Достаточно сказать, что для них практически неактуальна проблема компьютерных вирусов.

Драйверы устройств

Драйвер (от англ. *driver* — водитель) — это программа специального типа, которая постоянно находится в оперативной памяти и обеспечивает обмен данными между ядром ОС и внешним устройством.

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 (рис. 6.70). Задача драйвера — преобразовать команды ввода/вывода в команды конкретного устройства.

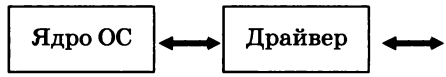


Рис. 6.70

Драйверы загружаются в память и фактически становятся частью ОС. Такая схема позволяет устанавливать и использовать устройства, которые были разработаны уже после выпуска операционной системы. Для подключения нового устройства нужно написать программу-драйвер. 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

Утилита — это служебная программа для проверки и настройки компьютера.

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 *macOS* используется программа *Finder*, а в операционной системе *Linux* — файловые менеджеры *Krusader*, *Midnight Commander* и др.;
- *антивирусные программы*: *Антивирус Касперского* (**www.kaspersky.ru**), *DrWeb* (**www.drweb.com**), *Nod32* (**www.eset.com**), *McAfee* (**home.mcafee.com**) и др.; бесплатные 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 *Windows* или *hexedit* (**rigaux.org/hexedit.html**) для *Linux*;
- программы для проверки различных устройств компьютера — оперативной памяти, жёсткого диска, мониторов и т. п. — *AIDA64* (**www.lavalys.com**), *Memtest* (**www.hcidesign.com**), *HDDScan* (**rlab.ru**), *Nokia Monitor Test*;
- сетевые утилиты для проверки связи в локальной и глобальной сетях; например, утилиты *ping*, *traceroute* (*tracert*), *nslookup* в *Windows* и *Linux*.

Часто к утилитам относят также:

- программы для *записи CD и DVD*; 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-накопителях. Однако жёсткий диск и флэш-накопитель «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 папка. 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. 6.71).

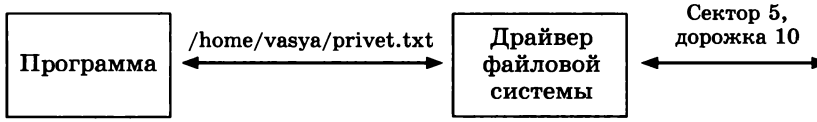


Рис. 6.71

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»).

Диски большой ёмкости состоят из огромного количества секторов, и для ОС трудно отслеживать состояние каждого отдельного сектора. Поэтому для размещения файлов используют более крупные блоки — **кластеры**, состоящие из нескольких секторов[1)].

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,

[1)] В современных ОС (2016 г.) один кластер может содержать от 1 до 128 секторов, так что его размер составляет от 512 байт до 64 Кбайт.

значительная часть места пропадает впустую. 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 произошёл 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 пользователей);
- позволяет задать *квоту* (ограничение) на использование диска каждому пользователю;
- позволяет использовать сжатие файлов и папок без дополнительных программ.

В состав *Linux* входят драйверы для работы с файловыми системами *FAT32* и *NTFS*.

В операционной системе *macOS* 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 (папки), а сами каталоги вложены друг в друга. Такая структура называется **деревом каталогов**.

В операционной системе *Linux* существует один корневой каталог (обозначаемый «/»), 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. 6.72).

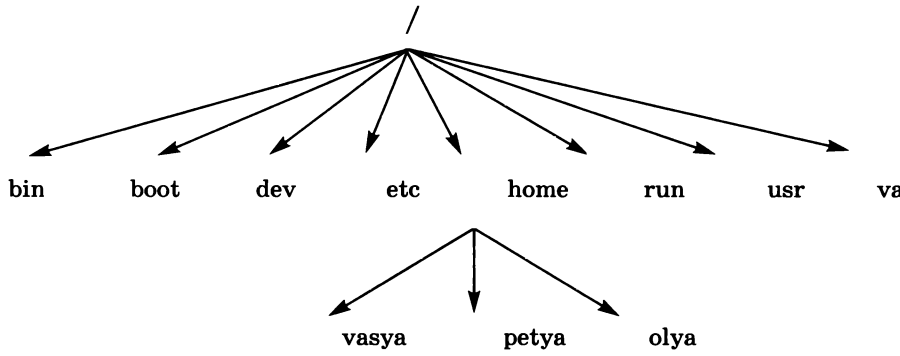


Рис. 6.72

Вот что хранится в каталогах, показанных на схеме на рис. 6.72:

- *bin* — команды операционной системы;
- *boot* — ядро ОС и данные для загрузки;
- *dev* — файлы устройств, подключенных к ОС; устройства присоединяются к файловой системе (монтируются) с помощью специальной команды;
- *etc* — файлы с настройками ОС и некоторых программ;
- *home* —домашние каталоги пользователей;
- *run* — каталог, в котором монтируются внешние носители данных; например, для пользователя **vasya** флэш-накопитель нужно искать по адресу */run/media/vasya/<имя накопителя>*;
- *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 «/» (он называется «слэш»). Например, адрес домашнего каталога пользователя *petya* запишется как */home/petya*, а адрес файла *qq.txt* в этом каталоге — как */home/petya/qq.txt*.

К файлу можно обратиться по относительному адресу — относительно текущего каталога. Например, при обращении из каталога */home/olya* адрес файла *qq.txt* запишется так: *../petya/qq.txt*. Здесь две точки означают переход в каталог более высокого уровня.

Дерево каталогов в операционной системе *Windows* строится отдельно для каждого диска. Диски обозначаются латинскими буквами (A, B, C, ...) — рис. 6.73.

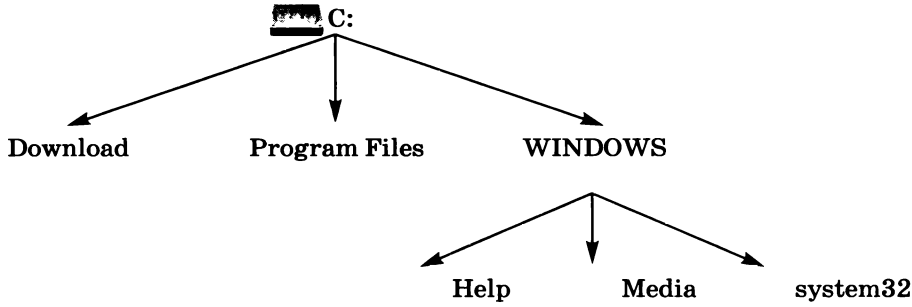


Рис. 6.73

В качестве разделителя при записи адреса файла или каталога (в *Windows* каталог также называют *папкой*) 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 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 1 символа;
`*x*.??*`	файлы, в имени которых есть буква «x», а расширение содержит не менее 2 символов;
`*z.a?`	файлы, имя которых заканчивается на букву «z», а расширение начинается с буквы «a» и состоит из 2 символов.

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

отчёт2011.doc
Самый_важный_отчёт.docx
Новый_отчёт_март_2011.docx

В операционной системе *Windows* прописные и строчные буквы в названиях файлов и папок 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*, *macOS*) 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.

Выводы

- Операционная система (ОС) — это комплекс программ, обеспечивающих пользователю и прикладным программам удобный интерфейс (способ обмена данными) с аппаратными средствами компьютера.
- Все современные ОС многозадачные.
- В состав операционной системы обычно входят: начальный загрузчик, система управления памятью, система ввода/вывода, командный процессор, утилиты.
- ОС мобильных устройств разработаны специально для маломощного оборудования.
- ОС реального времени обеспечивают выполнение задачи в течение заданного интервала времени.
- Драйвер — это программа специального типа, которая постоянно находится в оперативной памяти и обеспечивает обмен данными между ядром ОС и внешним устройством.
- Утилита — это служебная программа для проверки и настройки компьютера.
- Файловая система — это порядок размещения, хранения и именования данных на носителе информации.
- Для работы с группами файлов применяются маски (шаблоны). Маска может включать два специальных символа: знак «*» заменяет любое количество любых символов, а знак «?» — один любой символ.

Нарисуйте в тетради интеллект-карту этого параграфа.

Вопросы и задания

1. Зачем нужны операционные системы? Можно ли обойтись без них?
2. Сравните задачи, которые решают ОС, утилиты, драйверы и прикладные программы.
3. Как обеспечивается многозадачность на компьютерах с одним процессором?
4. Какими достоинствами и недостатками обладает система *Linux*?
5. Чем различаются ОС для мобильных устройств и ОС для стационарных компьютеров?
6. Зачем нужны операционные системы реального времени? Где они применяются?
7. Сравните принципы работы *UNIX*-подобных ОС и ОС *Windows*.
8. Как ОС обменивается данными с внешними устройствами? В чём достоинства такой схемы?
9. Что происходит, когда ОС обнаруживает новое устройство?
10. Зачем нужна файловая система? Обязательна ли она для компьютеров?
11. Почему невыгодно хранить мелкие файлы в файловой системе с большим размером кластера?
12. Какие достоинства и недостатки имеет журналирование в файловой системе?
13. Чем различаются одноуровневые и многоуровневые файловые системы?
14. 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»

Проекты

а) Сравнение файловых систем
б) Сравнение ОС для мобильных устройств

§ 43
Системы программирования

Ключевые слова:

- машинно-ориентированный язык
- ассемблер
- язык высокого уровня
- транслятор
- интерпретатор
- компилятор
- переносимость программ
- виртуальная машина
- байт-код
- компоновщик
- отладчик
- профилировщик
- динамически подключаемая библиотека
- интерфейс программирования приложений
- быстрая разработка приложений

Зачем нужны системы программирования?

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

```
B82301052500
```

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

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 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.

Чтобы переводить программу с такого языка в машинные коды, используют программы-**ассемблеры** (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**. 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.

Вспомним, что процессор может выполнить только программу, написанную в машинных кодах. Поэтому возникает задача: перевести программу, написанную на языке высокого уровня, в машинные коды. Для этого 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 году написала программу для Аналитической машины Чарльза Бэббиджа. В 1979 году в США был разработан язык программирования *Ада*, названный в её честь. Один из первых языков высокого уровня — *Фортран* — создан в 1957 году, 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.

По области применения языков программирования выделяют:

- профессиональные языки общего назначения: *Java, C, C++, C#, Visual Basic, Delphi, Python*;
- 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*.

Из всего множества языков программирования только 20–30 широко используются на практике. Большинство из этих языков — **императивные** (от англ. *imperative* — приказ, повелительное наклонение). Программа, написанная на императивном языке, — это список инструкций (команд) для компьютера. Эти команды должны выполняться последовательно одна за другой, при выполнении следующей команды результаты всех предыдущих можно прочитать из памяти, после выполнения очередной команды её результаты могут быть записаны в память. Программист должен подробно описать, как решить задачу и как представить результат. Ключевое понятие в императивном языке — это переменная, значение которой изменяется во время выполнения программы.

Существует и другое направление — другая **парадигма программирования — декларативное программирование**. В 1970-х годах для решения задач искусственного интеллекта был разработан язык логического программирования Prolog. Программа на языке Prolog — это не алгоритм, а фактически только формулировка задачи. Нужно описать исходные данные, правила, которые используются для решения, и определить цель (что нужно найти). Дальше Prolog-машина сама решает задачу на основе своего внутреннего алгоритма, обрабатывая введённые факты и правила, т. е. пытается сама получить нужный вывод из имеющихся данных. К сожалению, это удаётся сделать не всегда. Сейчас эти языки используются довольно редко, в основном в среде научных работников.

Разновидностью декларативных языков часто считают языки **функционального программирования**. В них основное понятие — это функция, вся программа представлена как набор функций, вызывающих друг друга.

Результат работы функции должен зависеть только от переданных ей параметров, т. е. «правильная» (или, как говорят,

«чистая») функция не должна использовать операции ввода/вывода и данные из памяти. В некоторых функциональных языках вообще нет переменных. Такой подход позволяет повысить надёжность программ, но при этом возникают проблемы с организацией ввода и вывода данных. Примеры функциональных языков — *Haskell*, *Scala*, *F#*, *LISP*, *Scheme*. Они достаточно редко применяются в коммерческих разработках и популярны главным образом в научной среде.

Трансляторы

Основа любой системы программирования — транслятор.

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

Существует два типа трансляторов — *интерпретаторы* и *компиляторы*.

Интерпретатор анализирует текст программы по частям. Разобрав очередной фрагмент, он немедленно выполняет описанные в нём действия и затем 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 запустить, 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**, P-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.

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 этой 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.

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

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 6.74 показано, как собирается программа на языке Си, состоящая из двух модулей (исходные файлы *qq.c* и *qq1.c*);

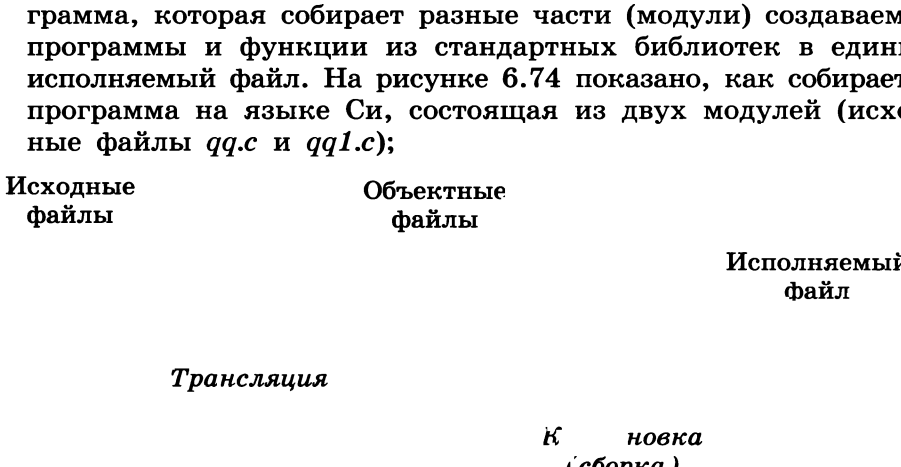


Рис. 6.74

- **отладчик** (англ. *debugger*[1]) — программа для поиска ошибок в других программах; отладчик позволяет:
  - — 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 часть программы 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.

---

[1] Согласно одной из версий, это название связано с жучком (англ. *bug*), который попал между контактами реле компьютера Mark II в 1947 году. Дословно *debug* — «удаление жучков».

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. 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 программирования приложений** (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 для *Unix*). Многие популярные сервисы Интернета, например *Google* (**www.google.ru**), *Яндекс* (**yandex.ru**), *Википедия* (**ru.wikipedia.org**), *Twitter* (**twitter.com**), *ВКонтакте* (**vk.com**), также публикуют свои API, позволяющие разработчикам сайтов безопасно использовать возможности этих сервисов.

Сейчас для разработки программ чаще всего применяют **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 6.75 показано окно RAD-среды *Lazarus* для программирования на объектной версии языка Паскаль.

Среди профессиональных *RAD*-сред нужно в первую очередь назвать *Microsoft Visual Studio*. Это коммерческая программа, но все желающие могут скачать и использовать бесплатную версию (*Community Edition*) с ограниченной лицензией.

Большой популярностью пользуется также среды *Dev-C++* и *Delphi*. Кроссплатформенная среда *Code::Blocks* распространяется бесплатно, существуют её версии для *Windows*, *macOS* и *Linux*.

Рис. 6.75

Выводы

- Системы программирования — это программные средства для создания и отладки новых программ.
- Транслятор — это программа, которая переводит в машинные коды тексты программ, написанных на языке высокого уровня.
- Псевдокод (байт-код) — это программа, предназначенная для выполнения с помощью виртуальной машины.
- Компоновщик (редактор связей, сборщик) — это программа, которая собирает разные части (модули) создаваемой программы и функции из стандартных библиотек в единый исполняемый файл.
- Отладчик — это программа для поиска ошибок в других программах.

- Профилировщик — это программа, позволяющая оценить время работы каждой процедуры и функции.
- Динамически подключаемая библиотека — это библиотека, которую могут использовать несколько одновременно работающих программ.
- Интегрированная среда разработки обычно включает текстовый редактор, транслятор, компоновщик, отладчик и профилировщик.
- Среды быстрой разработки приложений позволяют строить интерфейс программы с помощью мыши.

Нарисуйте в тетради интеллект-карту этого параграфа.

Вопросы и задания

1. Что такое машинный код?
2. Зачем нужны системы программирования? Можно ли обходиться без них?
3. Почему язык ассемблера называется машинно-ориентированным?
4. Сравните возможности машинно-ориентированных языков и языков высокого уровня.
5. Сравните два типа трансляторов. В чём их достоинства и недостатки?
6. Зачем нужен компоновщик?
7. Сравните задачи, которые решаются с помощью отладчика и профилировщика.
8. В чём вы видите достоинства и недостатки интегрированных сред разработки программ?
9. За счёт чего среды быстрой разработки (*RAD*-среды) ускоряют создание программ?

Подготовьте сообщение

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

Проекты

а) Использование профилировщика
б) Использование *API* веб-сайтов

в) Разработка интерпретатора
г) Использование языка Пролог
д) Использование языков функционального программирования

Интересные сайты

tiobe.com/tiobe_index — рейтинг языков программирования
visualstudio.com — среда быстрой разработки *Microsoft Visual Studio*
codeblocks.org — бесплатная кроссплатформенная среда *Code::Blocks* для программирования на языках *C*, *C++*, *Fortran*
wxdsgn.sourceforge.net — бесплатная среда *wxDevC++* для программирования на языке *C++*
wingware.com/downloads/wingide-101 — бесплатная среда для программирования на языке *Python*
lazarus-ide.org — бесплатная кроссплатформенная *RAD*-среда *Lazarus* для программирования на объектной версии языка Паскаль
ideone.com — веб-среда для отладки программ на разных языках программирования
codepad.org — веб-среда для отладки программ на разных языках программирования
www.onlinegdb.com — веб-среда для отладки программ на разных языках программирования

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»

Практические работы к главе 6

Работа № 18. «Инсталляция программ»
Работа № 19. «Сканирование и распознавание текстов»
Работа № 20. «Возможности текстовых процессоров»
Работа № 21. «Набор математических текстов (текстовые процессоры)»
Работа № 22. «Набор математических текстов (LaTeX)»
Работа № 23. «Оформление реферата»
Работа № 24. «Коллективная работа над документами»
Работа № 25. «Знакомство с программой Scribus»
Работа № 26. «Знакомство со средой SciLab»
Работа № 27. «3D-моделирование в программе *КОМПАС*»
Работа № 28. «Чертежи в программе *КОМПАС*»
Работа № 29. «Пакеты прикладных программ по специализации»
Работа № 30. «Пакеты прикладных программ по специализации»
Работа № 31. «Знакомство с аудиоредактором»
Работа № 32. «Знакомство с видеоредактором»
Работа № 33. «Онлайн-сервисы для разработки презентаций»

Глава 7
КОМПЬЮТЕРНЫЕ СЕТИ

§ 44
Основные понятия

Ключевые слова:

- топология сети
- общая шина
- звезда
- кольцо
- протокол
- сервер
- клиент

Компьютерная сеть — это группа компьютеров, соединённых линиями связи.

Все устройства, которые подключены к сети, называются узлами сети (по аналогии с узлами рыболовной сети). Кроме компьютеров к ним относятся вспомогательные устройства, участвующие в передаче данных.

Персональные сети (англ. **PAN**: *Personal Area Network*) объединяют устройства, принадлежащие одному человеку. Самый известный стандарт таких сетей — *Bluetooth*.

Локальные сети (англ. **LAN**: *Local Area Network*), как правило, объединяют компьютеры в одном или нескольких соседних зданиях.

Доступ компьютеров в сеть Интернет из локальной сети происходит через сеть фирмы-провайдера — поставщика услуг (англ. **WAN**: *Wide Area Network*).

Глобальная сеть Интернет — это объединение сетей различных провайдеров.

Структуры (топологии) сетей

Для объединения компьютеров в сеть используют три основные структуры — **топологии**: общую шину, звезду и кольцо.

Шина — это единая линия связи, которую несколько устройств используют для обмена данными. В схеме **«общая шина»** компьютеры (рабочие станции) подключены к одному кабелю с помощью специальных разъёмов (рис. 7.1).

Рис. 7.1

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 идет по шине, получают все компьютеры, но каждый из них обрабатывает только те данные, которые ему предназначены.

Общая шина — самая простая и дешёвая схема, к такой сети легко подключать новые рабочие станции, при выходе из строя любого компьютера сеть продолжает работать.

Однако разрыв кабеля или выход из строя терминатора приводит к отключению всей сети. Данные плохо защищены — каждая рабочая станция имеет доступ ко всем данным, которые идут по сети. Поскольку используется один канал связи, при увеличении числа компьютеров (больше 10–15) падает скорость передачи. Для нормальной работы сети общий объём передаваемых данных не должен превышать 30–40% пропускной способности шины. Возможны конфликты, когда две рабочие станции одновременно хотят передать данные по шине.

Общая шина считается устаревшей структурой и в современных локальных сетях практически не применяется. Однако беспроводные сети фактически используют именно общую шину, потому что все устройства работают в радиоэфире в одном диапазоне частот.

В схеме **«звезда»** (рис. 7.2) есть центральное устройство, через которое идёт весь обмен данными. Обычно в центре находится **коммутатор** (его часто называют «свитч», от англ. *switch* — переключать). Коммутатор передаёт принятые данные только адресату, а не всем компьютерам в сети.

При выходе из строя любой рабочей станции сеть, построенная по схеме «звезда», остаётся работоспособной. Поскольку все точки подключения собраны в одном месте (это порты коммутатора), сетевому администратору легко искать неисправности и обрывы кабеля. Каждая рабочая станция получает только «свои» данные, поэтому обеспечивается высокий уровень безопасности.

В то же время такая структура приводит к большому расходу кабеля, что повышает её стоимость. Важнейшую роль играет

Рис. 7.2

надёжность работы коммутатора — если он выйдет из строя, то сеть не будет работать.

Если портов коммутатора не хватает для подключения всех рабочих станций, то несколько частей (*сегментов*) сети можно объединить в иерархическую структуру (она называется «расширенная звезда») с помощью дополнительных коммутаторов (рис. 7.3).

Рис. 7.3

В схеме **«кольцо»** (рис. 7.4) каждый компьютер соединён с двумя соседними, причём от одного он только получает данные, а другому только передаёт. Таким образом, данные 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.

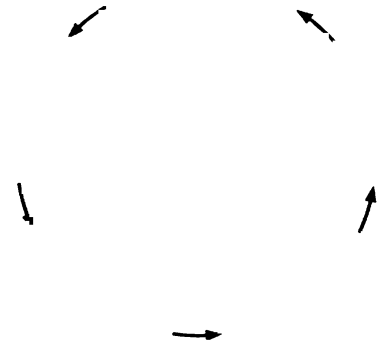


Рис. 7.4

Каждый компьютер участвует в передаче сигнала и усиливает его, поэтому размер сети может быть очень велик (до 20 км), ограничено лишь расстояние между соседними узлами (для оптоволоконных сетей — до 2 км). Такая структура обеспечивает надёжную работу при большом потоке данных, конфликты практически невозможны. Кроме того, не нужно дополнительное оборудование — коммутаторы.

Поскольку все данные передаются по кольцу через несколько компьютеров, в кольце трудно обеспечить безопасность данных. Для подключения нового узла приходится останавливать всю сеть. Довольно сложно настраивать кольцевую схему и искать неисправности в её работе.

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 схему «кольцо».

Обмен данными

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 нужно 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 очередного 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). По сети одновременно передаются пакеты, принадлежащие нескольким файлам. На рисунке 7.5 узлы 1, 2, 5 и 6 — это компьютеры, на которых работают пользователи, а узлы 3 и 4 — специальные сетевые компьютеры, которые называются **маршрутизаторами**. По одной линии связи (между узлами 3 и 4) одновременно выполняется передача данных от узла 2 к узлу 5 (эти пакеты обозначены чёрными прямоугольниками) и от узла 1 к узлу 6 (белые прямоугольники).

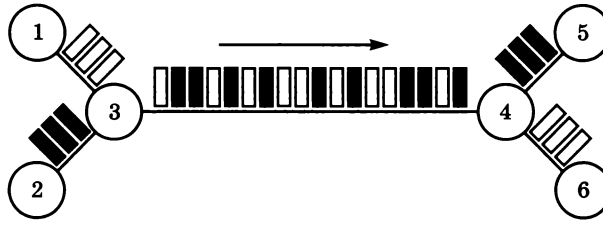


Рис. 7.5

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 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 и пр. 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 1500 байт.

Серверы и клиенты

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)**. Они включены постоянно и чаще всего 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 клиентов.

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).

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 результаты 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.

Во многих организациях применяют **терминальные серверы** — мощные компьютеры, которые предоставляют пользователям свои ресурсы (процессорное время, оперативную и дисковую память) и

рабочий стол. Рабочие станции (**терминалы**, или **тонкие клиенты**) выполняют только две задачи:

- передают серверу данные, введённые пользователем с помощью клавиатуры и мыши;
- выводят на экран изображение рабочего стола, полученное от сервера.

Эти задачи не требуют сложных расчётов, поэтому в качестве терминалов можно использовать маломощные и устаревшие компьютеры.

Выводы

- Существуют три основных структуры (топологии) сетей: общая шина, звезда и кольцо. Большинство современных локальных сетей строятся по схеме «звезда».
- Топология «кольцо» используется для соединения коммутаторов.
- Протокол — это набор правил и соглашений, определяющих порядок обмена данными в сети.
- Данные, передаваемые по сети, делятся на фрагменты (пакеты). Это позволяет одновременно передавать по одному каналу несколько сообщений. В случае сбоя повторно передаётся только один сбойный пакет.
- Сервер — это компьютер, предоставляющий свои ресурсы (файлы, программы, внешние устройства и т. д.) в общее использование. Клиент — это компьютер, использующий ресурсы сервера. Понятия «сервер» и «клиент» относят также к программам.
- Терминальный сервер — это мощный компьютер, который предоставляет пользователям свои ресурсы (процессорное время, оперативную и дисковую память) и рабочий стол.

Нарисуйте в тетради интеллект-карту этого параграфа.

Вопросы и задания

1. Может ли один компьютер выполнять роли сервера и клиента?
2. Что такое протокол? Зачем нужны протоколы?
3. Зачем данные, передаваемые по сети, делятся на пакеты?
4. Почему размер пакета не должен быть очень маленьким?
5. Сравните достоинства и недостатки сетей со структурами «общая шина», «звезда» и «кольцо».
6. Какую структуру вы предложили бы использовать для школьной сети (рассмотрите разные ситуации)?

Подготовьте сообщение

а) «Пакетная передача данных»
б) «Технология "клиент — сервер"»
в) «Протоколы Интернета»

Проект

Обмен данными между программами по схеме «клиент — сервер»

§ 45
Локальные сети

Ключевые слова:

- шлюз
- одноранговая сеть
- выделенный сервер
- серверная ОС
- терминальный доступ
- беспроводная сеть
- Ethernet
- сетевая карта
- коммутатор
- маршрутизатор

Локальной обычно называют сеть, которая объединяет компьютеры в одном или нескольких соседних зданиях. Для связи с другими сетями компьютеры локальной сети используют специальный узел сети, который называют **шлюзом** (англ. *gateway*).

Чтобы организовать локальную сеть, компьютеры должны работать под управлением **сетевой операционной системы**, которая поддерживает:

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

Эти возможности существуют во всех современных ОС (*Windows, Linux, macOS* и др.).

Сетевое оборудование

Для обмена данными в современных локальных кабельных сетях используется семейство стандартов, которое называется **Ethernet** (от лат. *aether* — эфир[1]). Для связи компьютеров могут применяться электрические кабели или оптоволокно. Все передаваемые данные делятся на небольшие блоки — **фреймы**, каждый фрейм

[1] Слово «эфир» связано с принципом этой технологии — все сигналы, передаваемые одним узлом, принимаются всеми остальными (как при радиовещании).

содержит адреса источника и получателя, а также контрольную сумму, позволяющую обнаруживать ошибки. Существующий стандарт определяет скорости передачи данных[1] до 100 Гбит/с.

Для того чтобы можно было подключить компьютер к кабельной сети, он должен иметь **сетевую карту** (сетевой адаптер, англ. *network interface card*, рис. 7.6, *а*). В современных материнских платах настольных компьютеров и в ноутбуках обычно уже есть встроенная сетевая карта, поддерживающая стандарт *Ethernet* со скоростью до 1 Гбит/с.

а *б* *в*

Рис. 7.6

Для локальных сетей чаще всего используется восьмижильный кабель **«витая пара»**, который представляет собой четыре пары скрученных проводов (рис. 7.6, *б*). Восьмиконтактный разъём с защёлкой часто называют **RJ-45** (рис. 7.6, *в*). Сейчас наиболее распространены сети со скоростью 100 Мбит/с, построенные с помощью кабеля «витая пара» по схеме «звезда».

Для передачи данных на большие расстояния применяют **оптоволоконные кабели**, в которых информация передаётся с помощью светового луча. Свет идет внутри кабеля, отражаясь от стенок стеклянного или пластикового цилиндра-световода (рис. 7.7).

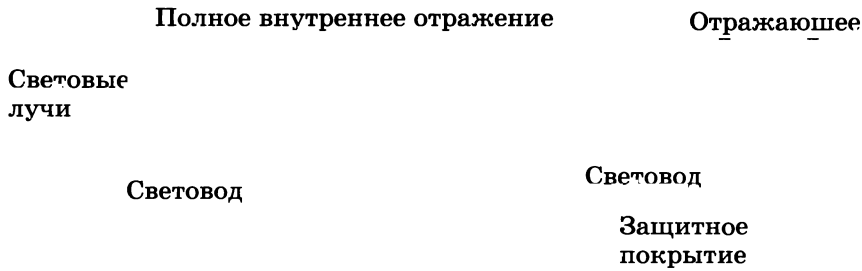


Рис. 7.7

[1] При указании скорости передачи данных используются десятичные приставки (а не двоичные, как при измерении количества информации), например 1 Мбит/с = 10^6 бит/с.

Коммутаторы (рис. 7.8) используются для объединения компьютеров в единую сеть по схеме «звезда», которая чаще всего применяется на практике.

Рис. 7.8

Компьютер соединяют с коммутатором отрезком кабеля с двумя разъёмами RJ-45, который называется **патч-корд** (от англ. *patching cord* — соединительный шнур).

Обычно все компьютеры, входящие в локальную сеть, получают доступ в Интернет через один канал связи. Для связи локальной сети с Интернетом необходим **маршрутизатор** (роутер, от англ. *router*) — специальный сетевой компьютер. В отличие от коммутатора маршрутизатор пересылает пакеты данных между различными подсетями, а не внутри одной подсети.

Задача маршрутизатора — определить дальнейший путь 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 сеть и сеть провайдера.

При создании беспроводных сетей компьютеры должны иметь **адаптеры Wi-Fi**, они обычно встроены в современные мобильные устройства. Если встроенного адаптера нет, можно использовать дополнительный адаптер, который подключается к USB-порту (рис. 7.9, *а*). Для связи компьютеров в беспроводной сети и для обеспечения доступа в Интернет используют точки доступа (рис. 7.9, *б*) и беспроводные маршрутизаторы (рис. 7.9, *в*).

а *б* *в*

Рис. 7.9

Одноранговые сети

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. Каждому пользователю можно предоставить свои права для работы с ресурсом (например, разрешить только чтение или предоставить полный доступ).

Для создания одноранговой сети не нужна дорогая аппаратура и сложное программное обеспечение, их просто настраивать. Компьютеры независимы друг от друга, каждый из них в случае отказа сети может работать автономно.

В одноранговой сети нет единого центра управления, поэтому на каждом компьютере приходится создавать учётные записи для всех пользователей, которые могут на нём работать. Данные хранятся, как правило, на компьютерах пользователей, поэтому пользователь должен делать резервные копии самостоятельно.

Сети с выделенными серверами

С увеличением количества компьютеров (больше 10–15) становится очень сложно управлять одноранговыми сетями, а также обеспечивать безопасность данных. Поэтому в крупных организациях применяют **сети с выделенными серверами**, в которых один или несколько мощных компьютеров играют роль **серверов** (пользователи на них не работают), а остальные (**клиенты, рабочие станции**) 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 требуется специальная **серверная операционная система** (*Windows Server, Linux, FreeBSD, Solaris*). В таких ОС основное внимание уделяется стабильной и надёжной 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 серверных 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, планшетные компьютеры, смартфоны) могут легко подключаться к сети и получать доступ 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 настольный и планшетный компьютеры, мобильный телефон, ноутбук, принтер, цифровой фотоаппарат, мышь, клавиатура, наушники.

Радиус действия такой сети обычно[1)] не более 20 м, он зависит от мощности передатчиков, а также от преград и помех.

Теоретически максимальная скорость обмена данными может достигать 24 Мбит/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*

1) Стандарт предусматривает радиус действия до 100 м.

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. 7.10).

Интернет

Рис. 7.10

Все современные операционные системы для персональных компьютеров и мобильных устройств поддерживают технологию и устройства *Wi-Fi*.

Согласно стандарту, скорость передачи данных в сетях *Wi-Fi* может быть от 0,1 Мбит/с до 480 Мбит/с, радиус действия 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 доступ к Интернету 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.

Выводы

- Локальной обычно называют сеть, которая объединяет компьютеры в одном или нескольких соседних зданиях. Для связи с другими сетями компьютеры локальной сети используют специальный узел сети, который называют шлюзом.
- В одноранговой сети все компьютеры равноправны, каждый может выступать как в роли клиента, так и в роли сервера.

- В крупных сетях применяют выделенные серверы, на которых не работают пользователи. Остальные компьютеры (рабочие станции) используют их ресурсы.
- Чаще всего используются два стандарта беспроводных сетей — *Bluetooth* (персональные сети) и *Wi-Fi* (локальные сети). Главная проблема беспроводных сетей — обеспечение безопасности данных.
- К сетевому оборудованию относят сетевые карты, сетевые кабели, коммутаторы, маршрутизаторы.
- В отличие от коммутатора маршрутизатор пересылает пакеты данных между различными сегментами сети, а не внутри одного сегмента. Дальнейший маршрут пакета определяется с помощью таблиц маршрутизации.

Нарисуйте в тетради интеллект-карту этого параграфа.

Вопросы и задания

1. Сравните возможности одноранговых сетей и сетей с выделенными серверами. Какой тип сети, на ваш взгляд, лучше использовать в школе?
2. Чем отличаются серверные ОС от клиентских?
3. В чём достоинства и недостатки терминального доступа?
4. Назовите преимущества и недостатки беспроводных сетей.
5. Как обеспечивается защита данных в беспроводных сетях?
6. Найдите в Интернете сведения о происхождении названия «Bluetooth».
7. Что такое точка доступа? Зона доступа *Wi-Fi*?
8. Какое сетевое оборудование необходимо для кабельных сетей?
9. Сравните функции коммутатора и маршрутизатора.
10. Какие оборудование необходимо для создания беспроводной сети?

Подготовьте сообщение

а) «Серверные операционные системы»
б) «Что такое терминальный сервер»
в) «Стандарт Ethernet»
г) «Сети Bluetooth»
д) «Сети Wi-Fi»
е) «Защита данных в беспроводных сетях»

Проекты

а) Обмен данными с помощью беспроводных технологий
б) Управление роботами с помощью беспроводных технологий

§ 46
Сеть Интернет

Ключевые слова:

- провайдер
- распределённая сеть
- пакет
- протоколы TCP/IP
- маршрутизатор

Что такое Интернет?

Интернет — это глобальная компьютерная сеть.
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, которая предоставляет услуги связи, в том числе подключение к Интернету. 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 24 Мбит/с, однако на телефонной станции необходимо устанавливать дополнительное оборудование, которое разделяет низкочастотный телефонный сигнал и высокочастотный сигнал, передающий цифровые данные;
- через *кабельную сеть провайдера* (если она существует в вашем доме); в этом случае телефонная линия не задействована;
- с помощью *оптических сетей* с высокой пропускной способностью (англ. **PON**: *Passive Optical Network* — пассивные оптические сети); в таких сетях для передачи данных со скоростью до 2,5 Гбит/с используются оптоволоконные кабели и оптические разветвители, которые не требуют питания и обслуживания;
- с помощью *беспроводных модемов* (*USB-модемов*, рис. 7.11), 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.

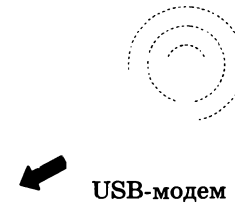


Рис. 7.11

Краткая история

В 1960-е годы 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;
- в сети нет единого центра, она состоит из отдельных ячеек (рис. 7.12), это обеспечивает живучесть в случае выхода из строя любого узла; такая сеть называется **распределённой**;

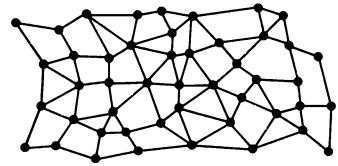


Рис. 7.12

- пакетная передача данных: передаваемые данные разбиваются на пакеты небольшого размера, которые передаются независимо друг от друга; одна линия связи используется для одновременной передачи нескольких потоков данных.

В 1969 году состоялся первый обмен данными по сети между компьютерами, установленными в Калифорнийском университете и Стэнфордском исследовательском центре на расстоянии 640 км друг от друга. В 1971 году была создана программа для работы с электронной почтой, которая сразу стала очень популярной. Начиная с 1973 года, к новой сети подключаются университеты и колледжи не только США, но и Европы.

В 1984 году в США была создана межуниверситетская сеть Национального фонда науки *NSFNet* (англ. *National Science Foundation Network*), которая объединяла более мелкие сети научных организаций и имела гораздо бóльшую пропускную способность, чем *ARPANET*. Именно эта сеть и стала впоследствии называться «интернетом», а сеть *ARPANET* к 1990 году прекратила своё существование.

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

В 1991 году британский учёный 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 популярной службой Интернета. В 1993 году был создан браузер с графическим интерфейсом *Mosaic* — первая программа для просмотра страниц с гипертекстом.

Тим Бернес-Ли
род. в 1955 г.

Набор протоколов TCP/IP

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 набор протоколов **TCP/IP**, разработанный в середине 1970-х годов. Название TCP/IP происходит от двух протоколов — **TCP** (англ. *Transmission Control Protocol* — протокол управления передачей) и **IP** (англ. *Internet Protocol* — межсетевой протокол).

В Интернете используется четырёхуровневая система протоколов, каждый из которых «занимается своим делом»:

1) **уровень приложения** (англ. *Application Layer*) — формат запросов и ответов, которыми обмениваются программы;
2) **транспортный уровень** (англ. *Transport Layer*) — правила пакетной передачи блоков данных без учета их содержания (протокол TCP);
3) **сетевой уровень** (англ. *Internet Layer*) — система адресов компьютеров, позволяющая автоматически выбирать маршрут для отдельных пакетов без гарантии их доставки (протокол IP);
4) **канальный уровень** *(англ. Link Layer)* — правила передачи данных по кабельной, оптоволоконной или другой линии связи.

Попробуем разобраться, почему для работы в Интернете нужно использовать несколько протоколов. Предположим, что браузер на компьютере А запрашивает веб-страницу с сервера, который находится на компьютере Б. На рисунке 7.13 показана схема обмена данными между двумя компьютерами, А и Б, которые находятся в разных сетях и связаны через два маршрутизатора[1] — $М_1$ и $М_2$.

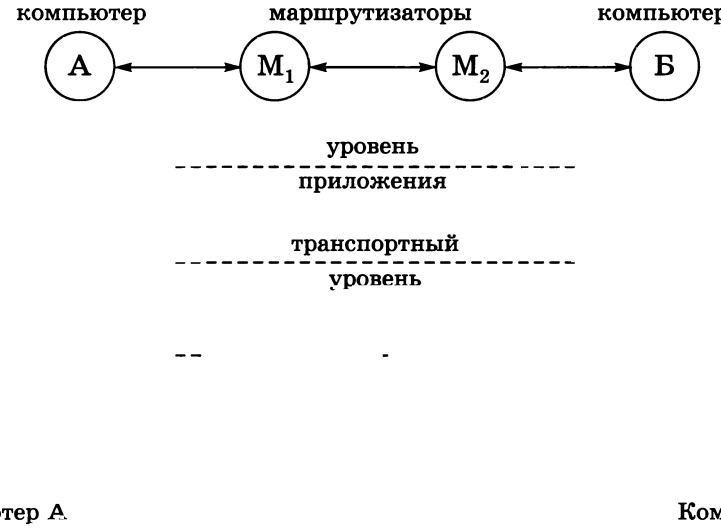


Рис. 7.13

«Разговор» между браузером и сервером идёт 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 гарантированную доставку данных. Передаваемый блок данных разбивается на фрагменты (каждый такой фрагмент называется **дейтаграммой**, от англ. *datagram*), и эти фрагменты передаются на сетевой уровень — **драйверу протокола IP**, который строит **IP-пакеты**: добавляет к дейтаграммам заголовки со служебной информацией (рис. 7.14).

---

[1] Маршрутизаторы могут быть связаны не напрямую, а через другие маршрутизаторы, но мы рассмотрим самый простой случай.

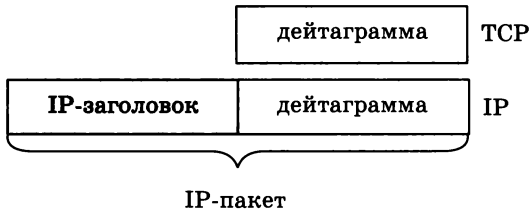


Рис. 7.14

IP-протокол определяет правила построения IP-пакетов и систему адресов компьютеров (IP-адресов), с помощью которой маршрутизаторы[1)] определяют маршруты движения пакетов. Каждый IP-адрес содержит номер сети, в которой находится компьютер, и числовой код компьютера в этой сети.

Обычно компьютеры А и Б напрямую не связаны, поэтому важно определить, куда нужно отправить пакет, чтобы он дошёл до компьютера Б. Все пакеты, идущие в другие сети, направляются специальному узлу — маршрутизатору M_1, который определяет их дальнейший маршрут. Выделив из IP-адреса номер сети, в которой находится адресат — компьютер Б, — он передаёт эти пакеты (например, через оптоволоконную или спутниковую линию связи) маршрутизатору M_2. Далее IP-пакеты поступают по локальной сети компьютеру Б как цепочки битов.

Протокол IP не гарантирует доставку пакетов, поэтому драйвер TCP должен (с помощью установленного соединения) проверить, что компьютер Б действительно получил данные, 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).

На уровне приложения (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.

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

---

1) Маршрутизаторы могут автоматически обмениваться специальной информацией, позволяющей исправить маршруты в случае изменения сети. Кроме того, маршруты может вручную определить сетевой администратор.

Выводы

- Пользователь получает доступ к Интернету через провайдера — фирму, которая предоставляет услуги связи. Для этого могут использоваться кабельные и оптоволоконные каналы и радиосвязь.
- Интернет — это распределённая сеть, у которой нет единого центра.
- Для обмена данными в Интернете используется семейство протоколов TCP/IP, которое включает четыре уровня: уровень приложения, транспортный, сетевой и канальный уровни.

Нарисуйте в тетради интеллект-карту этого параграфа.

Вопросы и задания

1. Сравните различные способы получения доступа в Интернет через провайдера.
2. Какие идеи были положены в основу глобальной компьютерной сети?
3. Чем различаются понятия «Интернет» и «Всемирная паутина»?
4. Объясните, почему применяются несколько уровней протоколов.
5. Какова роль узлов-маршрутизаторов?
6. Как обеспечивается гарантированная доставка сообщений в Интернете?
7. Назовите наиболее известные протоколы уровня приложения. 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) «Протоколы UDP и TCP»
е) «Тим Бернес-Ли и его вклад в развитие Интернета»

Проекты

а) Сравнение модели OSI и набора протоколов Интернета
б) Коллективная презентация «История Интернета»

§ 47
Адреса в Интернете

Ключевые слова:

- IP-адрес
- маска
- трансляция сетевых адресов
- система доменных имён
- домен верхнего уровня
- регистратор
- DNS-сервер
- URL (адрес ресурса)

IP-адреса и маски

В Интернете любые два компьютера могут связаться друг с другом. Для этого каждый из них должен иметь уникальный адрес. Свои адреса могут иметь персональные и мобильные компьютеры, маршрутизаторы, принтеры и другие устройства. Система адресов в Интернете определяется IP-протоколом, поэтому такие адреса называются **IP-адресами.**

IP-адрес присваивается не компьютеру, а сетевому интерфейсу — сетевой карте, адаптеру беспроводной сети и т. д. Поэтому компьютер может иметь несколько IP-адресов, например, если на нём установлены две сетевые карты. У маршрутизатора всегда есть несколько интерфейсов, имеющих свои IP-адреса, которые принадлежат разным сетям.

Компьютерам удобнее работать с числовыми адресами, на каждый из которых выделяется одинаковое место в памяти. IP-адреса представляют собой 32-битные числа, например:

$$3232262259 = 11000000101010000110100001110011_2$$

Для удобства обычно разбивают это число на группы из 8 двоичных разрядов (октеты):

11000000.10101000.01101000.01110011

и записывают каждую группу в десятичной системе счисления:

192.168.104.115

Минимальное возможное значение каждого из четырёх чисел — 0, а максимальное — $11111111_2 = 255$.) Точки нужны для однозначности, иначе, например, адрес 172161231 может быть прочитан по-разному: как 172.16.12.31, 17.216.123.1 или 17.21.61.231.

В этих числах закодированы номер сети и числовой адрес (код) компьютера в сети. Маршрутизаторы хранят адреса сетей в своих таблицах маршрутизации. Приняв пакет, маршрутизатор выделяет адрес сети из IP-адреса назначения и по нему пытается найти в таблице дальнейший маршрут для этого пакета.

Для того чтобы выделить эти две части из IP-адреса, используют шаблоны — маски. **Маска** — это тоже 32-битное число, которое можно записать как четыре числа в диапазоне [0; 255]. Двоичный код маски строится особым образом, по принципу «сначала единицы, потом — нули» (стандартом разрешены и другие маски, но на практике они не используются). Например, маска 255.255.255.0 в двоичном коде запишется так:

11111111.11111111.11111111.00000000

В ней сначала идут 24 единицы, а потом — нули. Это значит, что первые 24 бита адреса — номер сети, а оставшиеся 8 бит — код компьютера (узла) в этой сети. На рисунке 7.15 область, отведённая номеру сети, выделена штриховой рамкой. В данном случае номер сети — 192.168.104.0, а код узла — 115.

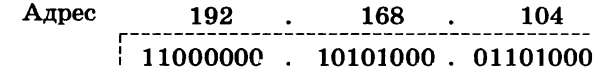


Рис. 7.15

Можно использовать другую запись, которая значит то же самое:

192.168.104.115/24

Здесь запись «/24» говорит о том, что в маске 24 единицы.

Хотя на номер узла отводится 8 бит, в такой сети может быть только 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 широковещательной рассылки: сообщение, отправленное по этому адресу, получают все компьютеры 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.

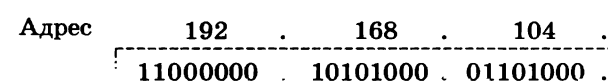


Рис. 7.16

Узел с адресом 192.168.104.115/29 — это узел с кодом 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.

Адреса 127.0.0.0–127.255.255.255 особые: они служат для обращения к своему компьютеру (обычно для этой цели применяют адрес 127.0.0.1).

Количество возможных 32-битных IP-адресов равно 2^{32} (это более 4 миллиардов), но их уже не хватает. Распределением IP-адресов занимается некоммерческая организация **IANA** (от англ. *Internet Assigned Numbers Authority* — Администрация адресного пространства Интернета). В 2011 году IANA выделила 5 последних блоков адресов региональным регистраторам.

Для того чтобы сэкономить адреса, договорились, что 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)

будут использоваться только в локальных сетях. Эти адреса называются **внутренними (частными**, англ. *private*). Остальные адреса называются **внешними (публичными**, англ. *public*).

Пакеты, в которых есть внутренние адреса, не выходят за пределы локальной сети. Поэтому компьютеры во многих локальных сетях могут использовать одни и те же внутренние адреса.

Пропуская пакет, идущий в Интернет от одного из компьютеров локальной сети, маршрутизатор заменяет его внутренний IP-адрес своим. Получив ответ сервера, он восстанавливает адрес отправителя по таблице открытых соединений и переправляет ему этот ответ. Такой механизм называется **трансляцией сетевых адресов** или **NAT** (англ. *Network Adress Translation*). Он позволяет всем компьютерам локальной сети выходить в Интернет, используя один внешний IP-адрес. Главный недостаток такого подхода — к компьютеру, имеющему только внутренний IP-адрес, невозможно обратиться из другой сети. Например, если разместить на нём веб-сервер, он будет доступен только из локальной сети.

Чтобы окончательно решить проблему нехватки IP-адресов, разработана новая (шестая) версия протокола IP, которая обозначается как **IPv6**. В ней на каждый адрес отводится 128 бит, а не 32, как сейчас. Адрес IPv6 записывается в виде восьми групп по 4 шестнадцатеричные цифры, разделённые двоеточиями, например:

2001:0DB8:11A3:09D7:1F34:8A2E:07A0:765D

Протокол IPv6 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 году была разработана **система доменных имен** (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).

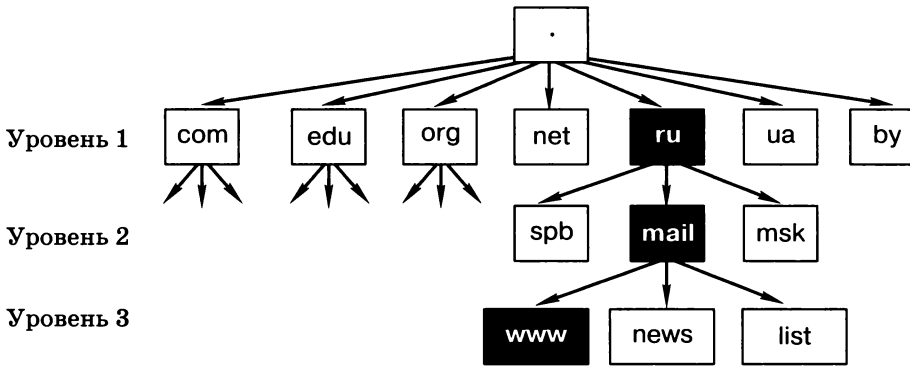


Рис. 7.17

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

Точка в корне дерева — это **корневой домен. Домены верхнего уровня** 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 верхнего уровня.

Распределением IP-адресов и доменов верхнего уровня занимается международная организация **ICANN** (англ. *Internet Corporation for Assigned Names and Numbers*). Российский домен **ru** был зарегистрирован в 1994 году. Кроме того, России принадлежит домен **su** (от англ. *Soviet Union* — Советский Союз).

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 **ucoz.ru** предоставляет всем желающим место под сайт и домен третьего уровня вида *ivanov.ucoz.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, ма-

[1] Здесь перечислены не все домены верхнего уровня.

ска 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, причём 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**). Вместе с тем несколько доменных имён могут быть связаны с одним IP-адресом (например, когда несколько небольших сайтов размещены на одном компьютере-сервере). Таким образом, соответствие между доменными именами и IP-адресами можно описать как «многие ко многим».

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 году создатель Всемирной паутины Т. 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** — маршрутизатора, который связывает локальную сеть с другими сетями. Ему 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*.

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

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.

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

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

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

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

в ОС *Windows* 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 11 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 `traceroute` можно определить, где именно нарушена связь.

Выводы

- IP-адрес — это 32-битное число, которое служит для обращения к узлу сети.
- IP-адрес присваивается не компьютеру, а сетевому интерфейсу — сетевой карте, адаптеру беспроводной сети и т. д. Поэтому компьютер может иметь несколько IP-адресов.
- Для удобства IP-адреса делятся на октеты (группы по 8 двоичных разрядов) и записываются в виде четырёх десятичных чисел в диапазоне от 0 до 255.
- IP-адрес содержит номер сети и числовой код узла в этой сети. Для того чтобы отделить одну часть от другой, используют маски.
- Маска в двоичной записи имеет структуру «сначала единицы, потом — нули». Часть, заполненная единицами, определяет номер сети.
- Существуют внутренние IP-адреса, которые используются только в локальных сетях.
- Трансляция сетевых адресов (NAT) позволяет использовать один внешний (публичный) IP-адрес для выхода в сеть всех компьютеров локальной сети.
- В связи с нехваткой свободных IP-адресов происходит постепенный переход на протокол IPv6.
- Система доменных имён (DNS) позволяет использовать символьные имена узлов. Таблицы на DNS-серверах содержат пары «IP-адрес — доменное имя». Одному IP-адресу может соответствовать несколько доменных имён, одно доменное имя может быть связано с несколькими IP-адресами для распределения нагрузки.
- URL — это точный адрес документа в Интернете. Он включает протокол, доменное имя сервера (или его IP-адрес), каталог и название файла.
- Для тестирования сети можно использовать утилиты `ping` (проверка связи с узлом) и `traceroute` (`tracert`, трассировка маршрута).

Нарисуйте в тетради интеллект-карту этого параграфа.

Вопросы и задания

1. Сколько места в памяти занимает IP-адрес?
2. Почему в сети с маской /24 может быть только 254 узла, а не 256?

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 имена?
9. В виде какой структуры можно представить доменную систему имён?
10. Какие бывают домены верхнего уровня?
11. Какие домены вы можете зарегистрировать (если они свободны)?
12. Какие функции выполняет DNS-сервер?
13. Приведите примеры URL для веб-страниц, рисунков, файлов на FTP-серверах.
14. Определите IP-адрес своего компьютера и маску подсети. Сколько компьютеров может быть в такой сети?

Подготовьте сообщение

а) «Протокол IPv6»
б) «Как работает DNS-сервер?»
в) «Как работает DHCP-сервер?»

Проекты

а) Программа для определения IP-адреса сети
б) DNS-сервер в локальной сети
в) DHCP-сервер в локальной сети

§ 48
Службы Интернета

Ключевые слова:

- веб-сервер
- каталог ссылок
- поисковая машина
- индекс
- FTP-сервер
- форум
- мессенджер
- пиринговая сеть
- информационная система

Всемирная паутина

Всемирная паутина (веб) — это служба для доступа к гипертекстовым документам (веб-страницам). Для просмотра веб-страниц используют программы-браузеры.

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

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, macOS*;
- **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).

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, 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 их «репутация» (или «карма»), появляется некоторое соперничество.

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

Появились специальные сайты, где пользователи могут вести **блоги** — личные сетевые дневники (**www.livejournal.com**, **www.blogspot.com**). Записи (**посты**) автора появляются в хронологическом порядке, поэтому блог можно использовать как доску объявлений (ленту новостей). Материалы блога можно обсуждать прямо в блоге — пользователи комментируют посты, автор отвечает на комментарии. Влияние блогов настолько возросло, что их стали приравнивать к средствам массовой информации. Людей, которые ведут блоги, называют **блогерами**.

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 таких 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 Т. 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 такой поиск позволит автоматически собирать всю информацию о личности (или организации), так что область «частного пространства», «личной тайны» значительно уменьшится.

Поиск 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 ссылок*.

Каталоги ссылок (англ. *web directory*) — это сайты, содержащие список ссылок на другие сайты с 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 году. Самый крупный из российских каталогов — **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, и часто случается, что нужный вам сайт не включён в каталог. Поэтому возникла естественная идея — заставить компьютерную программу искать новые сайты и автоматически анализировать информацию на их страницах. Так появились *поисковые системы*.

Поисковые системы — это сайты для поиска информации в Интернете по запросам пользователей. Работу поисковой системы обеспечивает специальное программное обеспечение — **поисковая машина**.

Ключевые слова — это слова, которые представляют содержание текста.

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

Робот-браузер поисковой машины (его часто называют **поисковым роботом** или **«пауком»**, 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.

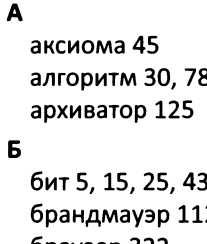


Рис. 7.18

Пользователь вводит в запросе ключевые слова, которые его интересуют. Поисковая система с помощью индекса находит те страницы, где встречаются эти слова.

Самая крупная международная поисковая система — **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. Поисковая система **TinEye (tineye.com)** позволяет находить изображения, похожие на образец.

При составлении запросов для поисковых систем используются следующие правила:

- простейший запрос — это перечисление ключевых слов;
- поисковые системы не различают прописные и строчные буквы;
- в запросах можно использовать логические операции & (И) и | (ИЛИ);
- если вам нужна точная фраза, нужно взять её в кавычки;
- чтобы исключить какое-то слово из поиска, перед ним ставят знак «минус».

Каждая поисковая система имеет свой язык составления запросов. С помощью этого языка можно, например:

---

1) Начальный список страниц обычно задают разработчики.

- искать только файлы нужного формата, например файлы в формате PDF;
- искать только на сайтах заданного домена;
- искать только страницы на заданном языке и т. п.

Нужно понимать, что информация, размещённая в Интернете, не всегда достоверна. Каждый может создать свой сайт и написать в нём всё, что угодно. В отличие от научных книг и журналов статьи в Интернете в большинстве своём никем не проверяются (*не рецензируются*), поэтому истинность информации остаётся целиком на совести автора.

Вообще говоря, проверить достоверность информации в Интернете очень сложно. Как же всё-таки это сделать?

Хорошо, если информация найдена на официальном сайте какой-либо организации, например правительства страны или города, фирмы, учебного заведения. Такие организации дорожат своим авторитетом, но даже на этих сайтах могут встречаться ошибки.

Сайты средств массовой информации (СМИ) должны указывать номер свидетельства о регистрации. За публикацию ложных сведений СМИ могут быть лишены лицензии, поэтому редакторы сайтов строго следят за правильностью информации.

Можно поискать на других сайтах похожую информацию (не скопированную слово в слово, а с тем же содержанием). Очень хорошо, если удаётся подтвердить полученные данные печатными источниками — материалами учебников, книг, научных статей.

Стоит проверить, считается ли автор материала специалистом в той области, о которой пишет. Можно доверять автору, который имеет учёную степень, например кандидата или доктора наук. Напротив, статьи с орфографическими ошибками явно не заслуживают доверия.

Для оценки достоверности информации важна авторитетность сайта — как часто на него ссылаются с других сайтов, какой рейтинг у сайта в поисковых системах (появляется ли ссылка на сайт на первой странице с результатами поиска или на 31-й). Известные сайты обычно дорожат своим авторитетом.

При поиске информации, которая может устареть (например, расписания автобусов), постарайтесь, если это удастся, найти дату последнего обновления страницы.

Отметим, что алгоритмы определения рейтинга сайта в поисковых системах обычно содержатся в тайне. Можно только сказать, что рейтинг повышается, если сайт часто обновляется и на нём публикуются новые оригинальные материалы, которые не встречаются на других сайтах.

Электронная почта

Для того чтобы отправлять и принимать сообщения, пользователь должен зарегистрировать **почтовый ящик** на одном из **почтовых серверов** в Интернете.
Отправлять и принимать сообщения можно с помощью специальной **почтовой программы** или браузера (через **веб-интерфейс**).
Электронный адрес состоит из двух частей — названия почтового ящика и имени сервера; они разделяются символом @.

На рисунке 7.19 показано, как работает электронная почта. 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 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).

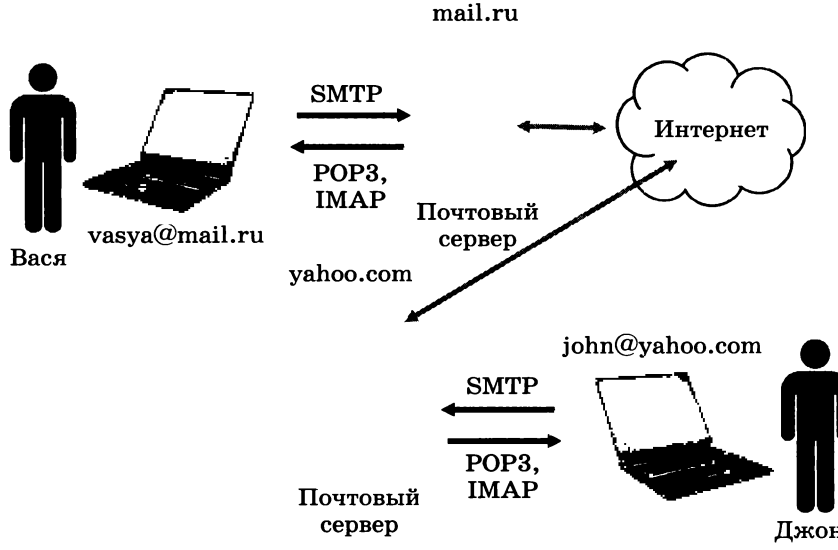


Рис. 7.19

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

Заголовок 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 дурным 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».

Основная часть письма строится по 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:

```
Здравствуй, Джон!
Нет ли у тебя желания купить слона?
С уважением, Василий Рогов,
генеральный директор ООО «Слонопотам»,
г. Слонов, ул. Потамная, 2
тел. +7 (1812) 111-22-33, факс +7 (1812) 111-22-34
http://slonopotam.ru
```

Вместе с письмом можно отправить любые небольшие файлы (ограничения на максимально допустимый размер файла устанавливаются администратором сервера). Для обмена файлами объёмом больше нескольких мегабайт можно использовать облачные хранилища — дисковую память на файловых серверах в Интернете

(например, **cloud.mail.ru** и **disk.yandex.ru**). Многие почтовые серверы запрещают пересылку исполняемых файлов (с расширением *exe*), потому что так могут распространяться вирусы и вредоносные программы.

Работать с электронной почтой можно прямо **в браузере** (такая возможность есть у большинства почтовых серверов) или с помощью специальных программ — **почтовых клиентов**. Если вы используете почтовую программу, ваша почта хранится на вашем компьютере и будет всегда доступна, даже когда нет связи с Интернетом.

Почтовые программы «умеют» создавать, отправлять и принимать сообщения, проверять почту через заданный интервал времени, раскладывать сообщения по папкам. Часто используемые адреса можно хранить в адресной книге.

В состав 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*. На компьютерах фирмы *Apple* устанавливается почтовый клиент *Apple Mail*.

Нередко пользователи имеют несколько адресов электронной почты, например один для рабочей переписки, второй — для личной. Почтовые серверы (например, **mail.ru**) обычно позволяют «собирать» все сообщения на какой-то один почтовый ящик, так чтобы можно было работать с ними без переключения на другую учётную запись.

Обмен файлами (FTP)

Для обмена файлами используется протокол **FTP** (англ. *File Transfer Protocol* — протокол передачи файлов), 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 (свободных, бесплатных и условно-бесплатных программ, 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 разрешают 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* — рис. 7.20. В одном окне программы показан каталог на компьютере пользователя, а в другом — каталог на FTP-сервере. Файлы можно перетаскивать между окнами с помощью мыши.

Рис. 7.20

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.

Рис. 7.21

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*.

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 — это специальные веб-сайты, предназначенные для общения посетителей в форме обмена сообщениями. Сообщения хранятся на серверах в Интернете и поэтому доступны всем участникам в любой момент.

Администратор 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 (рис. 7.22). 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 свое сообщение, обычно требуется регистрация. Могут быть и закрытые разделы, для доступа в которые кроме регистрации нужно специальное разрешения 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** — свою фотографию или любую другую картинку, которая его может характеризовать. Иногда используют также слово **«аватарка»** или **«юзерпик»** (от англ. *user picture* — картинка пользователя), имеющее то же значение, что и аватар.

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

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**).

Группы новостей сыграли большую роль в развитии информационных технологий. Достаточно сказать, что популярность операционной системы *Linux* началась с сообщения Линуса Торвальдса, отправленного в группу *comp.os.minix* 25 августа 1991 года: «... Я делаю (свободную) операционную систему... Эта система пишется с апреля и скоро будет готова...».

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 — обмен текстовыми сообщениями. **Чаты** (англ. *chat* — болтовня) позволяют «разговаривать» группе людей, которые как бы находятся в одной комнате. Для личного общения используют программы для мгновенного обмена сообщениями — **мессенджеры**, например: *Mail.ru Агент* (**www.mail.ru**), *QIP* (**qip.ru**), *Messages* (для компьютеров фирмы *Apple*). На мобильных устройствах популярны кроссплатформенные программы *WhatsApp* (**whatsapp.com**) *и* *Viber* (**www.viber.com**). 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, 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 переда-

ваемые 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 всех популярных операционных систем, в том числе и для мобильных устройств. 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.

Пиринговые сети

Большинство служб Интернета работает по технологии «клиент — сервер». Это значит, что программа-клиент, установленная на компьютере пользователя, посылает запрос серверу и принимает от него результат выполнения запроса. Но в Интернете существуют и так называемые одноранговые, или **пиринговые** (от англ. **P2P**: *peer-to-peer* — «равный к равному»), системы обмена данными, в которых каждый компьютер может попеременно выступать и как сервер (отдавать данные), и как клиент (получать данные). Такие виртуальные сети (работающие на основе Интернета) называют также *децентрализованными*, т. е. не имеющими единого центра.

Основное назначение пиринговых сетей — обмен файлами. Как вы знаете, для передачи по каналам связи файлы разбиваются на фрагменты (пакеты). При использовании схемы «клиент — сервер» все пакеты скачиваются с одного сервера, и скорость передачи зависит от нагрузки и скорости работы этого сервера. В пиринговой сети скачивание происходит по частям, одновременно со многих компьютеров (**пиров**), на которых обнаружен данный файл. Поэтому скорость скачивания зависит от количества найденных пиров и пропускной способности вашего канала связи.

В пиринговой сети существует одна проблема — найти компьютеры, на которых хранится нужный файл, и определить, какие из них сейчас активны и готовы к передаче данных. Чтобы облегчить решение этой задачи, используют **трекеры** (англ. *tracker*) — специальные серверы, на которых хранится информация о клиентах и контрольных суммах файлов. С помощью

трекеров клиенты устанавливают связь друг с другом. Таким образом, многие пиринговые сети всё же используют серверы для организации передачи данных.

Наиболее популярный протокол для организации пиринговых сетей в Интернете — **BitTorrent**. Его поддерживает большое количество программ-клиентов, в том числе **BitTorrent (www.bittorrent.com)** и **µTorrent (www.utorrent.com)**. Эти программы используют для загрузки так называемые **торрент-файлы** (файлы с расширением *torrent*), в которых записаны адрес трекера, сведения о файле, размеры и контрольные суммы каждой его части.

В крупных локальных сетях применяется протокол **Direct Connect**, который позволяет построить сеть на основе хабов (серверов). Программы-клиенты (например, *DC++*) связываются с одним или несколькими хабами и, получив от них данные, затем могут скачивать файлы напрямую с компьютеров других пользователей.

В пиринговых сетях принято соблюдать некоторые правила. Самое главное — нужно не только скачивать информацию, но и делиться скачанными файлами. Поэтому, получив файл, не нужно сразу «уходить с раздачи», т. е. завершать работу клиентской программы. Желательно, чтобы объём отданных вами данных был больше того, что вы скачали.

В пиринговых сетях можно найти и бесплатно скачать практически любую информацию, в том числе электронные книги, музыку, фильмы, программное обеспечение и т. п. К сожалению, часто такой обмен файлами нарушает авторские права и в этом случае может повлечь административную и уголовную ответственность.

Информационные системы

Информационная система — это аппаратные и программные средства, предназначенные для того, чтобы своевременно обеспечить пользователей нужной информацией.

Информационная система в Интернете 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 самолетов с учётом 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.

Многие пассажиры пользуются **электронными билетами** (англ. *e-ticket*) на поезда и самолёты. Электронный билет — это информация о заказе, сделанном через веб-сайт, которая внесена в базу данных. Распечатав электронный билет на поезд, достаточно затем предъявить его проводнику. 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 и даже найти ближайшие отели, кафе и рестораны. На этих сайтах доступны не только карты (хранящиеся в векторном формате), но также снимки многих районов из космоса и фотографии улиц крупных городов.

Практически во всех организациях и государственных органах документы хранятся в базах данных в цифровом виде. На портале **www.gosuslugi.ru** граждане могут подать в электронном виде заявление на **оказание различных государственных услуг**, например на оформление заграничного паспорта, представление налоговой декларации или регистрацию предприятия.

Широкие возможности предоставляют **мобильные устройства** — смартфоны и планшетные компьютеры. Геолокационные сервисы позволяют с помощью компьютерных сетей и спутниковой навигации (*GPS*, *ГЛОНАСС*) определить место, где находится пользователь устройства, и связать его с картой. С помощью бесплатных сервисов можно определить загруженность автомагистра́лей и время ожидания транспорта, построить маршрут в нужную точку, вызвать такси, найти ближайшие кафе и кинотеатры и т. п.

Выводы

- Чтобы сайт стал доступен другим компьютерам, на сервере должна быть запущена специальная программа — веб-сервер.
- Особенность современного Интернета — привлечение пользователей к наполнению сайтов информацией (технология *Web 2.0*).

- Для поиска в Интернете можно использовать каталоги ссылок и поисковые системы (поиск по ключевым словам).
- Для отправки и получения сообщений электронной почты необходимо зарегистрироваться и создать почтовый ящик на одном из почтовых серверов.
- FTP-серверы обеспечивают работу с файловыми архивами.
- Форумы — это специальные веб-сайты, предназначенные для общения посетителей в форме обмена сообщениями.
- Пиринговые сети — это одноранговые сети для обмена файлами.
- Информационная система — это аппаратные и программные средства, предназначенные для того, чтобы своевременно обеспечить пользователей нужной информацией.

Нарисуйте в тетради интеллект-карту этого параграфа.

Вопросы и задания

1. Как работает технология «клент — сервер» при просмотре веб-страниц?
2. В каких случаях используется протокол HTTPS? Чем он отличается от протокола HTTP?
3. Что такое *Web 2.0*? Расскажите о достоинствах и недостатках этой технологии.
4. Чем, на ваш взгляд, объясняется популярность блогов?
5. Что такое семантическая паутина? Каковы, на ваш взгляд, достоинства и недостатки этой идеи?
6. Сравните возможности каталогов и поисковых систем.
7. Что необходимо для отправки сообщений по электронной почте?
8. Какие протоколы используются для работы с электронной почтой?
9. Сравните работу с почтой через веб-интерфейс и с помощью почтовых программ.
10. Почему рекомендуется не оставлять поле «Тема» пустым?
11. Почему многие почтовые серверы не разрешают пересылку исполняемых файлов?
12. Как и в каком случае можно зайти на FTP-сервер, не имея своей учётной записи?
13. Как работать с FTP-серверами с помощью браузера?
14. Как узнать, что находится в файлах на FTP-сервере, не скачивая их? Всегда ли это возможно?

15. Как найти сервер, откуда можно загрузить интересующий вас файл?
16. Как вы думаете, зачем производители аппаратуры организуют форумы на своих сайтах?
17. Что такое электронный билет? Чем он удобнее бумажного?

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) «Блогер — хобби или профессия?»
е) «Сервисы Google»
ж) «Язык запросов поисковой системы Google»
з) «Язык запросов поисковой системы Яндекс»
и) «Вики-сайты»
к) «Протоколы POP3 и IMAP»
л) «Безопасность электронной почты»
м) «Почта Google»
н) «Служба FTP»
о) «Онлайн-общение в Интернете»
п) «Картографические сервисы Интернета»
р) «Поиск и заказ билетов в Интернете»
с) «Онлайн-переводчики»
т) «Онлайн-словари»

Проекты

а) Веб-сервер в локальной сети
б) FTP-сервер в локальной сети
в) Электронная почта в локальной сети
г) Форум в локальной сети
д) Чат в локальной сети

Интересные сайты

mail.ru — портал с бесплатной электронной почтой
mail.yandex.ru — бесплатная Яндекс-почта
ritlabs.com — почтовая программа *TheBat*
mozilla-russia.org — бесплатная почтовая программа *Mozilla Thunderbird*
filezilla-project.org — бесплатная кроссплатформенная программа *FileZilla* для работы с FTP-серверами

§ 49
Электронная коммерция

Ключевые слова:

- интернет-магазин
- интернет-аукцион
- электронная платёжная система
- код протекции

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 годов 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 году, 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 году был создан первый книжный интернет-магазин *Amazon* (**www.amazon.com**), который и сейчас остается 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 (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 серьёзные 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 определён, оформляется заказ: покупатель указывает свои данные, способ оплаты и доставки. На некоторых 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 существует 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 завершён, покупатель, сделавший наибольшую ставку, должен перевести деньги продавцу, а продавец обязан выслать товар покупателю по почте. Крупнейший интернет-аукцион — *eBay* (**www.ebay.com**).

При покупках в Интернете нужно соблюдать некоторую осторожность. В первую очередь обращайте внимание на то, чтобы на сайте магазина были указаны регистрационные данные и юридический адрес организации, контактные телефоны, условия доставки и возврата покупок. Сайт, сделанный непрофессионально, — серьёзный повод для отказа от покупки. Полезно заранее поискать в Интернете отзывы покупателей о выбранном магазине. Надёжнее всего обращаться в известные интернет-магазины (например, **ozon.ru**), которые дорожат своей репутацией.

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* и *ЯндексДеньги*. Самая популярная международная платёжная система — *PayPal*.

В платёжной системе можно завести электронный кошелёк, который пополняется с помощью специальных карт оплаты, через терминалы или с банковского счёта. 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 *ЯндексДеньги* 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 счёт. 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 на кошелек 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.

Выводы

- Электронная коммерция — это покупка и продажа товаров и услуг с помощью электронных систем, например через Интернет.

- 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.
- Интернет-магазин — это веб-сайт, который рекламирует товары или услуги, принимает заказы на покупку, предлагает варианты оплаты и получения заказа.
- Электронная платёжная система — это система расчётов с помощью электронных денег.

Нарисуйте в тетради интеллект-карту этого параграфа.

Вопросы и задания

1. Какие направления включает электронная коммерция?
2. Какие преимущества предоставляет электронная торговля для продавцов и покупателей?
3. Как можно оплатить покупку в интернет-магазине? Как доставляются покупки?
4. Сравните электронные деньги с «обычными» деньгами. В чём их достоинства и недостатки?
5. Зачем используется код протекции при переводе электронных денег?

Подготовьте сообщение

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

Проект

Сравнение электронных платёжных систем

Интересные сайты

money.yandex.ru — электронная платёжная система *Яндекс.Деньги*
webmoney.ru — электронная платёжная система *WebMoney*
paypal.com/ru/— электронная платёжная система *PayPal*

§ 50
Личное информационное пространство

Ключевые слова:

- облачные хранилища
- нетикет
- спам
- плагиат
- учётная запись
- пароль
- фишинг

Организация личных данных

Сейчас бóльшая часть **данных, необходимых человеку**, хранится в цифровом виде, на компьютерах. Такие данные образуют его личное информационное пространство. Для того чтобы человеку было легко находить, добавлять и изменять эти данные, нужно содержать их в порядке. Кроме того, нужно следить, чтобы они не были случайно потеряны и к ним не могли получить доступ посторонние.

Вы знаете, что данные в долговременной памяти хранятся в виде файлов. Имена файлов должны говорить о содержании этих файлов. Например, *Новый_файл.docx* — это неудачное имя файла, а *Схема_дирижабля.docx* — значительно лучше.

На компьютере обычно хранятся сотни ваших файлов, поэтому их нужно раскладывать в папки — объединять вместе файлы, которые связаны по какому-то признаку. Например, для фотографий можно создать папку *Фото*, в ней — вложенные папки для каждого года (или для каждого места, которое вы посетили).

К личному информационному пространству относится и **программное обеспечение, которое вы используете**, — операционная система, прикладные программы и т. д. Операционную систему можно настраивать «на свой вкус», так чтобы вам было удобно работать. Прежде всего вы можете выбрать рисунок рабочего стола, разместить на нём ярлыки только тех программ и документов, которые вы часто используете. Программы, которыми вы не пользуетесь, лучше удалить с компьютера, часто после этого его работа ускоряется.

Многие пользователи хранят свои данные в **облачных хранилищах** (*Google Диск*, *Яндекс.Диск* и др.). Конечно, это очень удобно, ведь доступ к ним можно получить с любого компьютера, имеющего доступ к Интернету. В то же время нужно помнить о безопасности этих данных: они хранятся на серверах в Интерне-

те и неизвестно, кто имеет к ним доступ. Поэтому размещать в облачных хранилищах секретные данные нельзя.

Для общения в сети Интернет вы можете создать свой **блог, сайт, форум или страницу в социальной сети.** Это тоже элементы вашего личного информационного пространства.

Регистрируясь на каком-то сайте, вы получаете **учётную запись (аккаунт**, от английского слова *account* — личный счёт), которая отличает вас от других пользователей. Она содержит имя пользователя (**логин**, от английского выражения *log in* — зарегистрировать) и **пароль.** Учётная запись — это ваш «личный кабинет» на этом сайте, логин — это адрес кабинета, а пароль — «ключ» к этому кабинету.

Организуя своё личное информационное пространство, необходимо соблюдать правила информационной безопасности: нельзя сообщать кому-либо ваши пароли и публиковать фотографии и сведения, которые могут вам как-то повредить. Об этом мы подробнее поговорим в главе 10 «Информационная безопасность».

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 правилами 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.

Нетикет запрещает оскорбления и переход на личности, клевету и распространение ложной информации, *плагиат* (присваивание себе авторства чужих материалов).

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 и частной жизни. Если вы хотите опубликовать личное сообщение, нужно спросить разрешения у автора. Строго говоря, если вы хотите выложить в сеть какие-то

фотографии, нужно убедиться, что все, кто на них изображён, не возражают против этого.

Электронные письма, как и бумажные, тоже пишут по правилам. Нужно всегда заполнять поле «Тема» в письме. Дело в том, что в списке сообщений мы в первую очередь видим автора письма и тему. Прочитав тему, можно сразу определить, о чём это сообщение и насколько срочно нужно на него реагировать. Кроме того, по теме можно легко найти нужное сообщение в архиве.

Основную часть письма нужно начинать с приветствия и заканчивать подписью, чтобы ваш адресат смог определить, от кого пришло письмо. Письма без понятной темы и подписи многие люди сразу удаляют.

На многие почтовые ящики часто приходят рекламные сообщения, которые вы не просили присылать. Такие письма называются **спамом** (от англ. *spam*). Часто они предлагают пройти какие-нибудь курсы или купить чудо-лекарства, которые лечат от всех болезней. Рассылка спама — это нарушение нетикета.

На форумах не разрешается умышленный отход от темы обсуждения (**оффтопик**, т. е. сообщение «вне темы»), реклама и самореклама. Иногда устанавливаются особые правила оформления сообщений: правила составления заголовков тем, ограничения на размер сообщения и личной подписи.

Старожилы форума иногда составляют для новичков список часто задаваемых вопросов с ответами (по английски — **FAQ**: *frequently asked questions*). Считается неприличным задавать вопросы, ответы на которые можно найти в этом документе. Поэтому прежде, чем задать вопрос на форуме, лучше некоторое время понаблюдать за его работой и понять, какие правила тут действуют.

Отправляя сообщение на форум, осознайте, что многие увидят ваш текст. Пишите кратко и точно, цените время других людей. Не используйте слэнг, пишите грамотно, не пропускайте пробелы и знаки препинания. Не разрешается набирать всё сообщение ПРОПИСНЫМИ БУКВАМИ, в Интернете это воспринимается как крик. Все эти правила относятся и к сообщениям, которые пересылаются по электронной почте.

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 серьёзно.

Цитируйте те высказывания, на которые отвечаете (собеседник может забыть содержание предыдущего письма).

Не участвуйте во **флейме** (от англ. *flame* — огонь, пламя) — так называется «спор ради спора», словесная война. Не разжигайте **холивары** (от англ. *holy war* — «священная война») — споры о двух идеях, каждая из которых имеет своих сторонников (например, что лучше: *Windows* или *Linux*, Паскаль или C++ и т. п.).

За соблюдением нетикета на форумах следят **модераторы**. В случае нарушения правил модератор может предупредить участника, а при повторном нарушении — запретить ему отправлять сообщения на форум или в чат (наложить **бан**, **забанить**). Бан может быть временный (на несколько дней, на месяц) или постоянный (навсегда).

Интернет и право

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 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 разреше-

ние на использование материала, отправив сообщение веб-мастеру (автору сайта) по электронной почте.

На своем веб-сайте **можно без разрешения** размещать:

- гиперссылки на другие сайты;
- бесплатную графику;
- произведения авторов, со дня смерти которых прошло более 70 лет;
- официальные документы.

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.

Неправомерный доступ к чужой информации («взлом» сайтов, почтовых ящиков, личных страничек) — это уголовное преступление, которое наказывается лишением свободы на срок до 5 лет (Уголовный кодекс РФ, статья 272). То же самое отно-

сится и к созданию, использованию и распространению вредоносных компьютерных программ (статья 273, до 7 лет лишения свободы).

Выводы

- Называйте файлы так, чтобы их имена говорили о содержимом.
- В облачных хранилищах нельзя хранить секретные данные.
- Данные, которые вы выкладываете в социальные сети, на форумы и в блоги, могут попасть в руки злоумышленников.
- Нетикет — это правила общения в Интернете.
- На форумах нарушением нетикета считается умышленный отход от темы, оскорбления, реклама и самореклама, спор ради спора.
- В электронных письмах нужно всегда заполнять поле «Тема», начинать письмо с приветствия и заканчивать подписью.
- Спам — это нежелательные рекламные сообщения.
- Плагиат — это использование чужого материала без ссылки.
- Неправомерный доступ к чужой информации («взлом» сайтов, почтовых ящиков, личных страничек) — это уголовное преступление.

Нарисуйте в тетради интеллект-карту этого параграфа.

Вопросы и задания

1. Какую роль выполняют модераторы на форумах?
2. В каком случае можно опубликовать на форуме личное сообщение?
3. Сравните значения слов «флейм» и «холивар».
4. Обсудите в классе, к чему может привести нарушение нетикета при переписке по электронной почте.
5. Почему при пересылке больших файлов нужно спрашивать согласия получателя?
6. Какие юридические проблемы возникают в связи с куплей-продажей услуг в Интернете?
7. В каком случае можно размещать на своем веб-сайте чужой материал?
8. В каком случае можно бесплатно использовать произведения без согласия автора? Какие ограничения нужно при этом соблюдать?
9. Используя дополнительные источники, узнайте о наиболее распространённых нарушениях авторских прав в Интернете. К каким последствиям они привели?

10. Почему взлом персональной странички в социальной сети — это преступление?

Подготовьте сообщение

а) «Сетевой этикет»
б) «Что такое спам?»
в) «Что такое троллинг?»
г) «Интернет и право»
д) «Авторские права в Интернете»

Проекты

а) Создание блога класса (группы)
б) Словарь интернет-жаргона

Интересные сайты

livejournal.com — Живой Журнал — для размещения блогов
blogger.com — сервис для размещения блогов
internet-law.ru — сайт юридической фирмы «Интернет и право»

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»

Практические работы к главе 7

Работа № 34 «Сравнение поисковых систем»
Работа № 35 «Тестирование сети»
Работа № 36 «Информационные системы в Интернете»
Работа № 37 «Работа с FTP-сервером»
Работа № 38 «Электронная коммерция»

Глава 8
АЛГОРИТМИЗАЦИЯ И ПРОГРАММИРОВАНИЕ

§ 51
Алгоритмы

Ключевые слова:

- этапы решения задач на компьютере
- алгоритм
- свойства алгоритма
- способы записи алгоритма
- анализ алгоритмов

Этапы решения задач на компьютере

В решении любой задачи с помощью компьютера можно выделить несколько этапов.

1. **Постановка задачи.** Нужно чётко определить, что в задаче дано (исходные данные) и что нужно найти (результаты).
2. **Формализация.** На этом этапе строится **модель**, которая включает только существенные данные, влияющие на решение задачи. Например, при моделировании полёта брошенного в воду камня нужно решить, учитываем ли мы притяжение Земли, сопротивление воздуха и влияние Луны. Затем модель записывается с помощью **формального языка**, например в виде математических формул или уравнений, связывающих исходные данные и результаты.

Далее мы можем использовать готовое программное обеспечение или составить свою собственную программу. Рассмотрим второй вариант.

3. **Построение алгоритма**, с помощью которого на основе модели по исходным данным вычисляется результат.
4. **Составление программы** на языке программирования.
5. **Тестирование и отладка программы.** Цель тестирования — найти как можно больше ошибок в программе. Для этого используют **тесты** — специально подобранные наборы исходных данных, для которых известен правильный результат.

Тестирование не может доказать, что программа правильная (для этого пришлось бы проверить все возможные варианты исходных данных), оно может только выявить ошибки. После того как ошибки найдены, необходимо исправить их — выполнить **отладку** программы.

6. **Выполнение расчётов**. В отличие от тестирования на этом этапе мы выполняем расчёты для тех исходных данных, для которых результаты неизвестны.
7. **Анализ результатов**. После завершения расчётов нужно проверить, не противоречат ли результаты теории и здравому смыслу. Например, время движения автомобиля не может быть отрицательным. Если такие противоречия обнаружены, нужно искать ошибку в модели, алгоритме или программе.

Ключевой этап в этой последовательности — построение алгоритма.

Алгоритм — это точное описание последовательности действий некоторого исполнителя.

Основные **свойства алгоритма**:

- **дискретность** — алгоритм состоит из отдельных команд, каждая из которых выполняется ограниченное (не бесконечное) время;
- **понятность** — алгоритм содержит только команды, входящие в систему команд исполнителя, для которого он предназначен;
- **определённость** — при каждом выполнении алгоритма с одними и теми же исходными данными должен быть получен один и тот же результат.

Алгоритмы можно записывать на естественном языке (русском, английском и др.), по шагам, в виде блок-схем или на языках программирования.

Анализ алгоритмов

Для того чтобы грамотно использовать исполнителей, нужно понимать алгоритмы, по которым они работают. Часто мы не создаём алгоритмы сами, а используем готовые. В таких случаях полезно выполнять анализ алгоритмов, например, выяснив:

- какие исходные данные нужно подать на вход алгоритма, чтобы получить нужный результат;
- какие результаты могут быть получены при заданных исходных данных.

Рассмотрим несколько примеров.

Задача 1. По каналу связи передаются трёхзначные целые числа. Для каждой пары таких чисел строится контрольная сумма, позволяющая обнаруживать ошибки при передаче:

а) записывается сумма старших разрядов передаваемых чисел; к ней справа дописывается сумма средних разрядов, а затем слева — сумма младших разрядов тех же чисел;

б) контрольная сумма — это три цифры полученного числа: число тысяч, число сотен и число десятков.

Пример: передаются числа 768 и 156. Поразрядные суммы — 8, 11, 14. После шага а) получаем число $N = 14811$, контрольная сумма — 481.

Найдите минимальное и максимальное значения, которые может принимать контрольная сумма.

Решение. Поскольку числа трёхзначные, все они находятся на отрезке [100; 999]. Очевидно, что сумма значений двух любых десятичных цифр не меньше 0 и не больше 18. Минимальное число (N), которое может быть получено после шага *а*), равно 20 (при обработке чисел 100 и 100), а максимальное — 181818 (для чисел 999 и 999). Отсюда сразу следует, что минимальное значение контрольной суммы — 2.

Определим максимальное значение контрольной суммы. Допустим, что контрольная сумма равна 999. Это значит, что число N имеет вид «..999.», где точка обозначает десятичную цифру, возможно — 0. Сумма значений двух цифр не может быть больше 18, поэтому есть только один вариант разбиения такого числа на поразрядные суммы: $9|9|9x$, где x — какая-то цифра. Но этот вариант невозможен — сумма средних разрядов не может быть больше 18, поэтому максимальное значение контрольной суммы — 991. Такая контрольная сумма получается, например, для чисел 259 и 760.

Задача 2. Для условия задачи 1 приведите примеры данных, при которых контрольная сумма равна 106.

Решение. Если контрольная сумма равна 106, то число N может быть записано в виде «..106.». Возникает вопрос — как разбить это число на 3 поразрядные суммы?

Поскольку вторая справа цифра — 6, а двузначная сумма значений двух цифр не может начинаться с 6, есть только один вариант: $10|6|x$, где x — неизвестная однозначная сумма средних разрядов чисел. Поэтому нужно найти два трёхзначных числа, у которых сумма старших разрядов равна 6, сумма младших разрядов равна 10, а сумма средних разрядов меньше 10. Это могут быть, например, пары 209 и 491 или 138 и 522.

Мы увидели, что для нашей контрольной суммы может возникнуть *коллизия* — ситуация, когда при разных данных контрольная сумма одинакова. Коллизии возникают практически всегда, ведь у нас есть всего 990 теоретически возможных значения контрольной суммы (от 2 до 991), на которые «отображаются» $900 \cdot 900$ вариантов пар передаваемых чисел (напомним, что все числа — трёхзначные, на отрезке [100, 999]).

Задача 3. Для условия задачи 1 определите, сколько существует пар чисел, для которых контрольная сумма равна 421.

Решение. Если контрольная сумма равна 421, то число N может быть записано в виде «..421.». Сумма старших разрядов не может быть равна ни 42, ни 21, поэтому такое число может быть разбито на отдельные суммы только так: «.4|2|1.». Это значит, что сумма старших разрядов равна 2, сумма младших разрядов равна 4 или 14, а сумма средних может принимать любые значения от 10 до 18.

Подсчитаем, сколько есть возможных вариантов составления каждой из возможных сумм. Сумму 2 в старших разрядах можно получить только как $1+1$. Для суммы 4 имеем 5 вариантов: $0+4$, $1+3$, $2+2$, $3+1$ и $4+0$. Для суммы 10 минимальное из двух слагаемых равно 1, получается всего 9 вариантов: от $1+9$ до $9+1$. Рассуждая так же, для суммы 14 получаем 5 вариантов (от $5+9$ до $9+5$), а для суммы 18 — один вариант ($9+9$). Таким образом, существует $5+5=10$ вариантов различных пар младших разрядов и $9+8+7+6+5+4+3+2+1=45$ различных подходящих пар средних разрядов. Общее количество подходящих пар равно $1 \cdot 45 \cdot 10 = 450$.

Задача 4. Для условия задачи 1 приведите примеры значений, которые контрольная сумма принимать не может.

Решение. Рассмотрим сначала случай, когда сумма средних разрядов — однозначная (меньше 10). Тогда число N может быть разбито на отдельные суммы двумя способами: «..*ab*|*c*|.» или «..*a*|*bc*|.», где a, b и c — некоторые цифры. В первом случае ab — это любое число от 0 до 18 (сумма младших разрядов), c (сумма старших разрядов) может быть от 2 до 9. Поэтому правильные контрольные суммы — это числа 2...9, 12...19, ..., 182...189.

Во втором случае bc — любое число от 10 до 18, a — любое число от 0 до 9. Под такой шаблон подходят числа 10...18, 110...118, ..., 910...918.

Рассмотрим теперь вариант, когда сумма средних разрядов — двухзначная (от 10 до 18). Тогда последняя цифра контрольной суммы — это 1, и число N может быть разбито на отдельные

суммы двумя способами: «..$a|b|$1.» или «..$|ab|$1.», где a и b — некоторые цифры. В первом случае a — это любая цифра от 0 до 9, b (сумма старших разрядов) — любая цифра от 2 до 9; правильные контрольные суммы заканчиваются на 1 — это 21, 31, ..., 991.

Во втором случае ab — любое число от 10 до 18 (сумма старших разрядов), правильные контрольные суммы этого типа — 101, 111, ..., 181.

Все числа, которые не подходят ни под один шаблон (а их достаточно много!), не могут быть правильными контрольными суммами. Например, это числа 20, 100, 292 и т. д.

Задача 5. В некоторой информационной системе данные кодируются двоичными шестиразрядными словами. При передаче данных возможны искажения, поэтому в конец каждого слова добавляется седьмой (контрольный) разряд — бит чётности — таким образом, чтобы сумма разрядов нового слова, считая контрольный, была чётной. Например, к слову 110011 справа будет добавлен 0, а к слову 101100 — 1.

После приёма слова производится его обработка. При этом проверяется сумма его разрядов, включая контрольный. Если она нечётна, это означает, что при передаче этого слова произошёл сбой, и оно автоматически заменяется на зарезервированное слово 0000000. Если она чётна, это означает, что сбоя не было или сбоев было больше одного. В этом случае принятое слово не изменяется.

Исходное сообщение

1100101 1001011 0011000

было принято в виде

1100111 1001110 0011000.

Как будет выглядеть принятое сообщение после обработки?

Решение. Сообщение содержит три семибитных блока, по условию в каждом правильно принятом блоке число единиц должно быть чётное. В первом принятом блоке 1100111 — нечётное число единиц (5), поэтому он будет заменён на нули 000000. В остальных двух блоках чётное число единиц, поэтому принимающая сторона считает их правильными и изменять не будет. Заметим, что второй принятый блок не совпадает с переданным — он отличается в двух битах, но приёмник это не обнаруживает.

Ответ: 000000 1001110 0011000.

Выводы

- Этапы решения задач на компьютере: постановка задачи, формализация, построение алгоритма, составление программы, тестирование и отладка программы, выполнение расчётов, анализ результатов.
- Ключевой этап в решении задачи с помощью компьютера — составление алгоритма.
- Алгоритмы можно записывать на естественных языках, по шагам, в виде блок-схемы и на языке программирования.
- Анализ алгоритмов состоит в том, чтобы определить:
  — какие исходные данные нужно подать на вход алгоритма, чтобы получить нужный результат;
  — какие результаты могут быть получены при заданных исходных данных.

Нарисуйте в тетради интеллект-карту этого параграфа.

Вопросы и задания

1. Автомат получает на вход два трёхзначных числа. По этим числам строится новое число по следующим правилам[1)].
   1) Вычисляются три числа — сумма старших разрядов заданных трёхзначных чисел, сумма средних разрядов, сумма младших разрядов.
   2) Полученные три числа записываются друг за другом в порядке невозрастания (без разделителей).

   Какое наименьшее и наибольшее значения может иметь одно из исходных чисел, если:
   а) второе число равно 694, а в результате работы автомата получено число 11108;
   б) второе число равно 486, а в результате работы автомата получено число 13107?

2. Автомат получает на вход два трёхзначных числа. По этим числам строится новое число по следующим правилам.
   1) Вычисляются три числа — сумма старших разрядов заданных трёхзначных чисел, сумма средних разрядов, сумма младших разрядов.
   2) Полученные три числа записываются друг за другом в порядке неубывания (без разделителей).

   Какое наименьшее и наибольшее значения может иметь одно из исходных чисел, если:

1) Задачи 1–2 предложены Н. Е. Леко.

а) второе число равно 857, а в результате работы автомата получено число 81416;
б) второе число равно 714, а в результате работы автомата получено число 91012?

3. В некоторой информационной системе данные кодируются двоичными шестиразрядными словами с битом чётности. После приёма все слова, в которых обнаружено нечётное число единичных битов, автоматически заменяются на зарезервированное слово 0000000. Какое сообщение получилось после обработки принятых данных:
а) 1010010 1101001 1010101; б) 1110111 1101111 1010011;
в) 1110010 1101011 1010111; г) 1110100 1101101 0010011.

§ 52
Оптимальные линейные программы

Ключевые слова:

- оптимальная программа
- дерево вариантов

Задача 1. В системе команд исполнителя Удвоитель всего две команды:

1. `прибавь 1`
2. `умножь на 2`

Программа для исполнителя Удвоитель — это последовательность номеров команд. Составьте самую короткую программу для Удвоителя, которая преобразует число 6 в число 28.

Решение. Возможно, в этой простой задаче вы сразу сообразили, какой будет эта самая короткая программа. Но в более сложных случаях это не так просто. Кроме того, нам нужно доказать, что это действительно самая короткая программа. Поэтому рассмотрим общий способ решения таких задач.

Итак, нас интересует программа, содержащая меньше всего команд. Самую короткую программу можно назвать *оптимальной по длине*.

Оптимальная программа — это лучшая программа по какому-то показателю.

За одну команду мы можем из начального числа 6 получить только число 7 (командой 1) или число 12 (командой 2) — рис. 8.1.

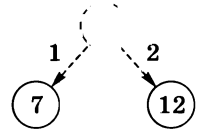


Рис. 8.1

Проверим все возможные двухшаговые и трёхшаговые программы. Мы обнаружим, что ни одна из возможных двухшаговых программ не приводит к нужному нам значению 28, но одна из трёхшаговых программ, 122 (она выделена голубыми стрелками на рис. 8.2), позволяет решить задачу.

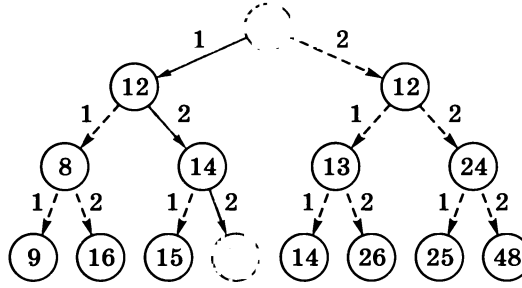


Рис. 8.2

Поскольку мы перебрали все возможные варианты, можно сделать вывод, что программа 122 из трёх команд — оптимальная, т. е. самая короткая.

Как видно из рис. 8.2, нет других программ из трёх команд, которые приводят к числу 28, поэтому оптимальная программа единственная. Все другие программы, с помощью которых можно получить число 28 из 6, длиннее, чем эта.

Схема, которую мы построили, называется **деревом возможных вариантов**. Действительно, мы рассмотрели **все возможные** программы, состоящие из одного, двух и трёх шагов.

Построение полного дерева вариантов при большой длине программы (например, 5 и больше команд) — это очень утомительная работа. Во многих задачах бывает проще двигаться не от начального числа к результату, а наоборот.

Итак, возьмём конечное число 28 и подумаем, какой последней командой мы могли его получить. Так как 28 делится на 2, это могла быть как команда 1 (из 27 получаем 28), так и

команда 2 (из 14 также получаем 28). Ни одно из этих чисел не совпадает с начальным (6), поэтому одной командой задачу не решить.

Делаем второй шаг — проверяем, из каких чисел мы могли получить 27 и 14. Число 27 не делится на 2, поэтому оно могло быть получено только командой 1 (из 26), а число 14 можно получить из 13 или из 7 (рис. 8.3).

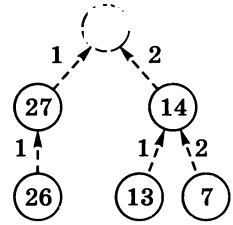


Рис. 8.3

До начального числа 6 снова добраться не удалось, поэтому решений из двух команд не существует. Делаем третий шаг, и на одной из ветвей получаем начальное число 6 (рис. 8.4).

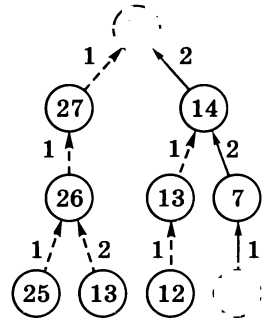


Рис. 8.4

Для того чтобы получить самую короткую программу, нужно пройти по стрелкам от начального числа к конечному, выписывая встречающиеся номера команд. Для нашей задачи получаем тот же ответ, что и раньше — 122 (путь выделен голубыми стрелками).

Обратите внимание, что дерево при движении «в обратном направлении» — от конечного числа к начальному, получилось меньше. Нам удалось найти решение быстрее, рассматривая меньше вариантов. Это связано с необратимостью одной из команд: любое число можно умножить на 2, но не любое целое число делится на 2 без остатка.

Выводы

- Оптимальная программа — это лучшая программа по какому-то показателю, например содержащая меньше всего команд.
- Для того чтобы найти оптимальную программу для исполнителя, можно сначала рассмотреть все возможные результаты его работы за один шаг, затем — за два шага и т. д., пока на каком-то шаге не будет получен желаемый результат. Первая найденная программа для перехода из начального состояния в конечное будет оптимальной по длине (самой короткой).
- Схема, показывающая все возможные результаты работы исполнителя, называется деревом возможных вариантов.
- Если в списке команд исполнителя есть необратимые команды, лучше строить дерево возможных вариантов от конечного состояния к начальному.

Нарисуйте в тетради интеллект-карту этого параграфа.

Вопросы и задания

1. Может ли быть так, что задача для Удвоителя решается с помощью нескольких различных программ одинаковой длины? Если да, то приведите примеры.
2. Как можно доказать, что построенная программа для Удвоителя действительно самая короткая?
3. Какие числа можно и какие нельзя получить из натурального числа N с помощью Удвоителя? Из нуля? Из отрицательного числа?
4. Как быстро построить самую короткую программу для получения некоторого числа N из нуля с помощью Удвоителя? Когда эта задача не имеет решений?
5. Исполнитель Калькулятор работает с целыми числами и умеет выполнять команды

   1. `прибавь 1`
   2. `раздели на 2`

   Команда 2 может применяться только к чётным числам. Нужно составить самую короткую программу для Калькулятора, с помощью которой из числа a можно получить число b. Как лучше перебирать варианты программ, от начального числа к конечному или наоборот? Почему?

§ 53
Анализ алгоритмов с ветвлениями и циклами

Ключевые слова:

- цикл
- ветвление
- исполнитель Робот
- исполнитель Чертёжник
- исполнитель Редактор

Цикл — это многократное выполнение одинаковых действий. Цикл состоит из заголовка и тела цикла — тех команд, которые находятся внутри цикла и выполняются несколько раз. **Ветвление** — это выбор одного из двух вариантов действий в зависимости от выполнения некоторого условия.

Если хотят, чтобы алгоритм был понятен наибольшему количеству людей, часто записывают его на алгоритмическом языке. **Алгоритмический язык** — это формальный язык, в нём есть все основные алгоритмические конструкции (ветвления, циклы), которые нужны для однозначной записи алгоритмов. Мы будем использовать алгоритмический язык с русскими служебными словами.

Исполнитель Робот

Система команд исполнителя Робот, «живущего» в прямоугольном лабиринте на клетчатой плоскости:

```
вверх
вниз
влево
вправо
```

При выполнении любой из этих команд Робот перемещается на одну клетку в соответствующем направлении. Четыре логические команды проверяют, есть ли стена у каждой стороны той клетки, где находится Робот:

```
сверху свободно        снизу свободно
слева свободно         справа свободно
```

Цикл `ПОКА <условие> КОНЕЦ` выполняется, пока условие истинно, после этого происходит переход на строку, следующую за циклом.

Задача 1. Сколько клеток лабиринта (рис. 8.5) соответствуют требованию, что, выполнив программу

```
ПОКА <снизу свободно> вниз КОНЕЦ
ПОКА <слева свободно> влево КОНЕЦ
ПОКА <сверху свободно> вверх КОНЕЦ
ПОКА <справа свободно> вправо КОНЕЦ
```

Робот остановится в той же клетке, с которой он начал движение?

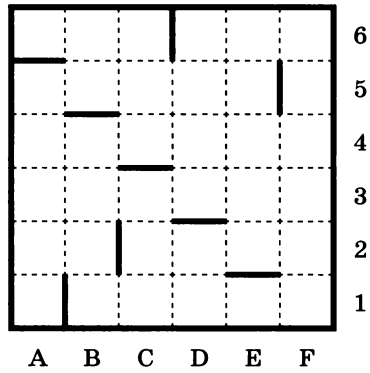


Рис. 8.5

Решение. Заметим, что цикл

```
ПОКА <снизу свободно> вниз КОНЕЦ
```

заставляет робота дойти до стены, расположенной снизу от него. А следующие три строки в теле цикла «доводят» робота до стены слева, сверху и справа.

Для того чтобы исполнитель вернулся обратно в ту клетку, откуда он начал движение, четыре «опорные» стенки, от которых он «отталкивается» при поворотах, должны быть расположены так, как на рис. 8.6.

Кроме того, необходимо, чтобы коридор, выделенный фоном на рис. 8.6, был свободен для прохода, в остальных местах стенки могут быть расположены, как угодно. Точка на рис. 8.6 обозначает клетку, начав движение с которой, Робот вернётся обратно. Обратите внимание, что возможны ещё «вырожденные» варианты, один из которых показан на рис. 8.7.

Рис. 8.6

Рис. 8.7

Так как начальная и конечная точки совпадают и последний цикл завершается у правой стенки, нужно рассматривать лишь те клетки-кандидаты, где есть стенка справа. Отметим их (рис. 8.8).

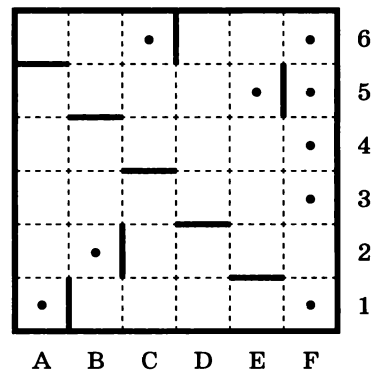


Рис. 8.8

Предположим, что Робот начинает движение с любой клетки на вертикали F. Тогда после выполнения первого цикла он окажется в клетке F1, после второго — в клетке B1, после третьего — в B4 и, завершив выполнение всей программы, остановится в клетке F4. Таким образом. одну подходяшую клетку мы

нашли — это F4, а остальные клетки вертикали F условию не удовлетворяют (рис. 8.9).

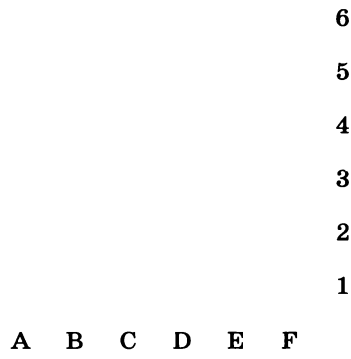


Рис. 8.9

Проверяем оставшиеся четыре клетки-кандидата: для каждой из них обнаруживаем, что после выполнения алгоритма Робот не приходит в ту клетку, откуда он стартовал. Поэтому условию удовлетворяет только одна клетка — F4.

Ответ: 1.

Исполнитель Чертёжник

Исполнитель Чертёжник перемещается на координатной плоскости, оставляя след в виде линии. Чертёжник может выполнять команду

```
сместиться на (a, b)
```

где a, b — целые числа. Эта команда перемещает Чертёжника из точки с координатами $(x; y)$ в точку с координатами $(x + a; y + b)$. Цикл

```
ПОВТОРИ число РАЗ
  последовательность команд
КОНЕЦ
```

означает, что последовательность команд будет выполнена указанное число раз (число должно быть натуральным).

Задача 2. Чертёжнику был дан для исполнения следующий алгоритм (буквами n, a, b обозначены неизвестные числа):

```
сместиться на (-2, -11)
ПОВТОРИ n РАЗ
```

```
    сместиться на (a, b)
    сместиться на (27, 12)
КОНЕЦ
сместиться на (-22, -7)
```

Укажите все возможные значения n, для которых найдутся такие значения чисел a и b, что после выполнения программы Чертёжник возвратится в исходную точку.

Решение. Запишем общее изменение координат Чертёжника в результате выполнения этого алгоритма:

$$\begin{cases} \Delta x = -2 + n(a + 27) - 22 = n(a + 27) - 24; \\ \Delta y = -11 + n(b + 12) - 7 = n(b + 12) - 18. \end{cases}$$

Поскольку Чертёжник должен вернуться в исходную точку, эти величины должны быть равны нулю; следовательно, нужно найти все натуральные значения n, при которых система уравнений

$$\begin{cases} n(a + 27) - 24 = 0; \\ n(b + 12) - 18 = 0 \end{cases} \Rightarrow \begin{cases} n(a + 27) = 24; \\ n(b + 12) = 18 \end{cases}$$

разрешима в целых числах относительно a и b. Из этих уравнений следует, что число n должно быть делителем числа 24 и (одновременно) делителем числа 18.

Второй сомножитель в каждом уравнении всегда можно составить, выбрав соответствующие значения a и b. Например, если взять $n = 1$, то $a = 24 - 27 = -3$ и $b = 18 - 12 = 6$. Таким образом, для решения задачи достаточно найти все числа, на которые одновременно делятся 24 и 18. Вот все такие делители: 1, 2, 3, 6.

Исполнитель Редактор

Редактор получает на вход строку символов и преобразовывает её. Редактор может выполнять две команды, в обеих командах v и w обозначают цепочки символов. Команда

```
нашлось (v)
```

проверяет, встречается ли цепочка v в строке. Если эта цепочка есть в строке, то команда возвращает логическое значение «истина», в противном случае возвращает значение «ложь». Строка при этом не изменяется. Команда

```
заменить (v, w)
```

заменяет в строке первое слева вхождение цепочки v на цепочку w.

Задача 3. Дана программа для исполнителя Редактор:

```
ПОКА нашлось (222) ИЛИ нашлось (888)
  ЕСЛИ нашлось (222)
    ТО заменить (222, 8)
    ИНАЧЕ заменить (888, 2)
  КОНЕЦ
КОНЕЦ
```

Какая строка получится в результате применения приведённой ниже программы к строке, состоящей из 68 идущих подряд цифр 8?

Решение. Из программы видим, что Редактор что-то делает только тогда, когда в строке есть цепочка 222 или цепочка 888. Если ни одной из этих цепочек нет, программа заканчивает работу.

Если в строке есть 222, то в первую очередь именно эта цепочка меняется (на 8). Если в строке нет цепочки 222, но есть 888, то цепочка 888 меняется на 2.

Попробуем выполнить первые шаги алгоритма для цепочки из 68 цифр 8. Сначала первая цепочка 888 заменяется на 2, получается

2 [65 цифр 8]

Дальше так же меняем следующие две группы из трёх восьмёрок:

222 [59 цифр 8]

Теперь у нас появилась цепочка 222, поэтому в соответствии с алгоритмом она сразу будет заменена на 8, получаем

[60 цифр 8]

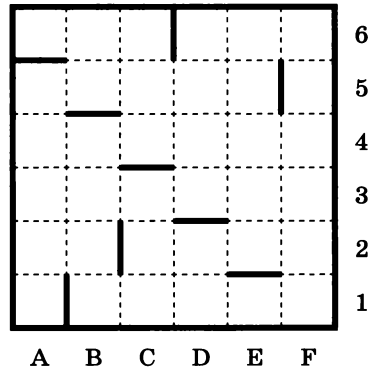
Таким образом, за первые 4 шага работы цикла мы заменили 9 восьмерок на одну восьмёрку или, что то же самое, удалили 8 восьмерок. Очевидно, что следующие 4 шага удалят ещё 8 восьмерок и т. д. Мы сможем повторить эти 4 шага 8 раз, после этого останется $68 - 8 \cdot 8 = 4$ восьмерки. На последнем шаге в цепочке 8888 заменяем 888 на 2 и получаем 28.

Ответ: 28.

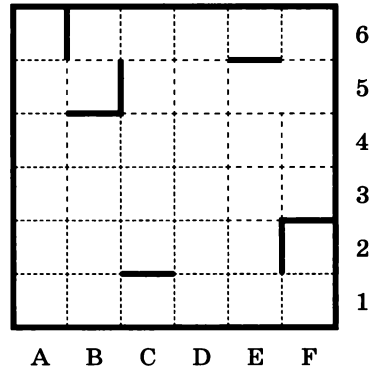
Вопросы и задания

1. Сколько клеток приведённого лабиринта соответствуют требованию: выполнив предложенную ниже программу, Робот остановится в той же клетке, с которой он начал движение?

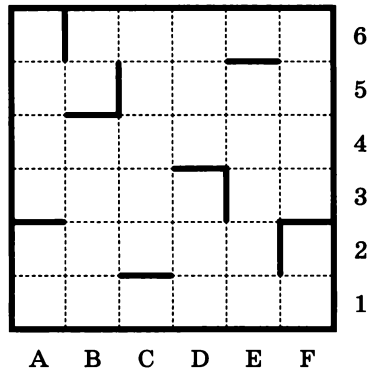
а)
```
ПОКА <справа свободно> вправо КОНЕЦ
ПОКА <сверху свободно> вверх КОНЕЦ
ПОКА <слева свободно> влево КОНЕЦ
ПОКА <снизу свободно> вниз КОНЕЦ
```



б)
```
ПОКА <сверху свободно> вправо КОНЕЦ
ПОКА <справа свободно> вниз КОНЕЦ
ПОКА <снизу свободно> влево КОНЕЦ
ПОКА <слева свободно> вверх КОНЕЦ
```



в)
```
ПОКА <справа свободно> вниз КОНЕЦ
ПОКА <снизу свободно> влево КОНЕЦ
ПОКА <слева свободно> вверх КОНЕЦ
ПОКА <сверху свободно> вправо КОНЕЦ
```



2. Дана программа для исполнителя Редактор:

```
ПОКА нашлось (2222) ИЛИ нашлось (8888)
  ЕСЛИ нашлось (2222)
    ТО заменить (2222, 8)
    ИНАЧЕ заменить (8888, 2)
  КОНЕЦ
КОНЕЦ
```

Какая строка получится в результате применения этой программы к строке, состоящей из:

а) 62 идущих подряд цифр 8; б) 65 идущих подряд цифр 8;
в) 72 идущих подряд цифр 8; г) 93 идущих подряд цифр 8;
д) 146 идущих подряд цифр 8; е) 156 идущих подряд цифр 8;
ж) 184 идущих подряд цифр 8; з) 193 идущих подряд цифр 8?

3. Дана программа для исполнителя Редактор:

```
ПОКА нашлось (333) ИЛИ нашлось (555)
  ЕСЛИ нашлось (555)
    ТО заменить (555, 3)
    ИНАЧЕ заменить (333, 5)
  КОНЕЦ
КОНЕЦ
```

Какая строка получится в результате применения приведённой ниже программы к строке, состоящей из:

а) 62 идущих подряд цифр 5; б) 65 идущих подряд цифр 5;
в) 72 идущих подряд цифр 5; г) 93 идущих подряд цифр 5;
д) 146 идущих подряд цифр 5; е) 156 идущих подряд цифр 5;
ж) 184 идущих подряд цифр 5; з) 193 идущих подряд цифр 5?

§ 54
Введение в язык Python

Ключевые слова:

- скрипт
- комментарий
- переменная
- тип переменной
- оператор присваивания
- арифметическое выражение

В основной школе вы изучали основы программирования, используя один из учебных языков. Скорее всего, это был школьный алгоритмический язык или язык Паскаль. Теперь мы познакомимся ещё с одним языком, который называется **Python** (по-русски — Питон или Пайтон). Язык Python — это профессиональный язык программирования, который активно используется в таких компаниях, как *Яндекс* и *Google*. На Python разрабатываются сайты и веб-сервисы, он применяется для составления **скриптов** (от англ. *script* — *сценарий*) — небольших программ, расширяющих возможности других программ, таких как *GIMP*, *Blender*, многие игры.

Python — современный и развивающийся язык, и мы надеемся, что при его изучении вы не только будете получать удовольствие от программирования, но и освоите важный инструмент, который сможете использовать на практике после окончания школы.

Простейшая программа

Программы на языке Python чаще всего выполняются **интерпретатором**, который читает очередную команду и сразу её выполняет, не переводя всю программу в машинный код конкретного процессора. Можно работать в двух режимах:

- через командную строку (в *интерактивном* режиме), когда каждая введённая команда сразу выполняется;
- в программном режиме, когда программа сначала записывается в файл (обычно имеющий расширение *py*), и при запуске выполняется целиком; такая программа на Python называется **скриптом**.

Мы будем говорить, главным образом, о программном режиме.

Пустая программа — это программа, которая ничего не делает, но удовлетворяет требованиям выбранного языка программирования. Пустая программа на Python (в отличие от многих других языков программирования) — действительно пустая, она не содержит ни одного **оператора** (команды). Можно добавить в

программу **комментарий** — пояснение, которое не обрабатывается интерпретатором:

```
# Это пустая программа
```

Как вы увидели, комментарий начинается знаком #. Если в программе используются русские буквы, в самом начале обычно записывают специальный комментарий, определяющий кодировку:

```
# -*- coding: utf-8 -*-
```

В данном случае указана кодировка UTF-8. В Python версии 3 она установлена по умолчанию, поэтому эту строку можно не писать.

Первое, что нужно научиться делать, — выводить сообщения на экран консоли — специального окна для текстового ввода и вывода. Это делает команда `print`:

```
# -*- coding: utf-8 -*-
print("Привет!")
```

Символы, которые нужно вывести, заключаются в кавычки (как в языке C) или в апострофы (как в Паскале). После выполнения этой команды происходит автоматический переход на новую строку на экране (в языке Паскаль это соответствует вызову процедуры `Writeln`).

Переменные

Переменная — это величина, значение которой можно изменять во время работы алгоритма.

Переменная имеет имя (идентификатор[1]), тип и значение. Для изменения значения переменной используется оператор присваивания.

Напишем программу, которая выполняет сложение двух чисел:

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

Числа, введённые пользователем, нужно записать в переменные. В языке Python (в отличие от многих других языков) переменные не нужно заранее объявлять. Они создаются в памяти при первом использовании, точнее, при первом присваивании им значения. Например, при выполнении оператора присваивания

```
a = 4
```

[1] От слова «идентифицировать» — отличить один объект от другого.

в памяти создаётся новая переменная (объект типа «целое число»), и она связывается с именем *a*.

В **именах переменных** можно использовать латинские буквы[1] (строчные и прописные буквы различаются), цифры (но имя не может начинаться с цифры, иначе транслятору будет сложно различить, где начинается имя, а где — число) и знак подчёркивания «_».

Желательно давать переменным «говорящие» имена, чтобы можно было сразу понять, зачем нужна та или иная переменная.

Тип переменной в Python определяется автоматически. Программа

```
a = 4
print(type(a))
```

выдаёт на экран тип (англ. *type*) переменной a:

```
<class 'int'>
```

В данном случае a — это переменная целого типа, на это указывает слово int (сокращение от англ. *integer* — целый). Говорят, что переменная a относится к классу int.

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 по-разному, см. главу 4).

В языке Python используется **динамическая типизация**, это значит, что тип переменной определяется по значению, которое ей присваивается (а не при объявлении переменной, как во многих языках). Таким образом, в разных частях программы одна и та же переменная может хранить значения разных типов.

Обсудите в классе, какие преимущества и недостатки имеет этот подход.

Вспомним, что нам нужно решить три подзадачи:

- ввести два числа с клавиатуры и записать их в переменные (назовём их a и b);
- сложить эти два числа и записать результат в третью переменную c;
- вывести значение переменной *c* на экран.

---

[1] Можно использовать и русские буквы, но лучше так не делать.

Для ввода используется функция input, результат работы которой можно записать в переменную, например так:

```
a = input()
```

При выполнении этой строки система будет ожидать ввода с клавиатуры и, когда пользователь введёт число и нажмёт клавишу *Enter*, система запишет введённое значение в переменную a. При вызове функции input в скобках можно записать сообщение-подсказку:

```
a = input("Введите целое число: ")
```

Сложить два значения и записать результат в переменную c очень просто:

```
c = a + b
```

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

```
i = i + 1
```

увеличивает значение переменной i на 1.

В языке Python возможно множественное присваивание, при котором значения записываются сразу в несколько переменных, например так:

```
a, b = 1, 34
c, d, b18 = 3, 100, 500
```

Важно, чтобы переменных слева от знака «=» было столько же, сколько значений справа от него.

Выведём значение переменной c на экран с помощью уже знакомой функции print:

```
print(c)
```

Казалось бы, теперь легко собрать всю программу:

```
a = input()
b = input()
c = a + b
print(c)
```

Однако после того, как мы запустим её и введём какие-то числа, допустим 21 и 33, мы увидим странный ответ: 2133. Вспомните, что при нажатии клавиши на клавиатуре в компьютер поступает её код, т. е. код соответствующего символа. И входные данные воспринимаются функцией input именно как поток символов. Поэтому в той программе, которая приведена выше, переменные a и b — это цепочки символов, при сложении этих цепочек (с помощью оператора «+») программа просто объединяет их — приписывает вторую цепочку в конец первой.

Чтобы исправить эту ошибку, нужно преобразовать символьную строку, которая получена при вводе, в целое число. Это делается с помощью функции `int` (от англ. *integer* — целый):

```
a = int(input())
b = int(input())
```

Возможен ещё один вариант: оба числа вводятся не в разных строках, а в одной строке через пробел. В этом случае ввод выполняется сложнее:

```
a, b = map(int, input().split())
```

В этой строке собрано сразу несколько достаточно сложных операций:

`input()` возвращает строку, которая введена с клавиатуры;

`input().split()` — эта строка разрезается на части по пробелам; в результате получается набор значений (список);

`map()` применяет какую-то операцию ко всем элементам списка; в нашем случае это функция `int`, которая превращает каждый элемент списка в целое число.

В результате после работы функции `map` мы получаем новый список, состоящий уже из чисел. Первое введённое число (первый элемент списка) записывается в переменную `a`, а второе — в переменную `b`. То же самое можно сделать за два шага, не используя функцию `map`:

```
a, b = input().split()
a, b = int(a), int(b)
```

Таким же способом можно прочитать и большее количество значений переменных. Важно только, чтобы было введено ровно столько чисел, сколько переменных указано слева от знака присваивания.

Итак, после того как мы преобразовали введённые значения в формат целых чисел, программа работает правильно — складывает два числа, введённые с клавиатуры. Однако у неё 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:

```
print(a, "+", b, "=", c)
```

Мы почти получили нужный результат, но почему-то знаки «+» и «=» отделены лишними пробелами, который мы «не заказывали»:

```
2 + 3 = 5
```

Действительно, функция `print` вставляет между выводимыми значениями так называемый **разделитель (сепаратор**, англ. *separator*). По умолчанию разделитель — это пробел, но мы можем его изменить, указав новый разделитель после слова `sep`:

```
print(a, "+", b, "=", c, sep = "")
```

Здесь мы установили пустой разделитель (пустую строку). Теперь все работает как надо, лишних пробелов нет.

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 его результат:

```
print(a, "+", b, "=", a+b, sep = "")
```

Для того чтобы после выполнения функции `print` программа не переходила на новую строку, можно задать пустую строку в дополнительном параметре `end`:

```
print("Привет, ", end = "")
print("Вася!")
```

В Python можно использовать **форматный вывод**:

```
print("{:d}+{:d}={:d}".format(a, b, a+b))
```

Строка-формат содержит специальные обозначения в фигурных скобках, например, `{:d}` — это место для вывода целого значения (от англ. *decimal* — десятичный). После слова `format` в скобках перечисляются сами значения, которые нужно вывести (числа, имена переменных или арифметические выражения). Первое значение выводится вместо первой отметки `{:d}`, второе — вместо второй и т. д.

Для каждого значения можно указать общее количество знакомест, отводимое на число. Например, программа

```
n = 123
print("Найдено сайтов:{:5d}".format(n))
```

выведет значение целой переменной a, заняв ровно 5 знакомест:

```
Найдено сайтов:··123
```

Поскольку само число занимает только 3 знакоместа, перед ним выводятся два пробела, которые здесь обозначены как «·». Если число не помещается в отведённое место, оно выводится полностью (количество знакомест увеличивается до длины числа).

Типы данных

Перечислим основные простые типы данных в языке Python:

- int — целые значения;
- float — вещественные значения (могут быть с дробной частью);
- bool — логические значения, True (истина, «да») или False (ложь, «нет»);
- str — символ или символьная строка, т. е. цепочка символов.

Тип переменной определяется в тот момент, когда ей присваивается новое значение. Мы уже видели, что для определения типа переменной можно использовать стандартную функцию type. Например, программа

```
a = 5
print(type(a))
a = 4.5
print(type(a))
a = True
print(type(a))
a = "Вася"
print(type(a))
```

выдаст на экран такой результат:

```
<class 'int'>
<class 'float'>
<class 'bool'>
<class 'str'>
```

Сначала в переменной a хранится целое значение 5, и её тип — целый (int). Затем мы записываем в неё вещественное значение 4,5, переменная будет вещественного типа (float, от англ. *floating point* — с плавающей точкой). Третье присваивание записывает в переменную a логическое значение (bool, от англ. *boolean* — булевская величина, в честь Дж. Буля). Последнее значение — символьная строка (str, от англ. *string* — строка), которая записывается в апострофах или в кавычках.

Целые переменные в Python могут быть сколь угодно большими (или, наоборот, маленькими, если речь идёт об отрицательных числах). Транслятор автоматически выделяет область памяти такого размера, который необходим для сохранения результата вычислений, единственное ограничение связано с объёмом доступной памяти. Поэтому в Python легко (в отличие от других языков программирования) точно выполнять вычисления с «длинными» целыми числами, которые содержат много значащих цифр.

Вещественные переменные, как правило, занимают 8 байт, что обеспечивает точность 15 значащих десятичных цифр. Большинство вещественных чисел хранится в памяти неточно, и в результате операций с ними накапливается вычислительная ошибка (см. главу 4).

Как размещаются переменные в памяти

Как вы увидели, одна и та же переменная в различных частях программы может хранить значения различных типов. Дело в том, что в Python имя переменной связывается с некоторым объектом, этот объект имеет определённый тип и содержит данные. При выполнении оператора

```
a = 5
```

в памяти создаётся объект — ячейка для хранения целого числа, которая связывается с именем a (рис. 8.10, *а*). Затем команда

```
a = 4.5
```

создаёт в памяти другой объект (для хранения вещественного числа) и переставляет ссылку для имени *a* (рис. 8.10, *б*). Объект, на который не ссылается ни одно имя, удаляется из памяти «сборщиком мусора».

Если выполнить оператор

```
b = a
```

то два имени будут связаны с одним и тем же объектом (рис. 8.10, *в*). Можно было бы подумать, что изменение одной из переменных приведёт к такому же изменению другой. Однако *для чисел* это не так. Дело в том, что при присваивании

```
a = 10
```

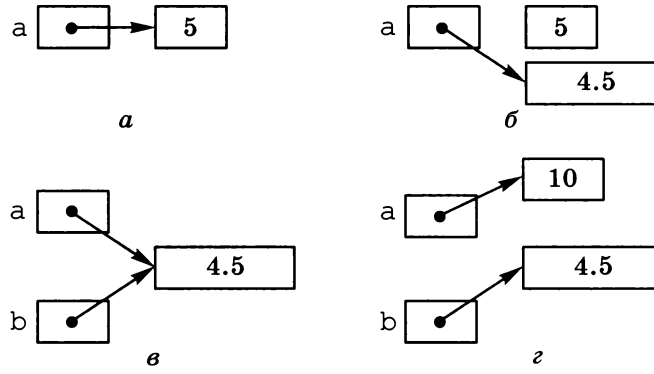


Рис. 8.10

в памяти будет создан новый объект — целое число 10, который связывается с именем a (рис. 8.10, *г*). Это отличается от других языков программирования, где в таких случаях просто изменяется значение той же самой ячейки памяти.

Арифметические выражения и операции

Арифметические выражения в любом языке программирования записываются в строку, без многоэтажных дробей. Они могут содержать числа, имена переменных, знаки арифметических операций, круглые скобки (для изменения порядка действий) и вызовы функций. Например:

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

Если запись длинного выражения не поместилась в одной строке на экране, её можно перенести на следующую с помощью знака «\» (он называется «обратный слэш»):

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

При переносе внутри скобок знак «\» вставлять не обязательно:

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

Эти правила переноса справедливы и для других операторов языка Python.

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

1) действия в скобках;
2) возведение в степень (**), справа налево;
3) умножение (*) и деление (/), слева направо;
4) сложение и вычитание, слева направо.

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

В Python (как и в языке C) разрешено каскадное присваивание. Запись

```
a = b = 0
```

равносильна паре операторов

```
b = 0
a = b
```

Так же как и в языке C, часто используют сокращённую запись арифметических операций.

Сокращённая запись:	*Полная запись*:
`a += b`	`a = a + b`
`a -= b`	`a = a - b`
`a *= b`	`a = a * b`
`a /= b`	`a = a / b`

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. Нужно помнить, что результат деления (операции «/») — это вещественное число, даже если делимое и делитель — целые и делятся друг на друга нацело[1)].

Выводы

- Программы (скрипты) на языке Python выполняются интерпретатором.
- Переменные в языке Python не нужно объявлять. Любой переменной в разных частях программы можно присваивать значения разных типов.

---

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

- При вводе числовых данных нужно преобразовывать их из символьного формата в числовой.
- Простые типы данных в Python — целые, вещественные, логические и символьные.

Нарисуйте в тетради интеллект-карту этого параграфа.

Вопросы и задания

1. Зачем нужен тип переменной?
2. Почему желательно выводить на экран подсказку перед вводом данных?
3. Когда, по вашему мнению, можно вычислять результат прямо в операторе вывода, а когда нужно заводить отдельную переменную?
4. Сравните запись арифметических операций в Python и в том языке программирования, который вы изучали в основной школе.
5. В каком порядке выполняются операции, если они имеют одинаковый приоритет?
6. Зачем используются скобки?
7. Что происходит, если в выражения входят переменные разных числовых типов? Какого типа будет результат?

Темы сообщений

а) «История языка Python»
б) «Философия языка Python (Zen of Python)»
в) «Применение языка Python»
г) «Скрипты на языке Python в других программах»
д) «Язык Python на мобильных устройствах»

Проекты

а) Сравнение языков Python и Паскаль
б) Сравнение сред программирования на языке Python

Интересные сайты

python.org — сайт разработчиков языка Python
wingware.com/downloads/wingide-101 — бесплатная среда для разработки программ на языке Python

pythonfiddle.com — онлайн-среда для программирования на языке Python

codeskulptor.org — онлайн-среда для программирования на языке Python 2 с использованием графического интерфейса

pythontutor.com — здесь можно посмотреть, как выполняется программа на языке Python

pythontutor.ru — онлайн-учебник по языку Python

§ 55
Вычисления

Ключевые слова:

- деление нацело
- остаток от деления
- модуль
- импорт модуля
- случайное число
- документация на программу

Деление нацело и остаток

Во многих практических задачах все данные — целые числа. Для них введены две особые операции: **деление нацело** и **остаток от деления**, которые обозначаются как «**//**» и «**%**» соответственно. Они имеют такой же приоритет, как умножение и деление.

```
d = 85
a = d // 10    # = 8
b = d % 10     # = 5
```

Обратим внимание на результат выполнения этих операций для отрицательных чисел. Программа

```
print(-7 // 2)
print(-7 % 2)
```

выдаст на экран числа −4 и 1. Дело в том, что, с точки зрения теории чисел, остаток — это неотрицательное число, поэтому $-7 = (-4) \times 2 + 1$, т. е. частное от деления (−7) на 2 равно −4, а остаток равен 1. В Python (в отличие от многих других языков, например Паскаля и C) эти операции выполняются математически правильно.

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

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

Возведение в степень имеет более высокий приоритет, чем умножение и деление.

Вещественные значения

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

```
x = 123.456
```

Вещественные значения по умолчанию выводятся на экран с большим количеством значащих цифр, например:

```
x = 1/3
print (x)           #  0.3333333333333333
```

При выводе очень больших или очень маленьких чисел используется **научный (экспоненциальный)** формат. Например, программа

```
x = 100000000000000000/3
print ( x )
```

выведет

```
3.3333333333333332e+16
```

что означает $3.3333333333333332 \cdot 10^{16}$, т. е. до буквы `e` указывают значащую часть числа, а после неё — порядок (см. главу 4).

Часто используют **форматный вывод**: все данные, которые нужно вывести, сначала преобразуют в символьную строку с помощью функции `format`:

```
a = 1/3
print( "{:7.3f}".format(a) )
```

В данном случае использован формат `7.3f`, который определяет вывод числа с фиксированной запятой (*f* от англ. *fixed* — фиксированный) в 7 позициях с тремя знаками в дробной части:

```
◦◦0.333
```

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

Можно использовать форматный вывод для нескольких значений сразу (список этих значений заключается в круглые скобки):

```
a = 1/3
b = 1/9
print("{:7.3f} {:7.3f}".format(a, b))#◦◦0.333◦◦◦0.111
```

Запись `%e` обозначает экспоненциальный формат:

```
print("{:10.3e} {:10.3e}".format(a, b))
```

Здесь числа 10 и 3 — это общее количество позиций и число знаков после десятичной точки в записи значащей части:

```
◦3.333e-01◦◦1.111e-01
```

Стандартные функции

Библиотека языка Python содержит большое количество готовых функций, которые можно вызывать из программы. Некоторые функции встроены в ядро языка, например, для вычисления модуля числа используется функция `abs`:

```
print(abs(-1.2))  # 1.2
```

Существуют встроенные функции для перехода от вещественных значений к целым:

`int(x)` — приведение вещественного числа `x` к целому, отбрасывание дробной части;

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

Функции `bin`, `oct` и `hex` переводят число соответственно в двоичную, восьмеричную и шестнадцатеричную системы счисления:

```
s2 = bin(29)  # '0b11101'
s8 = oct(29)  # '0o35'
s16 = hex(29) # '0x1d'
```

Большинство стандартных функций языка Python разбиты на группы по назначению, и каждая группа записана в отдельный файл, который называется **модулем**. Математические функции собраны в модуле `math`:

`sqrt(x)` — квадратный корень числа `x`;
`sin(x)` — синус угла `x`, заданного в радианах;
`cos(x)` — косинус угла `x`, заданного в радианах;
`exp(x)` — экспонента числа `x`;
`log(x)` — натуральный логарифм числа `x`.

В этом же модуле введена и константа `pi`, равная числу π.

Для подключения модуля используется команда **импорта (загрузки) модуля**:

```
import math
```

После этого для обращения к функциям используется так называемая **точечная запись**: указывают имя модуля и затем через точку название функции:

```
print(math.sqrt(x))
```

Можно поступить по-другому: загрузить в рабочее пространство все функции модуля:

```
from math import *
```

Теперь к функциям модуля `math` можно обращаться так же, как к встроенным функциям:

```
print(sqrt(x))
```

Этот способ обладает серьёзным недостатком: в рабочем пространстве появляется много дополнительных имён, которые могут совпасть с именами функций, объявленных в других модулях. Поэтому без острой необходимости лучше так не делать.

Третий вариант — загрузить только нужные функции:

```
from math import sqrt, sin, cos
```

В этом случае все функции модуля `math`, кроме перечисленных (`sqrt`, `sin`, `cos`), будут недоступны.

Документирование программы

К выпуску программы разработчик должен подготовить **документацию на программу**. Руководство пользователя (это наиболее важная часть документации) обычно содержит:

- назначение программы;
- формат входных данных;
- формат выходных данных;
- примеры использования программы.

Для примера составим документацию на простую программу, которая находит корни квадратного уравнения по известным вам формулам:

```
from math import sqrt
print("Введите коэффициенты квадратного уравнения:")
a, b, c = map(float, input().split())
D = b*b - 4*a*c
x1 = (-b + sqrt(D))/(2*a)
x2 = (-b - sqrt(D))/(2*a)
print("x1={:5.3f} x2={:5.3f}".format(x1, x2))
```

Назначение программы: вычисление корней квадратного уравнения $ax^2 + bx + c = 0$.

Формат входных данных: значения коэффициентов a, b и c вводятся с клавиатуры через пробел в одной строке.

Формат выходных данных: значения корней уравнения выводятся на экран через пробел в одной строке; перед значением первого корня выводится текст `x1=`, перед значением второго корня — текст `x2=`.

Пример использования программы (решение уравнения $2x^2 - 6x + 3 = 0$):

```
Введите коэффициенты квадратного уравнения:
2 -6 3
x1=2.366 x2=0.634
```

Случайные числа

В некоторых задачах необходимо моделировать случайные явления, например результат бросания игрального кубика (на нём может выпасть число от 1 до 6). Как сделать это на компьютере, который по определению «не случаен», 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 таких 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.

Функции для работы с псевдослучайными числами собраны в модуле `random`. Для получения псевдослучайных чисел в заданном диапазоне используют функции:

`random()` — случайное вещественное число из полуинтервала [0, 1);

`randint(a,b)` — случайное целое число из отрезка [a, b].

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

```
from random import randint
n = randint(1,6)
```

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

```
from random import random
x = 7*random() + 5
```

или с помощью функции `uniform` из модуля `random`:

```
from random import uniform
x = uniform(5, 12)
```

Выводы

- Деление нацело и вычисление остатка от деления выполняются с помощью операторов «//» и «%».
- Часть стандартных функций Python входит в ядро языка, а остальные разбиты на группы по назначению и записаны в отдельные файлы — модули.
- Для подключения модуля к программе нужно использовать команду `import`.
- Математические функции объединены в модуль `math`, а функции для работы со случайными числами — в модуль `random`.

Нарисуйте в тетради интеллект-карту этого параграфа.

Вопросы и задания

1. Опишите операции `//` и `%`.
2. Расскажите о проблеме вычисления остатка от деления в различных языках программирования. Обсудите в классе этот вопрос.
3. Выясните, в каких единицах задаётся аргумент тригонометрических функций.

[1] В библиотеке Python используется один из наиболее совершенных алгоритмов для генерации псевдослучайных чисел — «вихрь Мерсенна», разработанный в 1997 году.

4. Как выполнить округление вещественного числа до ближайшего целого?
5. Какие числа называют случайными? Зачем они нужны?
6. Почему «естественные» случайные числа почти не используются в цифровой технике?
7. Чем отличаются псевдослучайные числа от случайных?
8. Как получить случайное целое число в диапазоне от 50 до 100? Случайное вещественное число в том же диапазоне?

Темы сообщений

а) «Библиотеки языка Python»
б) «Вычисление значений математических функции (sin, cos и др.)»
в) «Случайные и псевдослучайные числа»
г) «Линейный конгруэнтный метод»

Проект

Сравнение датчиков псевдослучайных чисел

§ 56
Ветвления

Ключевые слова:

- условный оператор
- отступы
- вложенный условный оператор
- сложные условия
- порядок выполнения операций

Условный оператор предназначен для выбора из двух вариантов действий. После служебного слова **if** (**если**) записывается некоторое условие; если оно истинно, то выполняется блок операторов, расположенный далее. Вторая (необязательная) часть условного оператора начинается словом **else** (**иначе**), сразу после него записывается блок операторов, которые выполняются, если условие после слова **if** ложно.

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 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:

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

Если **истинно** (верно) условие, записанное после ключевого слова **if**, то затем выполняются все команды (**блок команд**), которые расположены до слова **else**. Если же условие после **if** **ложно** (неверно), выполняются команды, стоящие после **else**.

Обратите внимание, что после условия и после слова **else** ставятся двоеточия.

В Python, в отличие от других языков, важную роль играют сдвиги операторов относительно левой границы (отступы). Слова **if** и **else** начинаются на одном уровне, а все команды внутренних блоков сдвинуты относительно этого уровня вправо на одно и то же расстояние. Это позволяет не использовать особые ограничители блоков (слова **begin** и **end** в языке Паскаль, фигурные скобки в С-подобных языках). Для сдвига используют символы табуляции (которые вставляются при нажатии на клавишу *Tab*) или пробелы.

Если в блоке всего один оператор, иногда бывает удобно записать его в той же строке, что и ключевое слово **if** (**else**):

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

В приведённых примерах 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
if b > a:
  M = b
```

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

Поскольку операция выбора максимального из двух значений нужна очень часто, в Python есть встроенная функция `max`[1], которая вызывается так:

```
M = max(a, b)
```

Если выбирается максимальное из двух чисел, можно использовать особую форму условного оператора в Python:

```
M = a if a > b else b
```

которая работает так же, как и приведённый выше условный оператор в полной форме: записывает в переменную `M` значение `a`, если выполняется условие `a > b`, и значение `b`, если это условие ложно.

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`:

```
if a > b:
  c = a
  a = b
  b = c
```

Все операторы, входящие в блок, сдвинуты на одинаковое расстояние от левого края. Заметим, что в Python, в отличие от многих других языков программирования, есть множественное присваивание, которое позволяет выполнить эту операцию значительно проще:

```
a, b = b, a
```

Кроме знаков < и > в условиях можно использовать другие знаки отношений: <= (меньше или равно), >= (больше или равно), == (равно — два знака «=» без пробела, чтобы отличить от операции присваивания) и != (не равно).

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 воз-

[1] Есть также и аналогичная функция `min`, которая выбирает минимальное из двух или нескольких значений.

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

```
if a > b:
  print("Андрей старше")
else:
  if a == b:
    print("Одного возраста")
  else:
    print("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. Если после **else** сразу следует ещё один оператор **if**, можно использовать так называемое **«каскадное» ветвление** с ключевыми словами **elif** (сокращение от **else if**): если очередное условие ложно, выполняется проверка следующего условия и т. д.

```
if a > b:
  print("Андрей старше")
elif a == b:
  print("Одного возраста")
else:
  print("Борис старше")
```

Обратите внимание на отступы: слова **if**, **elif** и **else** находятся на одном уровне!

Если в цепочке **if-elif-elif-...** выполняется несколько условий, то срабатывает первое из них. Например, программа

```
if cost < 1000:
  print("Скидок нет.")
elif cost < 2000:
  print("Скидка 2%.")
elif cost < 5000:
  print("Скидка 5%.")
else:
  print("Скидка 10%.")
```

при `cost = 1500` выдаёт `Скидка 2%.`, хотя условие `cost < 5000` тоже выполняется.

Сложные условия

Предположим, что ООО «Слонопотам» набирает сотрудников, возраст которых от 25 до 40 лет включительно. Нужно написать программу, которая запрашивает возраст претендента и выдаёт 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). В языке Python они записываются английскими словами **and**, **or** и **not**.

Пусть в переменной v записан возраст сотрудника. Тогда нужный фрагмент программы будет выглядеть так:

```
if v >= 25 and v <= 40:
  print("Подходит")
else:
  print("Не подходит")
```

При вычислении сложного логического выражения действия выполняются в следующем порядке:

1) отношения (<, <=, >, >=, ==, !=),
2) операции **not**,
3) операции **and**,
4) операции **or**.

Одинаковые операции выполняются слева направо. Для изменения порядка действий используют круглые скобки.

Иногда условия получаются достаточно длинными, и их хочется перенести на следующую строку. Сделать это в Python можно двумя способами: использовать обратный слэш (это не рекомендуется):

```
if v >= 25 \
       and v <= 40:
```

или взять всё условие в скобки (перенос внутри скобок разрешён):

```
if (v >= 25
       and v <= 40):
```

В языке Python разрешены двойные неравенства, например

```
if A < B < C:
```

означает то же самое, что и

```
if A < B and B < C:
```

Выводы

- Условный оператор служит для выбора из двух вариантов.
- Слова **if** и **else** должны иметь одинаковый отступ.
- Все операторы, которые выполняются при выборе одного из вариантов (как в блоке **if**, так и в блоке **else**), записываются с одинаковым дополнительным отступом.
- Если число вариантов выбора больше двух, нужно использовать несколько условных операторов.
- Операции НЕ, И и ИЛИ в Python записываются как **not**, **and** и **or**.
- При обработке сложного условия сначала выполняются отношения, затем — операции **not**, после них — операции **and** и в конце — операции **or**.

Нарисуйте в тетради интеллект-карту этого параграфа.

Вопросы и задания

1. Чем отличаются разветвляющиеся алгоритмы от линейных?
2. Как вы думаете, почему не все задачи можно решить с помощью линейных алгоритмов?
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. Как организовать выбор из нескольких вариантов?
8. Как определяется порядок вычислений в сложном условии?

Темы сообщений

а) «Условные операторы в языке C»
б) «Условные операторы в языке Javascript»

Проекты

а) Экспертная система на выбранную тему
б) Игра «Угадай число»

§ 57
Циклические алгоритмы

Ключевые слова:

- цикл
- тело цикла
- счётчик
- цикл с условием
- трассировка
- алгоритм Евклида
- цикл с постусловием

Как организовать цикл?

Цикл — это многократное выполнение одинаковых действий. **Любой алгоритм может быть записан с помощью трёх алгоритмических конструкций**: последовательного выполнения команд (линейных участков алгоритмов), условных операторов и циклов.

Для того чтобы организовать цикл с заданным числом повторений, необходимо использовать ячейку памяти, в которой будет запоминаться количество выполненных шагов цикла (счётчик шагов). Можно сначала записать в неё ноль (ни одного шага не сделано), а после каждого шага цикла увеличивать значение счётчика на единицу. На псевдокоде (смеси естественного языка и какого-нибудь языка программирования) алгоритм можно записать так (здесь и далее операции, входящие в **тело цикла**, выделяются отступами):

```
счётчик = 0
пока счётчик != 10
  print("Привет!")
  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
  print("Привет!")
  уменьшить счётчик на 1
```

Этот вариант несколько лучше, чем предыдущий, поскольку счётчик сравнивается с нулём, а такое сравнение выполняется в процессоре автоматически (см. главу 4).

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 переменной n целого типа.

Сначала нужно разработать алгоритм решения задачи. 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 одной цифры.

Для отсечения первой слева 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:

n	Счётчик
1234	0
123	1
12	2
1	3
0	4

Запишем алгоритм на псевдокоде:

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

Соответствующая программа на Python выглядит так:

```
count = 0
while n > 0:
  n = n // 10
  count += 1
```

Слово **while** переводится как «пока», т. е. цикл выполняется, пока `n > 0`. После условия, как и в конце заголовка условного оператора **if**, ставится двоеточие. Переменная-счётчик имеет имя **count**.

Обратите внимание, что проверка условия выполняется в начале очередного шага 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. Кроме того, этот же цикл может быть использован и в том случае, когда число шагов известно заранее или может быть вычислено:

```
k = 0
while k < 10:
  print("Привет")
  k += 1
```

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 предыдущем цикле, программа **зациклится**:

```
k = 0
while k < 10:
  print("Привет")
```

Поиск максимальной цифры

Рассмотрим ещё один пример использования цикла с условием. Найдём максимальную цифру введённого натурального числа `n`. Вы уже научились считать цифры числа, используя операции деления, остаётся немного доработать эту программу.

Максимальную цифру будем хранить в переменной `M`. Сначала запишем в неё любое отрицательное число, тогда любая цифра будет больше, чем это начальное значение. Далее вспомним, что остаток от деления числа на 10 — это последняя цифра в его десятичной записи. Если значение этой цифры больше, чем `M`, записываем его в `M`. Когда мы таким образом переберём все цифры числа, в переменной `M` окажется максимальная цифра введённого числа `n`. Программа на языке Python запишется в виде цикла, внутри которого стоит условный оператор:

```
n = int(input())
M = -1
while n > 0:
  d = n % 10      # (1)
  if d > M:       # (2)
    M = d         # (3)
  n = n // 10
print(M)
```

В строке (1) выделяется последняя цифра числа и записывается в переменную d. В строках (2) и (3) мы изменяем значение M, если только что полученная цифра оказалась больше, чем значение M. Таким образом, в M всегда будет находиться значение максимальной из уже просмотренных цифр исходного числа.

Для проверки работы программы можно использовать **трассировку** («ручную прокрутку»). Для введённого числа 142 запишем в виде таблицы изменение значений всех переменных после выполнения каждого оператора программы (табл. 8.1).

Таблица 8.1

Оператор	Условие верно?	n	d	M
n = int(input())		142		
M = -1				-1
n > 0?	да			
d = n % 10			2	
d > M?	да			
M = d				2
n = n // 10		14		
n > 0?	да			
d = n % 10			4	
d > M?	да			
M = d				4
n = n // 10		1		
n > 0?	да			
d = n % 10			1	
d > M?	нет			
n = n // 10		0		
n > 0?	нет			

Обратите внимание, что на последнем шаге цикла значение переменной M не изменилось, потому что цифра 1 не больше, чем значение M = 4. Как только значение n стало равно 0, условие работы цикла (n > 0) нарушилось и его выполнение закончилось.

Алгоритм Евклида

Древнегреческий математик Евклид, живший в III веке до нашей эры, придумал замечательный алгоритм для вычисления наибольшего общего делителя (НОД) двух натуральных чисел. Этот алгоритм и сейчас используется, например, в задачах шифрования. Напомним, что НОД — это наибольшее число, на которое два заданных числа делятся без остатка.

Алгоритм Евклида для натуральных чисел: заменять большее из двух заданных чисел на их разность до тех пор, пока числа не станут равны. Полученное число и есть их НОД.

Алгоритм Евклида можно записать на языке Python так:

```
while a != b:
  if a > b:
    a = a - b
  else:
    b = b - a
print (a)
```

Этот вариант алгоритма работает довольно медленно, если одно из чисел значительно меньше другого, например для пары чисел 2 и 2014. Значительно быстрее выполняется улучшенный (модифицированный) алгоритм Евклида.

Модифицированный алгоритм Евклида для натуральных чисел: заменять большее из двух заданных чисел на остаток от деления большего на меньшее, пока этот остаток не станет равным нулю. Тогда второе число и есть их НОД.

Фрагмент программы на языке Python запишется так:

```
while a != 0 and b != 0:
  if a > b:
    a = a % b
  else:
    b = b % a
print(a + b)
```

Почему в конце выводится на экран сумма a + b? Дело в том, что цикл остановится тогда, когда одно из чисел будет равно нулю, тогда второе и есть их НОД. Но так как первое число равно нулю, сумма a + b равна второму числу.

Возможна и ещё более короткая запись:

```
while b:
  a, b = b, a % b
print(a)
```

Здесь запись **while** b означает то же самое, что и **while** b != 0. На каждом шаге цикла значения переменных меняются местами так, чтобы в переменной a оказалось большее из двух чисел. Цикл заканчивается, когда после очередного шага значение переменной b станет равно нулю, тогда в переменной a остаётся последний ненулевой остаток, это и есть НОД исходных чисел.

Циклы с постусловием

Во многих языках программирования существует **цикл с постусловием**, в котором условие проверяется *после* завершения очередного шага цикла. Это полезно в том случае, когда нужно обязательно *выполнить цикл хотя бы один раз*. Например, пользователь должен ввести с клавиатуры положительное число, и нужно защитить программу от неверных входных данных.

В языке Python нет цикла с постусловием, но его можно организовать с помощью цикла **while**:

```
print("Введите положительное число:")
n = int (input())
while n <= 0:
  print("Введите положительное число:")
  n = int (input())
```

Однако такой вариант не очень хорош, потому что нам пришлось написать два раза пару операторов. Но можно поступить иначе:

```
while True:
  print("Введите положительное число:")
  n = int (input())
  if n > 0: break
```

Цикл, который начинается с заголовка **while** True, будет выполняться бесконечно, потому что условие True всегда истинно. Выйти из такого цикла можно только с помощью специального

оператора **break** (в переводе с англ. — прервать; досрочный выход из цикла). В данном случае он сработает тогда, когда станет истинным условие `n > 0`, т. е. тогда, когда пользователь введёт допустимое значение.

Выводы

- Для организации цикла с заданным числом повторений нужно использовать переменную-счётчик.
- Цикл **while** выполняется до тех пор, пока условие в заголовке цикла остаётся истинным. Если этого не произойдёт, программа зациклится.
- Алгоритм Евклида для натуральных чисел: заменять большее из двух заданных чисел на их разность до тех пор, пока числа не станут равны. Полученное число и есть их НОД.
- Цикл с постусловием можно организовать с помощью оператора **break** (досрочный выход из цикла).

Нарисуйте в тетради интеллект-карту этого параграфа.

Вопросы и задания

1. Как будет работать приведённая в параграфе программа, которая считает количество цифр числа, при вводе отрицательного числа? Если вы считаете, что она работает неправильно, укажите, как её нужно доработать.
2. Что означает выражение «цикл с предусловием»?
3. В каком случае цикл с предусловием не выполняется ни разу?
4. В каком случае программа, содержащая цикл с условием, может зациклиться?

Подготовьте сообщение

а) «Циклы с условием в языке C»
б) «Циклы с условием в языке Javascript»

Проекты

а) Сравнение алгоритмов поиска НОД двух чисел
б) Программа для разложения числа на простые сомножители

§ 58
Циклы по переменной

Ключевые слова:

- переменная цикла
- диапазон
- шаг изменения переменной
- вложенный цикл

Что такое цикл по переменной?

Вернёмся снова к задаче, которую мы обсуждали в предыдущем параграфе, — вывести на экран 10 раз слово «Привет!». Фактически нам нужно организовать цикл, в котором блок операторов выполнится заданное число раз (в некоторых языках такой цикл есть, например, в школьном алгоритмическом языке он называется «цикл *N* раз»). На языке Python подобный цикл записывается так:

```
for i in range(10):
  print("Привет!")
```

Здесь слово **for** означает «для», переменная `i` (её называют **переменной цикла**) изменяется в диапазоне (**in** `range`) от 0 до 10, *не включая* 10 (т. е. от 0 до 9 включительно). Таким образом, цикл выполняется ровно 10 раз. Заметьте, что в конце заголовка цикла ставится двоеточие.

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
while  k <= 10 :
  print(2**k)
  k += 1
```

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). Цикл с переменной «собирает» все действия с ней в один оператор:

```
for k in range(1,11):
  print(2**k)
```

Здесь диапазон (`range`) задаётся двумя числами — начальным и конечным значениями, причём указанное конечное значение *не входит* в диапазон. Такова особенность функции `range` в Python.

Шаг изменения переменной цикла по умолчанию равен 1. Если его нужно изменить, указывают третье (необязательное) число в скобках после слова range — нужный **шаг**. Например, такой цикл выведет только нечётные степени числа 2 (2^1, 2^3 и т. д.):

```
for k in range(1,11,2):
  print(2**k)
```

С каждым шагом цикла переменная цикла может не только увеличиваться, но и уменьшаться. Для этого начальное значение должно быть больше конечного, а шаг — отрицательный. Следующая программа выводит квадраты натуральных чисел от 10 до 1 в порядке убывания:

```
for k in range(10,0,-1):
  print(k**2)
```

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 интервале от 2 до 1000. Простейший (но не самый быстрый) алгоритм решения такой задачи на псевдокоде выглядит так:

```
for n in range(2,1001):
  if число n простое:
    print(n)
```

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 получить 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):

```
for n in range(2,1001):
  count = 0
  for k in range(2,n):
    if n % k == 0:
      count += 1
  if count == 0:
    print (n)
```

Попробуем немного ускорить работу программы. Делители числа обязательно идут в парах, причём в любой паре меньший из делителей не превосходит $\sqrt{n}$ (иначе получается, что 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
while k*k <= n:
  if n % k == 0:
    count += 1
  k += 1
```

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 при `n%k == 0` выполним досрочный выход из цикла с помощью оператора **break** (оператор **break** во вложенном цикле прерывает только внутренний цикл, а внешний продолжает работать), причём переменная `count` уже не нужна:

```
k = 2
while k*k <= n:
  if n % k == 0: break
  k += 1
if k*k > n:
  print(n)
```

Если после завершения цикла `k*k > n` (нарушено условие в заголовке цикла), то число `n` простое.

В любом вложенном цикле переменная внутреннего цикла изменяется быстрее, чем переменная внешнего цикла. Рассмотрим, например, такой вложенный цикл:

```
for i in range(1,5):
  for k in range(1, i+1):
    print(i, k)
```

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.

Выводы

- Цикл по переменной использует переменную, для которой задаётся диапазон изменения: начальное и конечное значения, а также шаг изменения (по умолчанию равный 1).
- Если шаг положительный, конечное значение должно быть больше начального, иначе цикл не выполнится ни разу.
- Если шаг отрицательный, конечное значение должно быть меньше начального, иначе цикл не выполнится ни разу.
- Вложенный цикл — это цикл, находящийся в теле другого цикла.

Нарисуйте в тетради интеллект-карту этого параграфа.

Вопросы и задания

1. Сравните цикл по переменной и цикл с условием. Какие преимущества и недостатки есть у каждого из них?
2. В каком случае цикл по переменной не выполняется ни разу?
3. Верно ли, что любой цикл по переменной можно заменить циклом с условием? Верно ли обратное утверждение?
4. В каком случае можно заменить цикл с условием на цикл по переменной?

Темы сообщений

а) «Циклы по переменной в языке C»
б) «Циклы по переменной Javascript»

Проект

Программа для поиска простых чисел

§ 59
Процедуры

Ключевые слова:

- процедура
- параметр
- аргумент

Что такое процедура?

Предположим, что в нескольких местах программы требуется выводить на экран сообщение об ошибке: «Ошибка программы». Это можно сделать, например, так:

```
print ("Ошибка программы")
```

Конечно, можно вставить этот оператор вывода везде, где нужно вывести сообщение об ошибке. Но у такого решения есть два недостатка. Во-первых, строка-сообщение хранится 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.

```
def Error():
  print("Ошибка программы")
n = int (input())
if n < 0:
  Error()
```

Процедура, выделенная фоном, начинается с ключевого слова **def** (от англ. *define* — определить). После имени процедуры ставятся пустые скобки (чуть далее мы увидим, что они могут быть и непустыми!) и двоеточие. Тело процедуры записывается с отступом.

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 её **вызвать** по имени (не забыв скобки).

Процедура должна быть определена к моменту её вызова, т. е. должна быть выполнена инструкция **def**, которая создаёт объект-процедуру в памяти. Если процедура вызывается из основной программы, то нужно поместить её определение раньше точки вызова.

Как мы видели, использование процедур сокращает код, если какие-то операции выполняются несколько раз в разных местах программы. Кроме того, большую программу разбивают на несколько процедур для удобства и упрощения, оформляя в виде процедур отдельные этапы сложного алгоритма. Такой подход делает всю программу более понятной.

Процедуры с параметрами

Процедура `Error` при каждом вызове делает одно и то же. Более интересны процедуры, которым можно передавать **параметры** — данные, которые изменяют выполняемые действия. Значение параметра, которое передаётся процедуре, называют **аргументом**, или **фактическим параметром**.

Предположим, что в программе требуется многократно выводить на экран запись целого числа (в диапазоне 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:

```
n = 125
k = 128
while k > 0:
  print(n // k, end = "")
  n = n % k
  k = k // 2
```

Обратим внимание на вызов функции `print`, у которой указан именованный параметр[1] `end` — завершающий символ (по умолчанию — символ «новая строка», который обозначается как «`\n`»).

Напомним, что раньше мы уже использовали ещё один именованный параметр функции `print` — `sep` — разделитель между элементами списка вывода (по умолчанию — пробел).

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.

Работа процедуры зависит от параметра — числа, которое нужно перевести в двоичную систему. Получается такая программа:

```
def printBin(n):
  k = 128
  while k > 0:
    print(n // k, end = "")
    n = n % k
    k = k // 2
# основная программа
printBin(99)
```

[1] Так называют параметры, имеющие имена.

В заголовке процедуры в скобках записывают внутреннее имя параметра (т. е. имя, по которому к нему можно обращаться в процедуре). Основная программа содержит всего одну команду — вызов процедуры printBin. В скобках указан аргумент процедуры — значение параметра n, равное 99.

У процедуры может быть несколько параметров, в этом случае они перечисляются в заголовке через запятую. Например, процедуру, которая выводит на экран среднее арифметическое двух чисел, можно записать так:

```
def printSred (a, b):
  print((a+b)/2)
```

Локальные и глобальные переменные

В процедуре printBin используется **локальная** (внутренняя) **переменная** k — она известна только внутри этой процедуры, обратиться к ней из основной программы и из других процедур невозможно. Параметры процедуры — это тоже локальные переменные.

Если переменной присвоено значение в основной программе (вне всех процедур), она называется **глобальной**. Внутри процедуры можно обращаться к глобальным переменным, например эта программа выведет значение 5:

```
def qq():
  print(a)
a = 5
qq()
```

Однако для того, чтобы изменить значение глобальной переменной (не создавая локальную), в процедуре с помощью слова **global** надо указать, что мы используем именно глобальную переменную. Следующая программа выведет на экран число 1:

```
def qq():
  global a
  a = 1
a = 5
qq()
print(a)
```

Выводы

- Процедуры — это вспомогательные алгоритмы, которые могут многократно вызываться из основной программы и других подпрограмм.

- Данные, передаваемые в процедуру при вызове, называются параметрами процедуры. Внутри процедуры параметры обозначаются именами.
- При вызове процедуры в скобках записывают фактические значения параметров (аргументы).
- Если у процедуры несколько параметров, они перечисляются через запятую.
- Переменные, введённые в процедуре, доступны только внутри этой процедуры и называются локальными переменными.
- Глобальные переменные — это переменные, введённые в основной программе. Они доступны в процедуре для чтения, но для их изменения внутри процедуры нужно объявить их после служебного слова **global**.

Нарисуйте в тетради интеллект-карту этого параграфа.

Вопросы и задания

1. Что такое процедуры? В чём смысл их использования?
2. Как оформляются процедуры в Python? Достаточно ли включить процедуру в текст программы, чтобы она «сработала»?
3. Что такое параметры? Зачем они используются?
4. Какие переменные называются локальными? глобальными?
5. Как в процедуре прочитать и изменить значение глобальной переменной?
6. Как оформляются процедуры, имеющие несколько параметров?

Подготовьте сообщение

а) «Локальные и глобальные переменные»
б) «Почему глобальных переменных нужно избегать?»

§ 60
Функции

Ключевые слова:

- функция
- параметр
- результат функции
- вызов функции
- логическая функция

Что такое функция?

С функциями вы уже знакомы, потому что применяли встроенные функции языка Python (`input`, `int`, `randint`).

Например, при вводе данных мы использовали функцию `input` — она возвращает символьную строку, введённую с клавиатуры:

```
s = input()
```

Чтобы преобразовать эту строку в число, мы вызывали ещё одну функцию — `int`, которая получала на вход результат работы функции `input` и возвращала целое число, полученное из введённой строки:

```
n = int(input())
```

Теперь вы научитесь писать свои собственные функции.

Функция, как и процедура, — это вспомогательный алгоритм, который может принимать аргументы. Но, в отличие от процедуры, функция всегда **возвращает значение-результат**. Результатом может быть число, символ, символьная строка или любой другой объект.

Начнём с простой функции `lastDigit`, которая определяет последнюю цифру переданного ей целого числа. Схема её работы показана на рис. 8.11.

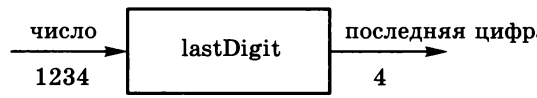


Рис. 8.11

Эта функция объявляется так:

```
def lastDigit(n):
  d = n % 10
  return d
```

Название функции должно говорить о том, что делает эта функция. Мы выбрали для неё имя `lastDigit`, образованное от английских слов *last* — последний и *digit* — цифра.

Объявление начинается служебным словом **def** (от англ. *define* — определить). В скобках после имени новой функции указан один параметр с именем *n*. В отличие от многих других языков программирования (например, Паскаля или C) в Python не нужно указывать тип параметров. Тело функции записывается с отступом, так же как и тело условного оператора или цикла.

После служебного слова return (от англ. *return* — вернуть) записывается результат, который возвращает функция. В данном случае результат — это значение переменной d, в которую мы ранее записали последнюю цифру числа. В функции может быть несколько операторов return, после выполнения любого из них работа функции заканчивается.

Теперь эту функцию можно использовать, например, при выводе результатов:

```
n = 4321
print("Число", n, "оканчивается на", lastDigit(n))
```

или при вычислениях:

```
sum = 0
while n > 0:
  sum += lastDigit(n)
  n = n // 10
```

В этом фрагменте программы определяется сумма цифр числа, записанного в переменную n.

Программы с вызовом функций легче понимать, потому что нам уже не нужно держать в голове алгоритмы, которые используются внутри этих функций, важен только их результат. Это один из способов борьбы с возрастающей сложностью программ.

Внутри функции можно использовать любые конструкции языка, например циклы и условные операторы. Например, функция maxDigit возвращает максимальную цифру переданного ей числа:

```
def maxDigit(n):
  M = -1
  while n > 0:
    d = n % 10
    if d > M: M = d
    n = n // 10
  return M
```

а функция sumDigits — сумму цифр числа:

```
def sumDigits(n):
  sum = 0
  while n!= 0:
    sum += n % 10
    n = n // 10
  return sum
```

Вызов функции

Ваши собственные функции можно применять точно так же, как и стандартные функции. Их можно вызывать везде, где может использоваться выражение того типа, который возвращает функция. Вызвать функцию sumDigits можно, например, так:

```
print(sumDigits(12345))
```

Вот несколько более сложных примеров:

```
x = 2*sumDigits(n+5)
z = sumDigits(k) + sumDigits(m)
if sumDigits(n) % 2 == 0:
  print("Сумма цифр чётная")
  print("Она равна", sumDigits(n))
```

Одна функция может вызывать другую. Например, эта функция вычисляет максимальную цифру квадрата числа:

```
def maxDigitX2(x):
  s = maxDigit(x*x) # вызов функции из функции
  return s
```

Функция, которая находит среднее по величине из трёх различных чисел (т. е. число, заключённое между двумя остальными), может быть определена так:

```
def middle(a, b, c):
  mi = min(a, b, c)
  ma = max(a, b, c)
  return a + b + c - mi - ma
```

Она использует встроенные функции min и max. Идея решения состоит в том, что если из суммы трёх чисел вычесть минимальное и максимальное, то получится как раз третье число.

Как вернуть несколько значений?

Функция может возвращать несколько значений. Например, функцию, которая вычисляет сразу и частное, и остаток от деления двух чисел[1)], можно написать так:

---

1) В Python есть такая встроенная функция.

```
def divmod(x, y):
  d = x // y
  m = x % y
  return d, m
```

При вызове такой функции её результат можно записать в две различные переменные:

```
a, b = divmod(7, 3)
print(a, b)          # 2 1
```

Если указать только одну переменную, мы получим **кортеж** — набор элементов, который заключается в круглые скобки:

```
q = divmod(7, 3)
print(q)       # (2, 1)
```

Логические функции

Часто применяют функции, которые возвращают **логическое значение** (`True` или `False`). Иначе говоря, такая *логическая* функция отвечает на вопрос «да или нет?» и возвращает 1 бит информации.

Построим функцию, которая определяет чётность числа. Напомним, что число чётное, если остаток от его деления на 2 равен нулю (в этом случае функция должна вернуть логическое значение `True` — истина), и нечётное, если этот остаток — не ноль (тут функция должна вернуть значение `False` — ложь):

```
def even(n):
  if n % 2 == 0:
    return True
  else:
    return False
```

Можно записать то же самое несколько проще:

```
def even(n):
  return (n % 2 == 0)
```

Здесь `(n%2 == 0)` — это условие, которое может быть истинно или ложно. Это логическое значение и вернёт функция.

Вызвать функцию `even` из основной программы можно так:

```
n = int(input())
if even(n):
  print("Число", n, "чётное.")
else:
  print("Число", n, "нечётное.")
```

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:

```
for i in range(2,1001):
  if  i - простое :
    print(i)
```

Предположим, что у нас уже есть логическая функция `isPrime`, которая определяет простоту числа, переданного ей как параметр, и возвращает значение `True`, если число простое, и `False` в противном случае. Такую функцию можно использовать вместо выделенного блока алгоритма:

```
if isPrime (i):
  print(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]:

```
def isPrime (n):
  k = 2
  while k*k <= n and n % k != 0:
    k += 1
  return (k*k > n)
```

Эта функция возвращает *логическое* значение, которое определяется как

```
k*k > n
```

Если это условие истинно, то функция возвращает `True`, иначе — `False`.

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:

```
n = int (input())
while isPrime(n):
  print(n, " - простое число")
  n = int (input())
```

[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.

Выводы

- Функция — это вспомогательный алгоритм (подпрограмма), возвращающий результат.
- Результат работы функции возвращается с помощью оператора **return**, при выполнении которого работа функции завершается.
- Функция может возвращать несколько значений.
- Логическая функция возвращает значение `True` («истина») или `False` («ложь»).

Нарисуйте в тетради интеллект-карту этого параграфа.

Вопросы и задания

1. Чем отличается функция от процедуры?
2. Как по тексту программы определить, какое значение возвращает функция?
3. Какие функции называются логическими? Зачем они нужны?
4. Проверьте с помощью компьютера, какое значение вернёт функция, если вы забудете записать в конце оператор **return**.

Подготовьте сообщение

а) «Функции в языках Python и Паскаль»
б) «Функции в языке Javascript»
в) «Функции в языке C»

§ 61
Рекурсия

Ключевые слова:

- рекурсия
- рекурсивная процедура
- базовый случай
- стек
- указатель стека

Что такое рекурсия?

Определение натуральных чисел в математике состоит из двух частей:

1) 1 — натуральное число;
2) если n — натуральное число, то $n + 1$ — тоже натуральное число.

Вторая часть этого определения называется **индуктивной**: натуральное число определяется через другое натуральное число, и таким образом определяется всё множество натуральных чисел. В программировании этот приём называют *рекурсией*.

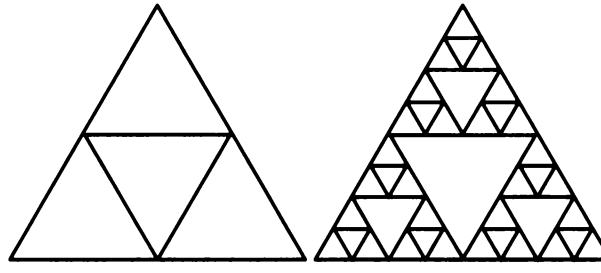
Рекурсия — это способ определения множества объектов через само это множество на основе заданных простых базовых случаев.

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 8.12 показан треугольник Серпинского — один из первых фракталов, который предложил в 1915 году польский математик В. Серпинский.



а *б*

Рис. 8.12

Равносторонний треугольник делится на 4 равных треугольника меньшего размера (рис. 8.12, *а*), затем каждый из полученных треугольников, кроме центрального, снова делится на 4 ещё более мелких треугольника и т. д. На рисунке 8.12, *б* 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 нанизаны на первый стержень. За один раз можно перекладывать только один диск, причем разрешается класть только меньший диск на больший, но не наоборот. Есть также и третий стержень, который можно использовать в качестве вспомогательного (рис. 8.13).

Рис. 8.13

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 диска 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)
```

Здесь запись `1 -> 3` обозначает «перенести один диск со стержня 1 на стержень 3». Процедура `перенести` (которую нам предстоит написать) принимает три параметра: число дисков, номера

начального и конечного стержней. Таким образом, мы свели задачу переноса n дисков к двум задачам переноса $n - 1$ дисков и одному элементарному действию — переносу одного диска. Заметим, что при известных номерах начального и конечного стержней легко рассчитать номер вспомогательного стержня, так как сумма номеров равна $1 + 2 + 3 = 6$. Получается такая (пока не совсем верная) процедура:

```
def Hanoi(n, k, m):
  p = 6 - k - m
  Hanoi(n-1, k, p)
  print(k, "->", m)
  Hanoi(n-1, p, m)
```

Эта процедура вызывает сама себя, но с другими параметрами. 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 остановки, и процедура бесконечно вызывает сама 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 языке Python используется оператор **`return`**. Правильная версия процедуры выглядит так:

```
def Hanoi (n, k, m):
  if n == 0:
    return
  p = 6 - k - m
  Hanoi(n-1, k, p)
  print(k, "->", m)
  Hanoi(n-1, p, m)
```

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.

Пример: вычисление суммы цифр числа

Начнём с примера. Эта функция вычисляет сумму цифр натурального числа:

```
def sumDigits(n):
  if n < 10: return n            # (1)
  d = n % 10                     # (2)
  s = d + sumDigits(n // 10)     # (3)
  return s
```

Рассмотрим её по строкам, используя нумерацию в комментариях.

В самом начале, в строке (1), проверяем особый случай: если натуральное число однозначное, сумма его цифр равна самому числу и можно сразу вернуть результат.

Далее рассуждаем так: сумма цифр числа равна значению последней цифры плюс сумма цифр числа, полученного из исходного отсечением этой последней цифры. В строке (2) вычисляем последнюю цифру числа, а в строке (3) записываем только что сформулированный алгоритм на языке Python. При этом функция `sumDigits` вызывает сама себя, но уже для другого аргумента: `n // 10` — это число, которое получается после отсечения последней цифры.

Посмотрим, как работает такая функция для числа 123 (рис. 8.14). При первом вызове в локальную переменную `d` записывается значение последней цифры (3). В следующей строке функция вызывает сама себя с аргументом 12. В памяти создаётся **новый набор локальных переменных** `s` и `d`, в переменную `d` записывается число 2 и происходит третий вызов функции, уже с аргументом 1. Теперь срабатывает первая строка в функции, возвращается результат 1 и нового вызова не происходит. Этот результат подставляется вместо `sumDigits(1)`, и таким образом

второй вызов функции завершает работу с результатом 2 + 1 = 3. Это число подставляется вместо `sumDigits(12)`, после этого первый вызов функции возвращает 3 + 3 = 6.

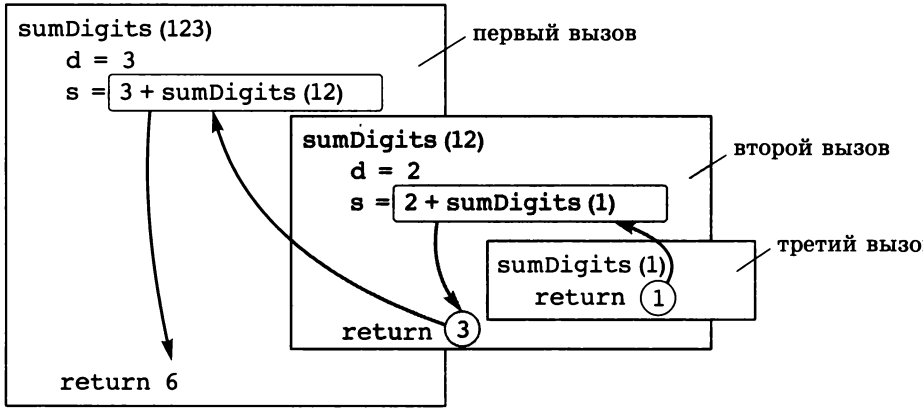


Рис. 8.14

В этой задаче локальные переменные `d` и `s` введены только для того, чтобы сделать объяснение более понятным. Можно было записать эту функцию в краткой форме:

```
def sumDigits (n):
  if n < 10:
    return n
  else:
    return (n % 10) + sumDigits(n // 10)
```

Не всякая задача может быть успешно решена с помощью рекурсии. В чём особенность задачи с вычислением суммы цифр числа? Нам удалось свести её к более простой задаче **того же типа** — к задаче вычисления суммы цифр более короткого числа (рис. 8.15).

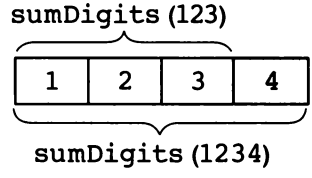


Рис. 8.15

Если решение задачи имеет такую «вложенную» структуру, рекурсия очень часто бывает полезна.

В более сложных случаях может получиться, что исходная задача включает решение *нескольких* более простых задач того же типа, тогда функция будет вызывать сама себя несколько раз.

При использовании рекурсивных решений нужно помнить про одну особенность — в рекурсивной функции обязательно должен быть **базовый случай** — вариант, при котором нового вызова не происходит. Действительно, если пропустить строку

```
if n < 10: return n
```

в функции `sumDigits`, то при каждом вызове будет происходить новый вызов функции, и эти вызовы (теоретически) никогда не закончатся, точнее, закончатся, когда будет исчерпана доступная память компьютера и программа завершится аварийно. Получится бесконечная рекурсия — серьёзная ошибка в программе. Для надёжности лучше начинать любую рекурсивную функцию с обработки варианта, когда рекурсия останавливается, как это сделано в нашем примере.

Ещё примеры

Пример 1. Составим процедуру, которая переводит натуральное число в двоичную систему счисления. 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:

```
while n!= 0:
  print (n % 2, end = "")
  n = n // 2
```

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 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 `n//2`, а затем — последнюю цифру: `n%2`. Если полученное число-параметр равно

нулю, нужно выйти из процедуры. Такой алгоритм очень просто программируется:

```
def printBin(n):
  if n == 0: return
  printBin(n // 2)
  print(n % 2, end = "")
```

Конечно, то же самое можно было сделать и с помощью цикла. Отсюда следует важный вывод: **рекурсия заменяет цикл.** При этом программа во многих случаях становится более понятной.

Пример 2. Напишем рекурсивную функцию, которая вычисляет натуральную степень заданного вещественного числа x. Если n — натуральное число, то x^n можно записать в виде

$$x^n = x \cdot x^{n-1}.$$

Мы записали n-ю степень x через $(n-1)$-ю степень того же числа. Рекурсия заканчивается, когда $n = 1$, в этом случае результат равен x:

```
def pow(x, n):
  if n == 1:
    return x
  else:
    return x*pow(x,n-1)
```

Пример 3. С алгоритмом Евклида мы уже знакомы (см. § 57). Его можно легко сформулировать в рекурсивном виде. 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:

```
def NOD(a, b):
  if a == 0 or b == 0:
    return a + b
  if a > b:
    return NOD(a - b, b)
  else:
    return NOD(a, b - a)
```

Заметим, что при равенстве одного из чисел нулю второе число совпадает с суммой двух, поэтому в качестве результата функции возвращается a + b.

Модифицированный алгоритм Евклида (использующий операцию взятия остатка) записывается так:

```
def NOD(a, b):
  if a == 0 or b == 0:
    return a + b
  if a > b:
    return NOD(a % b, b)
  else:
    return NOD(a, b % a)
```

или даже так:

```
def NOD(a, b):
  if b == 0:
    return a
  return NOD(b, a % b)
```

Заметьте, что условие окончания рекурсии тоже упростилось: достаточно проверить, что на очередном шаге остаток от деления (второй параметр) стал равен нулю, тогда результат — это значение первого параметра a.

Как работает рекурсия

Рассмотрим вычисление **факториала** — так называют произведение всех натуральных чисел от 1 до заданного числа N: $N! = 1 \cdot 2 \cdot 3 \cdot \ldots N$. Факториал может быть также введён с помощью рекуррентной формулы, которая связывает факториал данного числа с факториалом предыдущего:

$$N! = \begin{cases} 1, & \text{при } N = 1; \\ N \cdot (N-1)!, & \text{при } N \geq 2. \end{cases}$$

Здесь первая часть описывает базовый случай (условие окончания рекурсии), а вторая — переход к следующему шагу. Запишем соответствующую функцию на языке Python, добавив в начале и в конце операторы вывода:

```
def Fact(N):                          →3
  print ("->", N)                      →2
  if N <= 1: F = 1                      →1
  else:                                 ←1
    F = N * Fact (N - 1)                ←2
  print ("<-", N)                       ←3
  return F
```

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 и адрес возврата, там же выделяется место под локальные переменные.

На рисунке 8.16, *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, потом — адрес возврата *A* (рис. 8.16, *б*), затем в стеке размещаются локальные переменные (здесь — переменная *F*). При втором и третьем вложенных вызовах в стек добавляются аналогичные блоки данных (рис. 8.16, *в* и *г*). Заметьте, что в нашем случае адрес возврата A_F (точка в функции *Fact* после рекурсивного вызова) в последних двух блоках будет один и тот же.

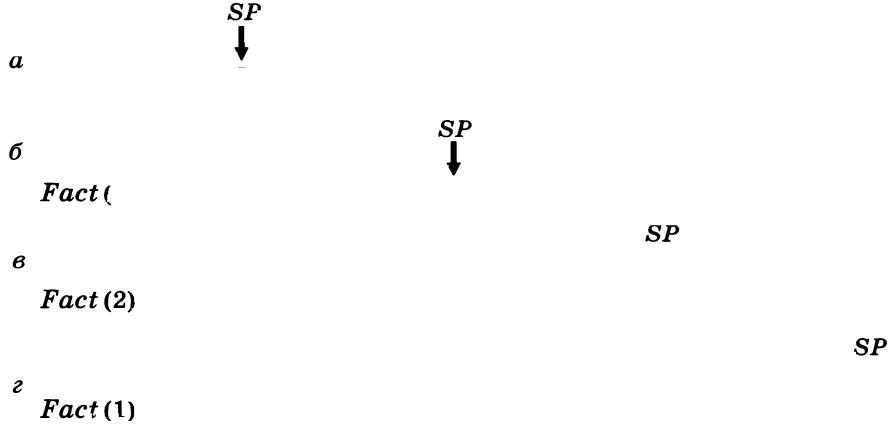


Рис. 8.16

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.

А 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). Например, для вычисления факториала можно использовать обычный цикл:

```
def Fact(n):
  f = 1
  for i in range(2, n + 1):
    f *= i
  return f
```

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 процедура использует много локальных данных.

Анализ рекурсивных функций

Задача состоит в том, чтобы выяснить, какое значение вернёт рекурсивная функция при известных входных данных.

Задача 1. Определите результат работы программы:

```
def f(x):
  if x < 3:
    return 1
  else:
    return f(x - 1) + 2
print(f(5))
```

Решение. Если функция содержит только один рекурсивный вызов, можно использовать прямую подстановку. Как следует из последней строки программы, нас интересует значение $f(5)$. Смотрим в тело функции: условие $5 < 3$ ложно, поэтому срабатывает рекурсивный вызов

$f(5) = f(4) + 2$

Далее таким же способом получаем $f(4) = f(3) + 2$, что даёт

$f(5) = f(3) + 2 + 2 = f(3) + 4$

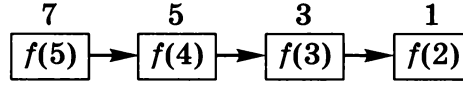
При следующем вызове $f(3) = f(2) + 2$, так что

$f(5) = f(3) + 4 = f(2) + 6$

При $f(2)$, срабатывает условие $x < 3$ и функция возвращает 1, так что $f(2) = 1$. Поэтому

$f(5) = f(2) + 6 = 7$.

Заметим, что в данном случае рекурсивные вызовы функций можно записать в виде цепочки (сверху указаны результаты вызовов функций):

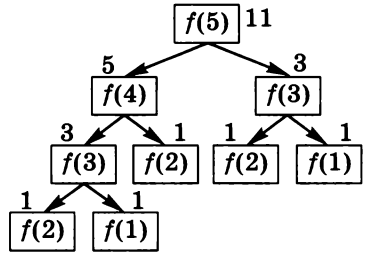


Ответ: 7.

Задача 2. Определите результат работы программы:

```
def f(x):
  if x < 3:
    return 1
  else:
    return f(x - 1) + 2*f(x - 2)
print(f(5))
```

Решение. Эту задачу также можно решать подстановкой, как и предыдущую. В отличие от задачи 1 в случае продолжения рекурсии на каждом шаге функция вызывает сама себя не один раз, а дважды. Эти вызовы можно представить в виде дерева (результаты вычисления записаны рядом с каждым блоком):



Обратите внимание, что значения $f(3)$ и $f(1)$ вычисляются дважды, а значение $f(2)$ — трижды. Это говорит о том, что использование рекурсии может замедлить вычисления.

Значительно удобнее использовать другой способ — составить таблицу значений функции для первых натуральных чисел. Из программы легко понять, что для вычисления $f(x)$ используется формула:

$$f(x) = \begin{cases} 1, & \text{при } x < 3; \\ f(x-1) + 2f(x-2), & \text{при } x \geq 3. \end{cases}$$

Такие формулы, в которых значение функции в точке x вычисляется через её значения в других точках, называются **рекуррентными**. Сначала внесём в таблицу базовые случаи (при $x < 3$), для которых значения функции известны и равны 1:

x	1	2	3	4	5
$f(x)$	1	1			

Заполним остальные ячейки слева направо, используя вторую часть формулы:

x	1	2	3	4	5
$f(x)$	1	1	3	5	11

Например, $f(5) = f(4) + 2 \cdot f(3) = 5 + 2 \cdot 3 = 11$ (значения $f(4)$ и $f(3)$ к этому моменту в таблице уже есть).

Ответ: 11.

Задача 3. Сколько символов «*» будет выведено на экран в результате работы этой программы:

```
def f(x):
  if x > 0: g(x - 1)
def g(x):
  print("*")
  if x > 1: f(x - 3)
f(11)
```

Решение. В этой программе две рекурсивные процедуры: они не возвращают никакого результата, а просто выполняют какие-то действия, в нашем случае — выводят на экран звёздочки.

Обе процедуры не вызывают сами себя напрямую. Но можно заметить, что они вызывают друг друга, а поэтому фактически вызывают сами себя через другую процедуру. Такой приём называется **косвенной рекурсией**.

В каждой процедуре есть только один вызов другой процедуры, поэтому получается такая цепочка вызовов:

$f(11) \to g(10) \to f(7) \to g(6) \to f(3) \to g(2) \to f(-1)$.

Обратим внимание, что в теле процедуры $f(x)$ вообще ничего не выводится, а при каждом вызове $g(x)$ выводится одна звёздочка. Поэтому остаётся подсчитать, сколько раз управление передаётся в $g(x)$ — три раза, столько звёздочек и будет выведено.

Ответ: 3.

Задача 4. Сколько символов «*» будет выведено на экран в результате работы программы?

```
def f(x):
  if x > 0:
    g(x - 1)
    f(x - 2)
  print("*")
def g(x):
  print("*")
  if x > 1: f(x - 3)
f(9)
```

Решение. В отличие от предыдущей задачи здесь процедура $f(x)$ вызывает сама себя и ещё «парную» процедуру $g(x)$, поэтому обойтись линейной цепочкой вызовов не получится.

Будем использовать табличный метод, а результатом работы процедуры считать количество звёздочек, которые она выводит. Процедура $f(x)$ выводит одну звёздочку при $x \le 0$, а при $x > 0$ — столько же звёздочек, сколько $g(x-1)$ и затем $f(x-2)$, а потом ещё одну, поэтому рекуррентная формула запишется так:

$$f(x) = \begin{cases} 1, \text{ при } x \le 0; \\ g(x-1) + f(x-2) + 1, \text{ при } x > 0. \end{cases}$$

Функция $g(x)$ всегда выводит одну звёздочку (первая строка в теле функции), а при $x > 1$ — ещё столько звёздочек, сколько выводится при вызове $f(x-3)$. Получаем

$$g(x) = \begin{cases} 1, \text{ при } x \le 1; \\ 1 + f(x-3), \text{ при } x > 1. \end{cases}$$

Сначала заполняем таблицу для базовых случаев:

x	–1	0	1	2	3	4	5	6	7	8	9
$f(x)$	1	1									
$g(x)$	1	1	1								

Заполним остальные ячейки слева направо, используя вторые части формул:

x	–1	0	1	2	3	4	5	6	7	8	9
$f(x)$	1	1	3	3	6	6	11	11	19	19	32
$g(x)$	1	1	1	2	2	4	4	7	7	12	12

Например: $f(9) = g(8) + f(7) + 1 = 12 + 19 + 1 = 32$.
Ответ: 32.

Задача 5. Определите результат работы программы.

```
def f(n):
  if n < 2: return 1
  return f(n - 1) + 2*g(n - 1)
def g(n):
  if n < 2: return 1
  return g(n - 1) + f(n - 1)
print(f(5) + g(5))
```

Решение. В этой задаче две функции, каждая из них вызывает вторую и саму себя. Из последней строки определяем, что результат работы программы — это сумма $f(5) + g(5)$. Применим табличный метод. Изучив функции, построим рекуррентные формулы:

$$f(n) = \begin{cases} 1, \text{ при } n < 2; \\ f(n-1) + 2g(n-1), \text{ при } n \geq 2; \end{cases}$$

$$g(n) = \begin{cases} 1, \text{ при } n < 2; \\ g(n-1) + f(n-1), \text{ при } n \geq 2. \end{cases}$$

Сначала внесём в таблицу базовый случай (при $n < 2$), для которого значения обеих функций известны и равны 1:

x	1	2	3	4	5
$f(x)$	1				
$g(x)$	1				

Заполним остальные ячейки слева направо, используя вторые части формул:

x	1	2	3	4	5
$f(x)$	1	3	7	17	41
$g(x)$	1	2	5	12	29

Ответ: 41 + 29 = 70.

Выводы

- Рекурсия — это способ определения множества объектов через само это множество на основе заданных простых базовых случаев.
- Рекурсивные алгоритмы основаны на последовательном сведении исходной задачи ко всё более простым задачам такого же типа (с другими параметрами).
- Рекурсивная процедура (функция) — это процедура (функция), которая вызывает сама себя напрямую или через другие процедуры и функции.
- Рекурсия может служить заменой циклу. Любой рекурсивный алгоритм можно записать без рекурсии, но во многих случаях такая запись более длинная и менее понятная.
- Стек — особая область памяти, в которой хранятся локальные переменные и адреса возврата из процедур и функций.

Нарисуйте в тетради интеллект-карту этого параграфа.

Вопросы и задания

1. Что такое рекурсия? Приведите примеры.
2. Как вы думаете, почему любое рекурсивное определение состоит из двух частей?
3. Расскажите о задаче «Ханойские башни». Попытайтесь придумать или найти в дополнительных источниках алгоритм её решения, не использующий рекурсию.
4. Процедура *А* вызывает процедуру *Б*, а процедура *Б* — процедуру *А* и сама себя. Какую из них можно назвать рекурсивной?
5. В каком случае рекурсия никогда не остановится? Докажите, что в рассмотренных в параграфе задачах этого не случится.
6. Что такое стек? Как он используется при выполнении программ?
7. Почему при использовании рекурсии может случиться переполнение стека?
8. Назовите достоинства и недостатки рекурсии. Когда её следует использовать, а когда — нет?

Подготовьте сообщение

а) «Задача о Ханойских башнях»
б) «Рекурсивные фигуры»
в) «Рекурсивные функции в языке С»
г) «Рекурсивные функции в языке Javascript»

Проекты


а) Рекурсия в рекламе и в искусстве
б) Рекурсивные алгоритмы без рекурсии
в) Программа для построения фракталов
г) Рекурсивный перебор вариантов

§ 62
Массивы

Ключевые слова:

- массив
- список
- индекс элемента
- ввод и вывод массива
- перебор элементов
- генератор

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 (или таблицей).

Массив — это группа переменных одного типа, расположенных в памяти рядом и имеющих общее имя. Каждая ячейка в массиве имеет уникальный номер (индекс).

Для работы с массивами нужно в первую очередь научиться:

- выделять память нужного размера под массив;
- записывать данные в нужную ячейку;
- читать данные из ячейки массива.

В языке Python нет такой структуры, как «массив». Вместо этого для хранения группы однотипных[1] объектов используют списки (тип данных `list`).

Список в Python — это набор элементов, каждый из которых имеет свой номер (индекс). Нумерация всегда начинается с нуля (как в С-подобных языках), второй по счёту элемент имеет номер 1 и т. д. В отличие от обычных массивов в большинстве языков программирования список — это динамическая структура, его размер можно изменять во время выполнения программы (удалять и добавлять элементы), при этом все операции по управлению памятью берёт на себя транслятор.

Список можно создать перечислением элементов через запятую в квадратных скобках, например так:

```
A = [1, 3, 4, 23, 5]
```

Списки можно «складывать» с помощью знака «+», например, показанный выше список можно было построить так:

```
A = [1, 3] + [4, 23] + [5]
```

Сложение одинаковых списков заменяется умножением «*». Вот так создаётся список из 10 элементов, заполненный нулями:

```
A = [0]*10
```

В более сложных случаях используют **генераторы списков** — выражения, напоминающие цикл, с помощью которых заполняются элементы вновь созданного списка:

```
A =[i  for i in range(10) ]
```

Как вы знаете, цикл `for i in range(10)` перебирает все значения `i` от 0 до 9. Выражение перед словом `for` (в данном случае — `i`) — это то, что записывается в очередной элемент списка для каждого `i`. В приведённом примере список заполняется значениями, которые последовательно принимает переменная *i*, т. е. получим такой список:

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Такой же список можно получить, применив функцию `list` (создание списка) к данным, полученным с помощью функции `range`:

```
A = list (range(10))
```

---

[1] И не только однотипных.

Для заполнения списка квадратами этих чисел можно использовать такой генератор:

```
A =[ i*i  for i in range(10)]
```

В конце записи генератора можно добавить условие отбора. В этом случае в список включаются лишь те из элементов, перебираемых в цикле, которые удовлетворяют этому условию. Например следующий генератор составляет список из всех простых чисел в диапазоне от 0 до 99:

```
A = [i for i in range(100)  if isPrime(i) ]
```

Здесь `isPrime` — логическая функция, которая определяет простоту числа (см. § 60).

Далее в этом параграфе под выражением «массив» мы будем подразумевать «однотипные данные, хранящиеся в виде списка». Переменная `i` будет обозначать индекс элемента списка.

Часто в тестовых и учебных программах массив заполняют случайными числами. Это тоже можно сделать с помощью генератора:

```
from random import randint
A = [ randint(20, 100)  for x in range(10)]
```

Здесь создаётся массив из 10 элементов и заполняется случайными числами из отрезка [20, 100]. Для этого используется функция `randint`, которая импортируется из модуля `random`.

Длина списка (количество элементов в нём) определяется с помощью функции `len`:

```
N = len(A)
```

Добавление и удаление элементов массива

Проще всего добавить элемент в конец массива, для этого используется метод `append`:

```
A = [1, 2, 3]
x = 5
A.append(x + 3)     # получаем A = [1, 2, 3, 8]
```

Метод — это подпрограмма (процедура или функция), связанная с каким-то объектом. Чтобы вызвать метод, используют **точечную запись**: название метода записывают через точку после названия объекта.

Для того чтобы вставить элемент в любое место массива, применяется другой метод — `insert`. В параметрах сначала указывается номер нового элемента в массиве, а затем — его значение:

```
A = [1, 2, 3]
A.insert(1, 35)     # получаем A = [1, 35, 2, 3]
```

В этом примере новый элемент встаёт на место *A*[1], а остальные элементы сдвигаются в конец массива.

Для удаления элемента с известным номером используется метод pop. Если номер не указан, удаляется последний элемент. Удалённый элемент возвращается как результат работы метода:

```
A = [1, 2, 3]
x = A.pop(1)    # получаем A = [1, 3], x = 2
x = A.pop()     # теперь   A = [1],    x = 3
```

Элементы массива можно удалять не только по номеру, но и по значению. Для этого используется метод remove. Например, первый элемент, равный 99, удаляется так:

```
A = [1, 2, 99, 3, 99]
A.remove(99)    # получаем A = [1, 2, 3, 99]
```

Ввод и вывод массива

Далее во всех примерах мы будем считать, что в программе создан список A, в котором хранится массив из N однотипных элементов — целых чисел. Переменная i будет обозначать индекс элемента списка (массива).

Чтобы ввести значения элементов массива с клавиатуры, нужно использовать цикл:

```
for i in range(N):
  print ("A[", i, "]=", sep = "", end = "")
  A[i] = int(input())
```

В этом примере перед вводом очередного элемента массива на экран выводится подсказка. Например, при вводе 3-го элемента будет выведено A[3]=.

Если никакие подсказки нам не нужны, создать массив из *N* элементов и ввести их значения можно с помощью генератора списка:

```
A = [ int(input())  for i in range(N)]
```

Здесь на каждом шаге цикла строка, введённая пользователем, преобразуется в целое число с помощью функции int, и это число добавляется к массиву.

Возможен ещё один вариант, когда весь массив вводится в одной строке. В этом случае строку, полученную от функции input, нужно «расщепить» на части с помощью метода split:

```
data = input()
s = data.split()
```

или сразу:

```
s = input().split()
```

Например, если ввести строку "1 2 3 4 5", то после «расщепления» мы получим массив

```
["1", "2", "3", "4", "5"]
```

Это массив символьных строк. Для того чтобы построить массив (список), состоящий из целых чисел, нужно применить к каждому элементу списка функцию int:

```
A = [ int(x)  for x in s]
```

Вместо генератора можно было использовать функцию map:

```
A = list(map(int, s))
```

Такая запись означает «применить функцию int ко всем элементам списка *s* и составить из полученных чисел новый список (объект типа list)».

Теперь поговорим о выводе массива на экран. Самый простой способ — вывести массив как один объект:

```
print (A)
```

В этом случае весь список берётся в квадратные скобки, и элементы разделяются запятыми:

```
[1, 12, 32, 14, 25]
```

Вывести массив на экран можно и поэлементно:

```
for i in range(N):
  print (A[i], end = " ")
```

После вывода каждого элемента ставится пробел (иначе все значения сольются в одну строку):

```
1 12 32 14 25
```

Удобно записывать такой цикл несколько иначе:

```
for x in A:
  print (x, end = " ")
```

Здесь не используется переменная-индекс i, а просто перебираются все элементы списка: на каждом шаге в переменную x заносится значение очередного элемента массива (в порядке возрастания индексов).

В языке Python есть ещё один способ вывести все элементы массива через пробел:

```
print(*A)
```

Здесь знак «*» означает, что вместо одного объекта (массива) мы хотим получить его отдельные элементы, перечисленные через запятую.

Перебор элементов

Перебор элементов массива состоит в том, что мы в цикле просматриваем все элементы и, если нужно, выполняем с каждым из них некоторую операцию. Переменная цикла изменяется от 0 до N - 1, где N — количество элементов массива, т. е. в диапазоне range(N):

```
for i in range(N):
  A[i] += 1
```

в этом примере все элементы массива A увеличиваются на 1.

Если массив изменять не нужно, для перебора его элементов удобнее всего использовать такой цикл:

```
for x in A:
  ...
```

Здесь вместо многоточия можно добавлять операторы, работающие с копией элемента, записанной в переменную x. Обратите внимание, что изменение переменной x в теле цикла не приведёт к изменению соответствующего элемента массива A.

Заметим, что для первой задачи (увеличить все элементы массива на 1) есть красивое решение в стиле Python, использующее генератор списка, который построит новый массив:

```
A = [x+1 for x in A]
```

В цикле перебираются все элементы исходного массива, и в новый массив они попадают после увеличения на 1.

Во многих задачах требуется 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 просматриваем все элементы массива. Если для очередного элемента выполняется заданное условие, то увеличиваем счётчик на 1. На псевдокоде этот алгоритм выглядит так:

```
счётчик = 0
for x in A:
  if условие выполняется для x:
    счётчик += 1
```

Предположим, что в массиве A записаны данные о росте игроков баскетбольной команды. Найдем количество игроков, рост которых больше 180 см, но меньше 190 см. В следующей программе используется переменная-счётчик count:

```
count = 0
for x in A:
  if 180 < x and x < 190:
    count += 1
```

Теперь усложним задачу: требуется найти средний рост этих игроков. Для этого нужно дополнительно в отдельной переменной складывать все нужные значения, а после завершения цикла разделить эту сумму на количество. Начальное значение переменной summa, в которой накапливается сумма, тоже должно быть равно нулю.

```
count = 0
summa = 0
for x in A:
  if 180 < x and x < 190:
    count += 1
    summa += x
print(summa/count)
```

Суммирование элементов массива — это очень распространённая операция, поэтому для суммирования элементов списка в Python существует встроенная функция sum:

```
print(sum(A))
```

С её помощью можно решить предыдущую задачу более элегантно, в стиле языка Python: сначала выделить в дополнительный массив все нужные элементы[1], а затем разделить их сумму на количество (длину массива).

Для построения нового массива будем использовать генератор:

```
B = [x for x in A  if 180 < x and x < 190 ]
```

Таким образом, мы отбираем в массив B те элементы из массива A, которые удовлетворяют этому условию. Теперь для вывода среднего роста выбранных игроков остается разделить сумму элементов нового массива на их количество:

```
print(sum(B)/len(B))
```

[1] Хотя это приведёт к дополнительному расходу памяти и может быть нежелательно, если список очень большой.

Выводы

- Массив — это группа переменных одного типа, расположенных в памяти рядом и имеющих общее имя. Каждая ячейка массива имеет уникальный индекс (как правило, это номер элемента).
- Цикл `for i in range(N)` используется для перебора всех индексов от `0` до `N - 1`.
- Если элементы массива не нужно изменять, для перебора лучше использовать цикл `for x in A`.
- Для создания списков (массивов) можно использовать генераторы.

Нарисуйте в тетради интеллект-карту этого параграфа.

Вопросы и задания

1. Как вы думаете, почему в языке Python нет массивов, а вместо них используются списки?
2. Какие способы создания списков вы знаете?
3. Зачем нужны генераторы списков с условием?
4. Как построить массив, состоящий из 15 единиц, с помощью генератора списка?
5. Как обращаться к отдельному элементу массива?
6. Как ввести массив с клавиатуры?
7. Как вывести массив на экран? Приведите разные варианты решения этой задачи. Какой из них вам больше нравится?
8. Как заполнить массив случайными числами в диапазоне от 100 до 200?
9. С помощью каких функций можно найти сумму и количество элементов массива?
10. Сравните разные способы решения задачи о среднем росте игроков. Какой из них вам больше нравится? Обсудите этот вопрос в классе.

Темы сообщений

а) «Массивы в языке C»

б) «Массивы в языке Javascript»

Проекты

а) Игра «Змейка»

б) Модель «падающего снега»

§ 63
Алгоритмы обработки массивов

Ключевые слова:

- поиск элемента
- реверс массива
- сдвиг элементов
- отбор элементов по условию
- срез массива

Поиск в массиве

Требуется найти в массиве элемент, равный значению переменной X, или сообщить, что его там 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 = 0
while A[i] != X:
  i += 1
print ("A[", i, "]=", X, sep = "")
```

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 окончания цикла это условие нарушено, то поиск был неудачным — элемента нет:

```
i = 0
while i < N and A[i] != X:
  i += 1
if i < N:
  print ("A[", i, "]=", X, sep = "")
else:
  print ( "Не нашли!" )
```

Отметим одну тонкость. В сложном условии `i < N and A[i] != X` первым должно проверяться именно отношение `i < N`. Если первая часть условия, соединённого с помощью операции И, ложна, то вторая часть, как правило, не вычисляется[1] — уже понятно, что всё условие ложно. Дело в том, что если $i \ge N$, то проверка условия `A[i] != X` приводит к *выходу за границы массива*, и программа завершается аварийно.

[1] Во многих современных языках (например, в C, C++, Python, Javascript, PHP) такое поведение гарантировано стандартом.

Возможен ещё один поход к решению этой задачи: используя цикл с переменной, перебрать все элементы массива и досрочно завершить цикл, если найдено требуемое значение.

```
nX = -1
for i in range(len(A)):
  if A[i] == X:
    nX = i
    break
if nX >= 0:
  print("A[", nX, "]=", X, sep = "")
else:
  print("Не нашли!")
```

Для выхода из цикла используется оператор **break**, номер найденного элемента сохраняется в переменной nX. Если её значение осталось равным −1 (не изменилось в ходе выполнения цикла), то в массиве нет элемента, равного X.

Последний пример можно упростить, используя особые возможности цикла **for** в языке Python:

```
for i in range(len(A)):
  if A[i] == X:
    print("A[", i, "]=", X, sep = "")
    break
else:
  print("Не нашли!")
```

Здесь мы выводим результат сразу, как только нашли нужный элемент, а не после цикла. Слово **else** после цикла **for** начинает блок, который выполняется при нормальном завершении цикла (без применения **break**). Таким образом, сообщение «Не нашли!» будет выведено только тогда, когда условный оператор в теле цикла ни разу не сработал.

Возможен другой способ решения этой задачи, использующий метод index для типа данных list; этот метод возвращает номер первого найденного элемента, равного X:

```
nX = A.index(X)
```

Тут проблема только в том, что эта строка вызовет ошибку при выполнении программы, если нужного элемента в массиве нет. Поэтому нужно сначала проверить, есть ли он там (с помощью оператора **in**), а потом использовать метод index:

```
if X in A:
  nX = A.index(X)
  print ("A[", nX, "]=", X, sep = "")
else:
  print ("Не нашли!")
```

Запись `if X in A` означает «если значение `X` найдено в списке `A`».

Поиск максимального элемента

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`.

Остаётся решить, каково должно быть начальное значение `M`. Можно записать туда значение, заведомо меньшее, чем любой из элементов массива. Например, если в массиве записаны натуральные числа, можно записать в `M` ноль или отрицательное число. Если содержимое массива неизвестно, можно сразу записать в `M` значение `A[0]`, а цикл перебора начать со второго по счёту элемента, т. е. с `A[1]`:

```
M = A[0]
for i in range(1, N):
  if A[i] > M:
    M  = A[i]
print ( M )
```

Вот ещё один вариант:

```
M = A[0]
for x in A:
  if x > M:
    M = x
```

Он отличается тем, что мы не используем переменную-индекс, но зато дважды просматриваем элемент `A[0]` (второй раз — в цикле, где выполняется перебор *всех* элементов).

Поскольку операции поиска максимального и минимального элементов нужны очень часто, в Python есть соответствующие встроенные функции `max` и `min`:

```
Ma = max(A)
Mi = min(A)
```

Теперь предположим, что требуется найти не только значение, но и номер максимального элемента. Казалось бы, нужно ввести ещё одну переменную nMax для хранения номера, сначала записать в неё 0 (считаем элемент A[0] максимальным) и затем, когда будет найден новый максимальный элемент, запомнить его номер в переменной nMax[1]:

```
M = A[0]; nMax = 0
for i in range(1,N):
  if A[i] > M:
    M = A[i]
    nMax = i
print("A[", nMax, "]=", M, sep = "")
```

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 = 0
for i in range(1,N):
  if A[i] > A[nMax]:
    nMax = i
print("A[", nMax, "]=", A[nMax], sep = "")
```

Для решения этой задачи можно использовать встроенные функции Python: сначала найти максимальный элемент, а потом его индекс с помощью функции index:

```
M = max(A)
nMax = A.index(M)
print("A[", nMax, "]=", M, sep = "")
```

В этом случае фактически придётся выполнить два прохода по массиву. Однако такой вариант работает во много раз быстрее, чем «рукописный» цикл с одним проходом, потому что встроенные функции написаны на языке C++ и подключаются в виде готового машинного кода, а не выполняются относительно медленным интерпретатором Python.

---

[1] Обратите внимание, что можно записывать несколько операторов в одной строке, они отделяются друг от друга точкой с запятой.

Реверс массива

Реверс массива — это перестановка его элементов в обратном порядке: первый элемент становится последним, а последний — первым (рис. 8.17).

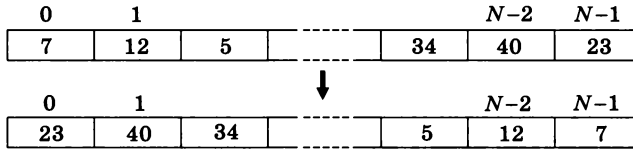


Рис. 8.17

Из рисунка следует, что 0-й элемент меняется местами с $(N-1)$-м, первый — с $(N-2)$-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:

```
for i in range(N):
  поменять местами A[i] и A[N-1-i]
```

Однако это неверно. Посмотрим, что получится для массива из четырёх элементов (рис. 8.18).

	0	1	2	3
	7	12	40	23
A[0]↔A[3]	23	12	40	7
A[1]↔A[2]	23	40	12	7
A[2]↔A[1]	23	12	40	7
A[3]↔A[0]	7	12	40	23

Рис. 8.18

Как видите, массив вернулся в исходное состояние: реверс выполнен дважды. Поэтому нужно остановить цикл на середине массива:

```
for i in range(N//2):
  поменять местами A[i] и A[N - 1 - i]
```

Для обмена можно использовать вспомогательную переменную c:

```
for i in range(N//2):
  c = A[i]
  A[i] = A[N-1-i]
  A[N-1-i] = c
```

или возможности Python:

```
for i in range(N//2):
  A[i], A[N-i-1] = A[N-i-1], A[i]
```

Эта операция может быть выполнена и с помощью стандартного метода `reverse` (в переводе с англ. — реверс, обратный) для объектов типа `list`:

```
A.reverse()
```

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[1]` переходит на место `A[0]`, `A[2]` — на место `A[1]` и т. д. (рис. 8.19).

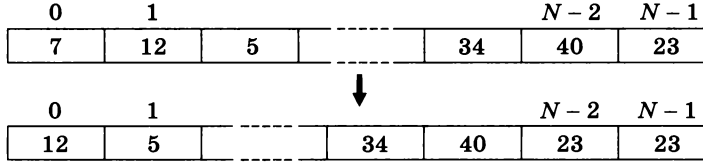


Рис. 8.19

Последний элемент остаётся на своем месте, поскольку новое значение для него взять неоткуда — массив кончился. Алгоритм выглядит так:

```
for i in range(N-1):
  A[i] = A[i+1]
```

Обратите внимание, что цикл заканчивается при `i = N - 2` (а не `N - 1`), чтобы не было выхода за границы массива, т. е. обращения к несуществующему элементу `A[N]`.

При таком сдвиге первый элемент пропадает, а последний дублируется. Можно старое значение первого элемента записать на место последнего. Такой сдвиг называется **циклическим** (см. § 26). Предварительно (до начала цикла) первый элемент нужно запомнить во вспомогательной переменной, а после завершения цикла записать его в последнюю ячейку массива:

```
c = A[0]
for i in range(N-1):
  A[i] = A[i+1]
A[N-1] = c
```

Срезы массива

Ещё проще выполнить сдвиг элементов, используя встроенные возможности списков в Python:

```
A = A[1:N] + [A[0]]
```

Здесь использован так называемый **срез** — выделение части массива. Срез `A[1:N]` означает «все элементы с `A[1]` до `A[N - 1]`», т. е. не включая элемент с индексом `N`. Таким образом, этот срез «складывается» со списком, состоящим из одного элемента `A[0]`, в результате получается новый массив, составленный из «списков-слагаемых».

Чтобы такая система (исключение последнего элемента) была более понятной, можно представить, что срез массива выполняется по *разрезам* — границам между элементами (рис. 8.20).

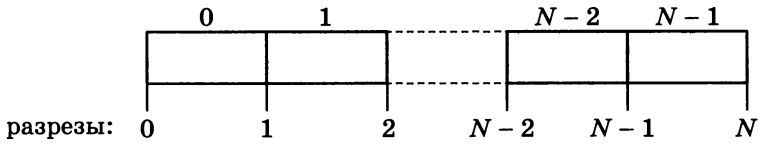


Рис. 8.20

Таким образом, срез `A[0:2]` выбирает все элементы между разрезами **0** и **2**, т. е. элементы `A[0]` и `A[1]`.

Если срез заканчивается на последнем элементе массива, второй индекс можно не указывать:

```
A = A[1:] + [A[0]]
```

Аналогично, `A[:5]` обозначает «первые 5 элементов массива» (начальный индекс не указан). Если не указать ни начальный, ни конечный индекс, мы получим копию массива:

```
Acopy = A[:]
```

Если использованы отрицательные индексы, к ним добавляется длина массива. Например, срез `A[:-1]` выбирает все элементы, кроме последнего (он равносилен `A[:N-1]`). А вот так можно выделить из массива три последних элемента:

```
B = A[-3:]
```

Заметим, что с помощью среза можно, например, выполнить реверс массива:

```
A = A[::-1]
```

Рассмотрим подробно правую часть оператора присваивания. Число –1 обозначает шаг выборки значений, т. е. элементы выбираются в обратном порядке. Слева от первого и второго знаков двоеточия записывают начальное и конечное значения индексов; если они не указаны, считается, что рассматривается весь массив.

Отбор нужных элементов

Требуется отобрать все элементы массива A, удовлетворяющие некоторому условию, в новый массив B. Поскольку списки в Python могут расширяться во время работы программы, можно использовать такой алгоритм: сначала создаём пустой массив, затем перебираем все элементы исходного массива и, если очередной элемент нам нужен, добавляем его в новый массив:

```
B = []
for x in A:
  if x % 2 == 0: # отбираем чётные  элементы
    B.append(x)
```

Здесь для добавления элемента в конец массива использован метод append.

Второй вариант решения — использование генератора списка с условием.

```
B = [x for x in A  if x % 2 == 0  ]
```

В цикле перебираются все элементы массива A, и только чётные из них включаются в новый массива.

Особенности копирования списков в Python

Как вы знаете, имя переменной в языке Python связывается с объектом в памяти: числом, списком и др. При выполнении операторов

```
A =[1, 2, 3]
B = A
```

две переменные A и B будут связаны с одним и тем же списком (рис. 8.21, *а*), поэтому при изменении одного списка будет изменяться и второй, ведь это фактически один и тот же список, к которому можно обращаться по двум разным именам. На рисунке 8.21, *б* показана ситуация после выполнения оператора A[0] = 0.

а *б*

Рис. 8.21

Эту особенность Python нужно учитывать при работе со списками. Если нам нужна именно копия списка (а не ещё одна ссылка на него), можно использовать срез, строящий копию:

```
B = A[:]
```

Теперь `A` и `B` — это независимые списки и изменение одного из них не меняет второй (рис. 8.22).

а *б*

Рис. 8.22

Вместо среза можно было использовать функцию `copy` из модуля `copy`:

```
import copy
A =[1,2,3]
B = copy.copy(A)
```

Это так называемая «поверхностная» копия — она не создаёт полную копию, если список содержит какие-то изменяемые объекты, например другой список. Для полного копирования используется функция `deepcopy` из того же модуля:

```
import copy
A =[1,2,3]
B = copy.deepcopy(A)
```

Выводы

- Поиск элемента в неотсортированном массиве сводится к просмотру всех его элементов.
- Для поиска максимума используют переменную, значение которой после каждого шага цикла содержит максимум из уже просмотренных элементов массива.

- Реверс массива — это перестановка элементов в обратном порядке.
- Для выделения части массива используют срезы.
- Для копирования списка применяют срез `A[:]` или функции модуля `copy`.

Нарисуйте в тетради интеллект-карту этого параграфа.

Вопросы и задания

1. Почему при поиске индекса максимального элемента не нужно хранить само значение максимального элемента?
2. Как вы думаете, какую ошибку чаще всего делают начинающие программисты, выполняя реверс массива без использования встроенной функции?
3. Как вы думаете, какие проблемы (и ошибки) могут возникнуть при циклическом сдвиге массива *вправо* (без использования срезов)?
4. Что произойдёт с массивом при выполнении следующего фрагмента программы:

```
for i in range(N-1):
  A[i+1] = A[i]
```

5. Как (при использовании приведённого алгоритма поиска) определить, что элемент не найден?
6. Что такое выход за границы массива? Почему он может быть опасен?
7. Сравните разные методы отбора части элементов одного массива в другой массив. Какой из них вам больше нравится? Почему?

§ 64
Сортировка

Ключевые слова:

- сортировка обменами
- метод выбора
- сортировка слиянием
- быстрая сортировка

Введение

Сортировка — это перестановка элементов массива в заданном порядке.

Порядок сортировки может быть любым; для чисел обычно рассматривают сортировку по возрастанию (или убыванию) значений. Для массивов, в которых есть одинаковые элементы, используются понятия «сортировка по неубыванию» и «сортировка по невозрастанию».

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 два метода из второй — сортировку слиянием и знаменитую «быструю сортировку», предложенную Ч. Хоаром.

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» (минимальный) элемент, 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. 8.23).

Рис. 8.23

Когда мы обработали пару (A[0], A[1]), минимальный элемент оказался на месте A[0]. Это значит, что на следующих этапах его можно не рассматривать. Первый цикл, устанавливающий на свое место первый (минимальный) элемент, можно на псевдокоде записать так:

```
для j от N-2 до 0 шаг -1
  if A[j+1]< A[j]:
    поменять местами A[j] и A[j+1]
```

Заголовок цикла тоже написан на псевдокоде. Обратите внимание, что на очередном шаге сравниваются элементы A[j] и A[j + 1], поэтому цикл начинается с j = N - 2. Если начать с j = N - 1, то на первом же шаге получаем выход за границы массива — обращение к элементу A[N].

Поскольку цикл **for in** range(…) в Python «не доходит» до заданного конечного значения, мы принимаем его равным –1, а не 0:

```
for j in range(N-2, -1 , -1):
  if A[j+1]< A[j]:
    поменять местами A[j] и A[j+1]
```

То есть в записи range(N-2, -1, -1) первое число –1 — это ограничитель (число, следующее за конечным значением), а второе число –1 — это шаг изменения переменной цикла.

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:

```
for j in range(N-2,  0 ,-1):
  if A[j+1] < A[j]:
    поменять местами A[j] и A[j+1]
```

При установке следующего (3-го по счёту) элемента конечное значение при вызове функции range будет равно 1 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 встаёт на своё место — другого места нет. Поэтому полный алгоритм сортировки представляет собой такой вложенный цикл:

```
for i in range(N-1):
  for j in range(N-2, i-1 ,-1):
    if A[j+1] < A[j]:
      A[j], A[j+1] = A[j+1], A[j]
```

Записать полную программу вы можете самостоятельно.

Часто используют более простой вариант этого метода, который можно назвать «методом камня» — самый «тяжёлый» элемент опускается в конец массива.

```
for i in range(N):
  for j in range(N-i-1):
    if A[j+1] < A[j]:
      A[j], A[j+1] = A[j+1], A[j]
```

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 на псевдокоде можно записать так:

```
for i in range(N-1):
  найти номер nMin минимального элемента
                        среди A[i]..A[N-1]
  if i != nMin:
    поменять местами A[i] и A[nMin]
```

Здесь перестановка происходит только тогда, когда найденный минимальный элемент стоит не на своём месте, т. е. `i != nMin`. Поскольку поиск номера минимального элемента выполняется в цикле, этот алгоритм сортировки также представляет собой вложенный цикл:

```
for i in range(N-1):
  nMin = i
  for j in range(i+1,N):
    if A[j] < A[nMin]:
      nMin = j
  if i!= nMin:
     A[i], A[nMin] = A[nMin], A[i]
```

Сортировка слиянием

Методы сортировки, описанные ранее, работают медленно для больших массивов данных (более 1000 элементов). Поэтому в середине XX века математики и программисты серьёзно занимались разработкой более эффективных алгоритмов сортировки.

В 1945 году Джон фон Нейман предложил алгоритм, основанный на процедуре слияния двух заранее отсортированных массивов. Предположим, что есть два отсортированных массива A и B, размеры которых соответственно N_a и N_b, и мы хотим объединить их элементы в один отсортированный массив C размером $N_a + N_b$. Для этого будем на каждом шаге сравнивать два первых (ещё не использованных) элемента массивов A и B, добавляя в конец массива C наименьший из них. Так продолжается до тех

пор, пока один из массивов не закончится, тогда оставшийся «хвост» второго массива просто добавляется в конец массива *C* (он ведь уже был отсортирован!). Пример такого слияния показан на рис. 8.24.

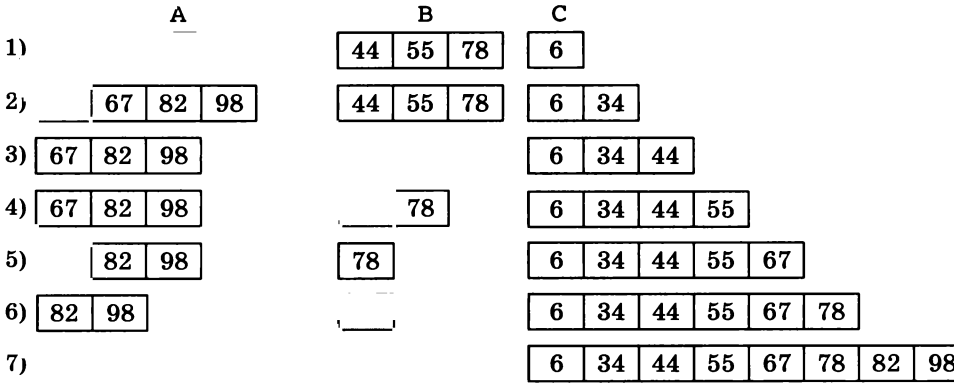


Рис. 8.24

Фоном выделены элементы, которые добавляются в массив C на очередном шаге. Шаги 1–6 — это добавление в массив C наименьшего из первых элементов массивов A и B, а шаг 7 — дописывание в конец массива C «хвоста» массива A (весь массив B уже добавлен в массив C).

Первая часть процедуры слияния, во время которой мы проверяем первые неиспользованные элементы обоих массивов, на языке Python может быть записана так:

```
C = []
iA = iB = 0
while iA < Na and iB < Nb:
  if A[iA] <= B[iB]:
    C.append(A[iA])
    iA += 1
  else:
    C.append(B[iB])
    iB += 1
```

Сначала массив C пуст. На каждом шаге цикла с помощью метода append в конец этого массива добавляется очередной элемент массива A или очередной элемент массива B (в зависимости от того, который из них меньше).

Теперь остаётся добавить в массив C оставшуюся часть массива A:

```
C = C + A[iA:]
```

и аналогично оставшуюся часть массива B.

На основе этой идеи был разработан быстрый алгоритм **сортировки слиянием** (англ. *merge sort*). Напишем процедуру mergeSort, которая сортирует переданный ей массив A по возрастанию. Разделим массив на две равные части, вычислим средний индекс:

```
mid = len(A) // 2
```

Теперь отсортируем отдельно первую и вторую половину:

```
L = A[:mid]; mergeSort(L)
R = A[mid:]; mergeSort(R)
```

и выполним слияние этих половин, вызвав функцию merge (мы скоро напишем её полностью):

```
C = merge(L, R)
```

Эта процедура построит новый массив C, той же длины, что и массив A. Остаётся только скопировать эти данные обратно в массив A. Получается такая процедура сортировки:

```
def mergeSort(A):
  if len(A) <= 1: return
  mid = len(A) // 2
  L = A[:mid]; mergeSort(L)
  R = A[mid:]; mergeSort(R)
  C = merge(L, R)
  for i in range(len(A)):
    A[i] = C[i]
```

Процедура mergeSort дважды вызывает сама себя, т. е. является рекурсивной. Строка

```
if len(A) <= 1: return
```

останавливает рекурсию: если остался всего один элемент, сортировать нечего и нужно выйти из процедуры.

Осталось написать функцию merge, которая сливает две отсортированные части. Она фактически повторяет алгоритм слияния, приведённый выше:

```
def merge(A, B):
  iA = iB = 0
```

```
Na = len(A); Nb = len(B)
C = []
while iA < Na and iB < Nb:
  if A[iA] <= B[iB]:
    C.append(A[iA]); iA += 1
  else:
    C.append(B[iB]); iB += 1
C = C + A[iA:] + B[iB:]
return C
```

Сортировка слиянием — это пример применения подхода «разделяй и властвуй» (англ. *divide and conquer*):

1) задача разбивается на несколько подзадач меньшего размера;
2) эти подзадачи решаются с помощью рекурсивных вызовов того же (или другого) алгоритма;
3) решения подзадач объединяются, и получается решение исходной задачи.

Сортировка слиянием применима для данных с последовательным доступом, например записанных на магнитные ленты. Этот алгоритм можно использовать в системах с параллельными процессорами с общей памятью, которые работают одновременно и делят работу между собой. Его главный недостаток состоит в том, что нужен дополнительный массив того же размера, что и исходный (в нашей программе это массив `C`).

«Быстрая сортировка»

Один из самых популярных «быстрых» алгоритмов, разработанный в 1960 году английским учёным Ч. Хоаром, так и называется — «быстрая сортировка» (англ. *quicksort*).

Ч. Э. Хоар (р. 1934)

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 (назовём его X). На первом этапе мы расставим элементы так, что слева от некоторой границы (в первой группе) находятся все числа, меньшие или равные X, а справа (во второй группе) — большие или равные X. Заметим, что элементы, равные X, могут находиться в обеих частях.

A[i] ≤ X	A[i] ≥ X

Теперь элементы расположены так, что ни один элемент из первой группы при сортировке не окажется во второй и наоборот. Поэтому далее достаточно отсортировать *отдельно* каждую часть массива. Мы опять применили подход «разделяй и властвуй».

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[1)], т. е. заранее решить ту самую задачу, которую мы собираемся решить этим способом. Поэтому обычно в качестве X выбирают средний элемент массива или элемент со случайным номером.

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, чем 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 две части: левую со значениями, меньшими или равными X, и правую со значениями, бóльшими 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 один элемент (и таким образом, массив будет отсортирован).

Чтобы понять сущность метода, рассмотрим пример. Пусть задан массив

78	6	82	67	55	44	34

X

Выберем в качестве X средний элемент массива, т. е. 67. Найдём первый слева элемент массива, который больше или равен

1) Хотя есть и другие, тоже достаточно непростые алгоритмы.

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, т. е. массив разбит на две части. В результате все элементы массива, расположенные левее A[L], меньше или равны X, а все элементы правее A[R] — больше или равны X.

Таким образом, сортировка исходного массива свелась к двум сортировкам частей массива, т. е. к двум задачам того же типа, но меньшего размера. Теперь нужно применить тот же алгоритм к двум полученным частям массива: первая часть — с 1-го до R-го элемента, вторая часть — с L-го до последнего элемента. Как вы знаете, такой приём называется *рекурсией*.

Процедура сортировки принимает три параметра: сам массив (список) и значения индексов, 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 Python:

```
def qSort (A, nStart, nEnd):
  if nStart >= nEnd: return
  L = nStart; R = nEnd
  X = A[(L+R)//2]
  while L <= R:
    while A[L] < X: L += 1  # разделение
    while A[R] > X: R -= 1
    if L <= R:
      A[L], A[R] = A[R], A[L]
      L += 1; R -= 1
  qSort (A, nStart, R)  # рекурсивные вызовы
  qSort (A, L, nEnd)
```

Для того чтобы отсортировать весь массив, нужно вызвать эту процедуру так:

```
qSort (A, 0, N-1)
```

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[randint(L,R)]
```

Напомним, что функцию randint нужно импортировать из модуля random.

Эта процедура сортирует массив «на месте», без создания нового списка. Можно также оформить алгоритм в виде функции, которая возвращает новый список, не затрагивая существующий:

```
import random
def qSort ( A ):
  if len(A) <= 1: return A                    # (1)
  X = random.choice(A)                        # (2)
  B1 = [ b for b in A  if b < X ]             # (3)
  BX = [ b for b in A  if b == X ]            # (4)
  B2 = [ b for b in A  if b > X ]             # (5)
  return qSort(B1)+BX+qSort(B2)               # (6)
```

Дадим некоторые пояснения к этой функции. Если длина массива не больше 1, то ответ — это сам массив, сортировать тут нечего (строка 1). Иначе выбираем в качестве разделителя X («стержневого элемента», англ. *pivot*) случайный элемент массива (строка 2), для этого используется функция choice (в переводе с англ. — выбор) из модуля random.

В строках 3–5 с помощью генераторов списков создаём три вспомогательных массива: в массив B1 войдут все элементы, меньшие X, в массив BX — все элементы, равные X, а в массив B2 — все элементы, бóльшие X. Теперь остается отсортировать массивы B1 и B2 (вызвав функцию qSort рекурсивно) и построить окончательный массив-результат, который «складывается» из отсортированного массива B1, массива BX и отсортированного массива B2.

Для того чтобы построить отсортированный массив, нужно вызвать эту функцию так:

```
Asorted = qSort(A)
```

В таблице 8.2 сравнивается время сортировки (в секундах) массивов разного размера, заполненных случайными значениями, с использованием изученных алгоритмов.

Таблица 8.2

N	Метод пузырька	Метод выбора	Сортировка слиянием	Быстрая сортировка
1000	0,08 с	0,05 с	0,006 с	0,002 с
5000	1,8 с	1,3 с	0,033 с	0,006 с
15000	17,3 с	11,2 с	0,108 с	0,019 с

Как показывают эти данные, преимущество быстрой сортировки становится подавляющим при увеличении N.

Сортировка в языке Python

В Python есть встроенная функция для сортировки массивов (списков), которая называется `sorted`. Она использует алгоритм *Timsort*[1]. Вот так можно построить новый массив `B`, который совпадает с отсортированным в порядке возрастания массивом `A`:

```
B = sorted (A)
```

Для того чтобы отсортировать массив по убыванию (невозрастанию), нужно задать дополнительный параметр `reverse` (от англ. «обратный»), равный `True`:

```
B = sorted (A, reverse = True)
```

Иногда требуется особый, нестандартный порядок сортировки, который отличается от сортировок по возрастанию и по убыванию. В этом случае используют ещё один именованный параметр функции `sorted`, который называется `key` (ключ). В этот параметр нужно записать название функции (фактически — ссылку на объект-функцию), которая возвращает число (или символ), используемое при сравнении двух элементов массива.

Предположим, что нам нужно отсортировать числа по возрастанию последней цифры (поставить сначала все числа, оканчивающиеся на 0, затем — все, оканчивающиеся на 1 и т. д.). В этом случае ключ сортировки — это последняя цифра числа, поэтому напишем функцию, которая выделяет эту последнюю цифру:

```
def lastDigit (n):
  return n % 10
```

Тогда сортировка массива `A` по возрастанию последней цифры запишется так:

```
B = sorted (A, key = lastDigit)
```

Функция `lastDigit` получилась очень короткой, и часто не хочется создавать лишнюю функцию с помощью оператора **def**, особенно тогда, когда она нужна всего один раз. В таких случаях удобно использовать функции без имени — так называемые *лямбда-функции*, которые записываются прямо при вызове функции в параметре `key`:

```
B = sorted (A, key =  lambda x: x % 10  )
```

---

[1] **http://ru.wikipedia.org/wiki/Timsort**

Фоном выделена лямбда-функция, которая для переданного ей значения `x` возвращает результат `x % 10`, т. е. последнюю цифру десятичной записи числа.

Функция `sorted` не изменяет исходный массив и возвращает его отсортированную копию. Если нужно отсортировать массив «на месте», лучше использовать метод `sort`, который определён для списков:

```
A.sort(key = lambda x: x % 10, reverse = True)
```

Он имеет те же именованные параметры, что и функция `sorted`, но изменяет исходный массив. В приведённом примере элементы массива `A` будут отсортированы по убыванию последней цифры.

Выводы

- Сортировка — это расстановка элементов массива в заданном порядке.
- Цель сортировки — облегчить последующий поиск.
- Методы сортировки с помощью обменов (метод пузырька) и выбора минимального элемента работают медленно для больших массивов данных.
- Метод сортировки слиянием и быстрая сортировка позволяют быстро сортировать массивы большого размера.

Нарисуйте в тетради интеллект-карту этого параграфа.

Вопросы и задания

1. На какой идее основан метод пузырька? Метод выбора?
2. Объясните, зачем нужен вложенный цикл в описанных методах сортировки.
3. Сравните метод пузырька и метод выбора. Какой из них требует меньше перестановок?
4. Расскажите про основные идеи метода сортировки слиянием.
5. Как вы думаете, можно ли использовать метод «быстрой сортировки» для нечисловых данных, например для символьных строк?
6. От чего зависит скорость «быстрой сортировки»? Какой самый лучший и самый худший случай?
7. Как вы думаете, может ли метод «быстрой сортировки» работать дольше, чем метод выбора (или другой «простой» метод)? Если да, то при каких условиях?

8. Как нужно изменить приведённые в параграфе алгоритмы, чтобы элементы массива были отсортированы по убыванию?
9. Какие встроенные средства сортировки массивов в Python вы знаете?
10. Как задать нестандартный порядок сортировки для функции `sorted`?
11. Что такое лямбда-функции? Когда их удобно использовать?
12. Чем отличаются функция `sorted` и метод `sort` для списков?
13. Используя информацию из дополнительных источников и модуль `time`, попробуйте построить зависимость времени выполнения сортировки от размера массива для разных методов. Какие зависимости у вас получились?

Проекты

а) Сравнение скорости работы алгоритмов сортировки
б) Сортировка слиянием для данных в файле
в) Экспериментальное определение скорости работы алгоритмов сортировки

§ 65
Двоичный поиск

Ключевые слова:

- 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 предвари-

тельно отсортированы. Для примера на рисунке показан поиск числа X = 44 в отсортированном массиве (рис. 8.25).

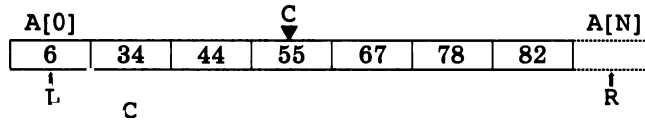


Рис. 8.25

Серым фоном выделены ячейки, которые уже не рассматриваются, потому что в них не может быть заданного числа. Переменные 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 c. 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.

```
L = 0; R = N                # начальный отрезок
while L < R-1:
  c = (L+R) // 2            # нашли середину
  if X < A[c]:              # сжатие отрезка
        R = c
  else: L = c
if A[L] == X:
      print("A[", L, "]=", X, sep = "")
else: print("Не нашли!")
```

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 8.3 сравнивается количество шагов для двух методов при разных значениях N.

Таблица 8.3

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.

Выводы

- Для отсортированного массива можно применить двоичный поиск, который работает значительно быстрее, чем линейный.
- При двоичном поиске на каждом шаге зона поиска уменьшается в два раза.

Нарисуйте в тетради интеллект-карту этого параграфа.

Вопросы и задания

1. Почему этот алгоритм поиска называется двоичным?
2. Приведите примеры использования двоичного поиска в обычной жизни.
3. Как можно примерно подсчитать количество шагов при двоичном поиске?
4. Сравните достоинства и недостатки линейного и двоичного поиска.

Проект

Экспериментальное определение скорости работы двоичного поиска

§ 66
Символьные строки

Ключевые слова:

- символьная строка
- длина строки
- подстрока
- срез строки
- поиск подстроки
- замена
- рекурсивный перебор
- сравнение строк
- сортировка строк

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, символьные массивы и символьные строки (которые, в отличие от массивов, рассматриваются как цельные объекты).

Основной тип данных для работы с символьными величинами в языке Python — это **символьные строки** (тип `str`).

Для того чтобы записать в строку значение, используют оператор присваивания

```
s = "Вася пошёл гулять"
```

Строка заключается в кавычки или в апострофы. Если строка ограничена кавычками, внутри неё могут быть апострофы, и наоборот.

Для ввода строки с клавиатуры применяют функцию `input`:

```
s = input("Введите имя: ")
```

Длина строки определяется с помощью функции `len` (от англ. *length* — длина). В этом примере в переменную `n` записывается длина строки `s`:

```
n = len(s)
```

Для того чтобы выделить отдельный символ строки, к нему нужно обращаться так же, как к элементам массива (списка): в квадратных скобках записывают номер символа. Например, так

можно вывести на экран символ строки s с индексом 5 (длина строки в этом случае должна быть не менее 6 символов):

```
print(s[5])
```

Отрицательный индекс говорит о том, что отсчёт ведётся с конца строки. Например, s[-1] означает то же самое, что s[len(s)-1], т. е. последний символ строки.

В отличие от большинства современных языков программирования в Python строка — это неизменяемый объект. Поэтому оператор присваивания s[5] = "a" не сработает — будет выдано сообщение об ошибке.

Тем не менее можно составить из символов существующей строки новую строку и при этом внести нужные изменения. Приведём полную программу, которая вводит строку с клавиатуры, заменяет в ней все буквы «а» на буквы «б» и выводит полученную строку на экран.

```
s = input("Введите строку:")
s1 = ""
for c in s:
  if c == "а": c = "б"
  s1 = s1 + c
print(s1)
```

Здесь в цикле **for** c **in** s перебираются все символы, входящие в строку s. Каждый из них на очередном шаге записывается в переменную c. Затем мы проверяем значение этой переменной: если оно совпадает с буквой «а», то заменяем его на букву «б», и прицепляем в конец новой строки s1 с помощью оператора сложения.

Нужно отметить, что показанный здесь способ (многократное «сложение» строк) работает очень медленно. В практических задачах, где требуется замена символов, лучше использовать стандартную функцию replace, о которой пойдёт речь дальше.

Операции со строками

Как мы уже видели, оператор + используется для **объединения (сцепления) строк** (англ. *concatenation*), эта операция иногда называется **конкатенация** (от лат. *catena* — цепь). Например:

```
s1 = "Привет"
s2 = "Вася"
s  = s1 + ", " + s2 + "!"
```

В результате выполнения приведённой программы в строку s будет записано значение

"Привет, Вася!".

Для того чтобы выделить часть строки (**подстроку**), в языке Python применяется операция получения **среза** (англ. *slicing*), например s[3:8] означает символы строки s с 3-го по 7-й (т. е. до 8-го, не включая его). Следующий фрагмент копирует в строку s1 символы строки s с 3-го по 7-й (всего 5 символов):

```
s = "0123456789"
s1 = s[3:8]
```

В строку s1 будет записано значение "34567".

Для удаления части строки нужно составить новую строку, объединив части исходной строки до и после удаляемого участка:

```
s = "0123456789"
s1 = s[:3] +  s[9:]
```

Срез s[:3] означает «от начала строки до символа s[3], не включая его», а запись s[9:] — «все символы, начиная с s[9] до конца строки». Таким образом, в переменной s1 остаётся значение "0129".

С помощью срезов можно вставить новый фрагмент внутрь строки:

```
s = "0123456789"
s1 = s[:3] + "ABC" + s[3:]
```

Переменная s получит значение "012ABC3456789".

Если заданы отрицательные индексы, к ним добавляется длина строки N. Таким образом, индекс –1 означает то же самое, что N – 1. При выполнении команд

```
s = "0123456789"
s1 = s[:-1]          # "012345678"
s2 = s[-6:-2]        # "4567"
```

мы получим s1 = "012345678" (строка s кроме последнего символа) и s2 = "4567".

Срезы позволяют выполнить реверс строки (развернуть её наоборот):

```
s1 = s[::-1]
```

Так как начальный и конечный индексы элементов строки не указаны, задействована вся строка. Число –1 означает шаг изме-

нения индекса и говорит о том, что все символы перебираются в обратном порядке.

В Python существует множество встроенных методов для работы с символьными строками. Например, методы `upper` и `lower` позволяют перевести строку соответственно в верхний и нижний регистры:

```
s = "aAbBcC"
s1 = s.upper()  # "AABBCC"
s2 = s.lower()  # "aabbcc"
```

а метод `isdigit` проверяет, все ли символы строки — цифры, и возвращает логическое значение:

```
s = "abc"
print(s.isdigit())   # False
s1 = "123"
print(s1.isdigit())  # True
```

О других встроенных функциях вы можете узнать в литературе или в Интернете.

Поиск в строках

В Python существует функция для **поиска подстроки** (и отдельного символа) в символьной строке. Эта функция называется `find`, она определена для символьных строк и вызывается как метод, с помощью точечной записи. В скобках нужно указать образец для поиска:

```
s = "Здесь был Вася."
n = s.find("с")           # n = 3
if n >= 0:
  print("Номер символа", n)
else:
  print("Символ не найден.")
```

Метод `find` 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 возвращает –1. В рассмотренном примере в переменную `n` будет записано число 3.

Аналогичный метод `rfind` (от англ. *reverse find* — искать в обратную сторону) ищет последнее вхождение образца в строке. Для той же строки `s`, что и в предыдущем примере, метод `rfind` вернёт 12 (индекс последней буквы «с»):

```
s = "Здесь был Вася."
n = s.rfind("с")          # n = 12
```

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
найти в строке s первый пробел
имя = всё, что слева от первого пробела
удалить из строки s имя с пробелом
найти в строке s первый пробел
отчество = всё, что слева от первого пробела
удалить из строки s отчество с пробелом
                                  # осталась фамилия
s = s + " " + имя[0] + "." + отчество[0] +  "."
```

Мы последовательно выделяем из строки три элемента: имя, отчество и фамилию, используя тот факт, что они разделены одиночными пробелами. После того как имя будет сохранено 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 срезы и метод `find`. Приведём сразу полную программу:

```
s = input("Введите имя, отчество и фамилию:")
n = s.find(" ")
name = s[:n]          # вырезать имя
s = s[n+1:]
n = s.find(" ")
name2 = s[:n]         # вырезать отчество
s = s[n+1:]           # осталась фамилия
s = s + " " +  name[0] + "." + name2[0] + "."
print(s)
```

Используя встроенные функции языка Python, эту задачу можно решить намного проще. Прежде всего нужно разбить введённую строку на отдельные слова, которые разделены пробелами. Для этого используется метод `split` (от англ. *split* — расщепить), который возвращает список слов, полученных при разбиении строки:

```
fio = s.split()
```

Если входная строка правильная (соответствует формату, описанному в условии), то в этом списке будут три элемента: `fio[0]` — имя, `fio[1]` — отчество и `fio[2]` — фамилия. Теперь остаётся только собрать строку-результат в нужном виде:

```
s = fio[2] + " " +  fio[0][0] + "." + fio[1][0] + "."
```

Запись `fio[0][0]` обозначает «0-й символ из 0-го элемента списка `fio`», т. е. первая буква имени. Аналогично `fio[1][0]` — это первая буква отчества. Вот полная программа:

```
s = input("Введите имя, отчество и фамилию:")
fio = s.split()
s = fio[2] + " " + fio[0][0] + "." + fio[1][0] + "."
print(s)
```

Преобразования «число ↔ строка»

В практических задачах часто нужно преобразовать число, записанное в виде цепочки символов, в числовое значение, и наоборот. Для этого в языке Python есть стандартные функции:

`int` — переводит строку в целое число;

`float` — переводит строку в вещественное число;

`str` — переводит целое или вещественное число в строку.

Приведём пример преобразования строк в числовые значения:

```
s = "123"
N = int(s)          # N = 123
s = "123.456"
X = float(s)        # X = 123.456
```

Если строку не удалось преобразовать в число (например, если в ней содержатся буквы), возникает ошибка, и выполнение программы завершается.

Теперь покажем примеры обратного преобразования:

```
N = 123
s = str(N)    # s = "123"
X = 123.456
s = str(X)    # s = "123.456"
```

Эти операции всегда выполняются успешно (ошибки быть не может).

Функция `str` использует правила форматирования, установленные по умолчанию. При необходимости можно использовать собственное форматирование. Например, в строке

```
s = "{:5d}".format(N)
```

значение переменной `N` будет записано по формату `d` (целое число) в 5 позициях, т. е. в начале строки будут стоять два пробела:

```
◦◦123
```

Для вещественных чисел можно использовать форматы `f` (с фиксированной точкой) и `e` (экспоненциальный формат, с плавающей точкой):

```
X = 123.456
s = "{:7.2f}".format(X)    # s = "◦123.46"
s = "{:10.2e}".format(X)   # s = "◦◦1.23e+02"
```

В первом случае формат `7.2f` обозначает «вывести в 7 позициях с 2 знаками в дробной части», во втором — формат `10.2e` обозначает «вывести научном формате в 10 позициях с 2 знаками в дробной части».

Строки в процедурах и функциях

Строки можно передавать в процедуры и функции как параметры, 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. В первую очередь в голову приходит такой псевдокод

```
while слово wOld есть в строке s:
  удалить слово wOld из строки
  вставить на это место слово wNew
```

Однако такой алгоритм работает неверно, если слово wOld входит в состав wNew, например нужно заменить "12" на "A12B" (покажите самостоятельно, что это приведёт 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. 8.26, *а*).

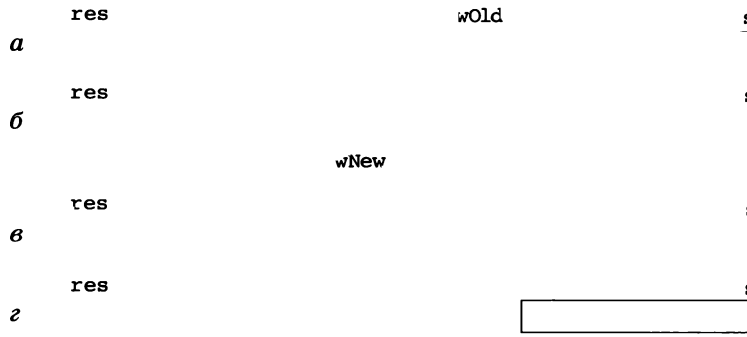


Рис. 8.26

Теперь нужно выполнить следующие действия:

1) ту часть строки s, которая стоит слева от образца, «прицепить» в конец строки res (рис. 8.26, *б*);
2) «прицепить» в конец строки res слово-замену wNew (рис. 8.26, *в*);
3) удалить из строки s начальную часть, включая найденное слово-образец (рис. 8.26, *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 найти не удалось, вся оставшаяся строка s приписывается в конец строки-результата, и цикл заканчивается.

В начале работы алгоритмы в строку res записывается пустая строка "", не содержащая ни одного символа. В таблице 8.4 приведён протокол работы алгоритма замены для строки "12.12.12", в которой нужно заменить слово "12" на "A12B".

Таблица 8.4

Рабочая строка s	Результат res
"12.12.12"	""
".12.12"	"A12B"
".12"	"A12B.A12B"
""	"A12B.A12B.A12B"

Теперь можно написать функцию на языке Python. Её параметры — это исходная строка s, строка-образец wOld и строка-замена wNew:

```
def replaceAll (s, wOld, wNew):
  lenOld = len(wOld)
  res = ""
  while len(s) > 0:
    p = s.find (wOld)
    if p < 0: return res + s
    if p > 0: res = res + s[:p]
    res = res + wNew
    if p+lenOld >= len(s):
      s = ""
    else:
      s = s[p+lenOld:]
  return res
```

Переменная p — это номер первого символа первого найденного слова-образца wOld, а в переменной lenOld записана длина этого слова. Если после поиска слова значение p меньше нуля (образец не найден), происходит выход из цикла:

```
if p < 0: return res + s
```

Если p > 0, то слева от образца есть какие-то символы, и их нужно «прицепить» к строке res:

```
if p > 0: res = res + s[:p]
```

Условие p+lenOld >= len(s) означает «образец стоит в самом конце слова», при этом остаток строки s — пустая строка. Функция возвращает изменённую строку.

Приведём пример использования этой функции:

```
s = "12.12.12"
s = replaceAll(s, "12", "A12B")
print(s)
```

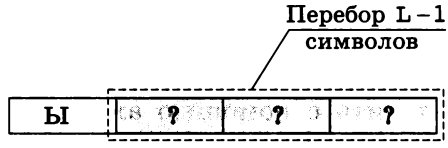
Так как операция замены одной подстроки на другую используется очень часто, в языке Python есть встроенная функция, которая выполняет эту операцию. Она оформлена как метод для переменных типа «строка» (str) и вызывается с помощью точечной записи:

```
s = "12.12.12"
s = s.replace("12", "A12B")
print(s)
```

Рекурсивный перебор

В алфавите языке племени «тумба-юмба» четыре буквы: «Ы», «Ш», «Ч» и «О». Нужно вывести на экран все слова, состоящие из L 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 L - 1 позициях:



Далее поочерёдно ставим на первое место все остальные буквы, повторяя процедуру:

```
def перебор L символов:
  w = "Ы"; перебор последних L-1 символов
  w = "Ш"; перебор последних L-1 символов
  w = "Ч"; перебор последних L-1 символов
  w = "О"; перебор последних L-1 символов
```

Здесь через w обозначена символьная строка, в которой собирается рабочее слово. Таким образом, задача для слов длины L

свелась к четырём задачам для слов длины L - 1. Как вы знаете, такой приём называется **рекурсией**, а процедура — **рекурсивной**.

Когда рекурсия должна закончиться? Тогда, когда все символы будут расставлены, т. е. длина строки w станет равной L. При этом нужно вывести получившееся слово на экран и выйти из процедуры.

Подсчитаем количество всех возможных слов длины L. 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, слов из L букв может быть 4^L.

Процедура для перебора слов может быть записана так:

```
def TumbaWords(A, w, L):
  if len(w) == L:
    print (w)
    return
  for c in A:
    TumbaWords(A, w + c, L)
```

В параметре A передаётся алфавит языка, параметр w — это уже построенная часть слова, а L — нужная длина слова. Если длина построенного слова равна заданной длине L, это слово выводится на экран и происходит выход из процедуры (окончание рекурсии). Если слово не достроено, в цикле перебираем все символы, входящие в алфавит, по очереди добавляем каждый из них в конец строки и вызываем процедуру рекурсивно.

В основной программе остаётся вызвать эту процедуру с нужными исходными данными:

```
TumbaWords("ЫШЧО", "", 3)
```

При таком вызове будут построены все трёхбуквенные слова, которые можно получить с помощью алфавита «ЫШЧО».

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 идёт раньше в **алфавитном порядке**. Например, слово «паровоз» будет «меньше», чем слово «пароход»: они различаются 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 и строчные, и прописные 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 прописная, а во втором — такая же, но строчная. В таблице символов прописные буквы стоят раньше строчных и имеют меньшие коды. Поэтому "А" < "а", "П" < "п" 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; прописные 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: ввести с клавиатуры несколько слов (например, фамилий) и вывести их на экран в алфавитном порядке.

Для решения этой задачи с помощью языка Python удобно записать строки в массив (список) и затем отсортировать с помощью метода sort:

```
aS = []
print("Введите строки для сортировки:")
while True:
  s1 = input()
  if s1 == "": break
  aS.append(s1)
aS.sort()
print(aS)
```

Строки заносятся в список aS. Сначала этот список пустой, затем в цикле мы вводим очередную строку с клавиатуры и записываем её в переменную s1. Ввод заканчивается, когда введена пустая строка, т. е. вместо ввода строки пользователь нажал клавишу *Enter*. В этом случае сработает условный оператор и выполнится оператор **break**, прерывающий цикл.

Выводы

- Символьная строка — это последовательность символов, расположенных в памяти рядом (в соседних ячейках). Строка в языке Python — это неизменяемый объект.
- Для объединения строк используется оператор +.
- Для работы с частями строк применяют срезы (так же, как и для массивов).
- Метод `find` служит для поиска подстроки в строке. Он возвращает −1, если образец не найден.
- При сравнении строк используется таблица кодов символов.

Нарисуйте в тетради интеллект-карту этого параграфа.

Вопросы и задания

1. Как задать значение для символьной строки? Рассмотрите разные способы.
2. Почему нельзя сразу записать новое значение в заданную позицию строки? Как можно решить эту задачу?
3. Что обозначает оператор + применительно к строкам?
4. Перечислите основные операции со строками и приведите примеры их использования.
5. Как определить, что при поиске в строке образец не найден?
6. Как преобразовать число из символьного вида в числовой и обратно?
7. Почему строку не всегда можно преобразовать в число?
8. Объясните выражение «рекурсивный перебор».
9. Сравните рекурсивные и нерекурсивные методы решения переборных задач.

Проекты

а) Библиотека функций для обработки имён файлов
б) Интерпретатор языка управления исполнителем

§ 67
Матрицы

Ключевые слова:

- матрица
- строка
- столбец
- главная диагональ
- побочная диагональ

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. 8.27).

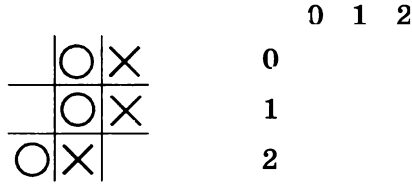


Рис. 8.27

Такие таблицы называются **матрицами** или **двумерными массивами**[1]. Каждый элемент матрицы, в отличие от обычного (линейного) массива, имеет два индекса — номер строки и номер столбца. На рисунке 8.27 серым фоном выделен элемент, находящийся на пересечении 1-й строки и 2-го 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.

Поскольку в Python нет массивов, нет и матриц в классическом понимании. Для того чтобы работать с таблицами, используют **списки**. Двумерная таблица хранится как список, каждый элемент которого тоже представляет собой список («список списков»). Например, таблицу на рис. 8.27, можно записать так:

```
A = [[-1, 0, 1],
     [-1, 0, 1],
     [0, 1, -1]]
```

или в одну строку:

```
A = [[-1, 0, 1], [-1, 0, 1], [0, 1, -1]]
```

Первый способ более нагляден, если мы задаём начальные значения для матрицы.

---

[1] Иногда используют и **многомерные массивы**, в которых каждый элемент имеет больше двух индексов. В практических задачах они нужны довольно редко, поэтому мы не будем их рассматривать.

Простейший способ вывода матрицы — с помощью одного вызова функции print:

```
print(A)
```

В этом случае матрица выводится в одну строку, что не очень удобно. Поскольку человек воспринимает матрицу как таблицу, лучше и на экран выводить её в виде таблицы. Для этого можно написать такую процедуру (в классическом стиле):

```
def printMatrix (A):
  for i in range (len(A)):
    for j in range (len(A[i])):
      print("{:4d}".format(A[i][j]), end = "")
    print()
```

Здесь i — это номер строки, а j — номер столбца; len(A) — это число строк в матрице, а len(A[i]) — число элементов в строке i (совпадает с числом столбцов, если матрица прямоугольная). Формат вывода 4d означает «вывести целое число в 4 позициях», после вывода очередного числа не делаем перевод строки (end = ""), а после вывода всей строки — делаем (print()).

Можно написать такую функцию в стиле Python:

```
def printMatrix (A):
  for row in A:
    for x in row:
      print("{:4d}".format(x), end = "")
    print()
```

Здесь первый цикл перебирает все строки в матрице; каждая из них по очереди попадает в переменную row. Затем внутренний цикл перебирает все элементы этой строки и выводит их на экран.

Иногда нужно создать в памяти матрицу заданного размера, заполненную некоторыми начальными значениями, например нулями[1]. Первая мысль — использовать такой алгоритм с операцией повторения *:

```
N = 3
M = 2
# неверное создание матрицы!
row = [0]*M    # создаём список - строку длиной M
A = [row]*N    # создаём массив (список) из N строк
```

[1] Для работы с матрицами и вообще для сложных вычислительных задач лучше использовать расширение *NumPy* (**www.numpy.org**), обсуждение которого выходит за рамки учебника.

Однако этот способ работает неверно из-за особенностей языка Python. Например, если после этого выполнить присваивание

```
A[0][0] = 1
```

то мы увидим, что *все* элементы столбца 0, т. е. `A[0, 0]`, `A[1, 0]` и т. д. стали равны 1. Дело в том, что матрица — это список ссылок на списки-строки (список адресов строк). При выполнении оператора

```
A = [row]*N
```

интерпретатор устанавливает все ссылки на одну-единственную строку (рис. 8.28).

Рис. 8.28

Естественно, что, когда мы меняем элемент 0 в этой строке (он выделен серым фоном на рисунке), меняются и все элементы с номером 0 в каждой строке матрицы.

Для создания полноценной матрицы нам нужно как-то заставить транслятор создать все строки в памяти как разные объекты. Для этого сначала создадим пустой список, а потом будем в цикле добавлять к нему (с помощью метода `append`) новые строки, состоящие из нулей:

```
A = []
for i in range(N):
  A.append([0]*M)
```

или так, с помощью генератора:

```
A = [[0]*M for i in range(N)]
```

Теперь все строки расположены в разных областях памяти (рис. 8.29).

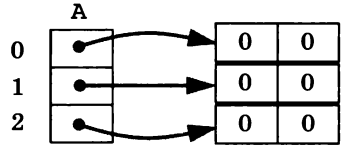


Рис. 8.29

Каждому элементу матрицы можно присвоить любое значение. Поскольку индексов два, для заполнения матрицы нужно использовать вложенный цикл. Далее будем считать, что существует матрица A, состоящая из N строк и M столбцов, а i и j — целочисленные переменные, обозначающие индексы строки и столбца. В этом примере матрица заполняется случайными числами и выводится на экран:

```
import random
for i in range(N):
  for j in range(M):
    A[i][j] = random.randint (20, 80)
    print("{:4d}".format(A[i][j]), end = "")
  print()
```

Такой же двойной цикл нужно использовать для перебора всех элементов матрицы. Вот как вычисляется сумма всех элементов:

```
s = 0
for i in range(N):
  for j in range(M):
    s += A[i][j]
print(s)
```

Поскольку мы не изменяем элементы матрицы, более красиво решать эту задачу в стиле Python:

```
s = 0
for row in A:
  s += sum(row)
print(s)
```

В цикле перебираются все строки матрицы A, каждая из них по очереди записывается в переменную row. В теле цикла сумма элементов строки прибавляется к значению переменной s.

Обработка элементов матрицы

Покажем, как можно обработать (например, сложить) некоторые элементы квадратной матрицы A, содержащей N строк и N столбцов.

На рисунке 8.30, *а* выделена **главная диагональ** матрицы, на рис. 8.30, *б* — вторая **(побочная)** диагональ, на рис. 8.30, *в* — главная диагональ и все элементы под ней.

а *б* *в*

Рис. 8.30

Главная диагональ — это элементы `A[0][0]`, `A[1][1]`, ..., `A[N - 1][N - 1]`, у которых номер строки равен номеру столбца. Для перебора этих элементов достаточно одного цикла:

```
for i in range(N):
  # работаем с  A[i][i]
```

Элементы побочной диагонали — это `A[0][N - 1]`, `A[1][N - 2]`, ..., `A[N - 1][0]`. Заметим, что сумма номеров строки и столбца для каждого из этих элементов равна `N - 1`, поэтому получаем такой цикл перебора:

```
for i in range(N):
  # работаем с  A[i][N-1-i]
```

В случае рис. 8.30, *в* (обработка всех элементов на главной диагонали и под ней) нужен вложенный цикл: номер строки будет меняться от 0 до `N - 1`, а номер столбца для каждой строки `i` — от 0 до `i`:

```
for i in range(N):
  for j in range( i+1 ):
    # работаем с  A[i][j]
```

Чтобы переставить столбцы матрицы, достаточно одного цикла. Например, переставим столбцы 2 и 4, используя вспомогательную переменную `c`:

```
for i in range(N):
  c = A[i][2]
  A[i][2] = A[i][4]
  A[i][4] = c
```

или, используя возможности Python:

```
for i in range(N):
  A[i][2], A[i][4] = A[i][4], A[i][2]
```

Переставить две строки можно вообще без цикла, учитывая, что A[i] — это ссылка на список элементов строки i. Поэтому достаточно просто переставить ссылки. Оператор

```
A[i], A[j] = A[j], A[i]
```

переставляет строки матрицы с номерами i и j.

Для того чтобы создать копию строки с номером i, нельзя делать так (подумайте почему):

```
R = A[i]
```

а вместо этого нужно создать копию в памяти:

```
R = A[i][:]
```

Построить копию столбца с номером j несколько сложнее, так как матрица расположена в памяти по строкам. В этой задаче удобно использовать генератор:

```
C = [row[j] for row in A]
```

В цикле перебираются все строки матрицы A, они по очереди попадают в переменную row. Генератор выбирает из каждой строки элемент с номером j и составляет список из этих значений.

С помощью генератора легко выделить в отдельный массив элементы главной диагонали:

```
D = [A[i][i] for i in range(N)]
```

Здесь предполагается, что матрица A состоит из N строк и N столбцов.

Выводы

- Матрица — это прямоугольная таблица, составленная из элементов одного типа (чисел, строк и т. д.). Каждый элемент матрицы имеет два индекса — номера строки и столбца.
- Матрица в Python — это список ссылок на списки-строки (список адресов строк).
- Для перебора всех элементов матрицы нужно использовать двойной цикл (по строкам и, внутри каждой строки, по столбцам).

Нарисуйте в тетради интеллект-карту этого параграфа.

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. Что такое главная и побочная диагонали матрицы?
5. Почему суммирование элементов главной диагонали требует одиночного цикла, а суммирование элементов под главной диагональю — вложенного?

Проекты

а) Игра «Крестики-нолики»
б) Игра «Морской бой»
в) Игра «Жизнь»
г) Игра «Найди пару»
д) Игра «Тетрис»
е) Программа для построения магических квадратов
ж) Программа для построения случайного лабиринта

§ 68
Работа с файлами

Ключевые слова:

- файл
- текстовый файл
- двоичный файл
- файловый указатель
- открытие файла
- закрытие файла

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 работать только с текстовыми файлами.

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 файла все последние изменения, сделанные программой в этом файле, записываются на диск.

Такой принцип работы иногда называют «принципом сэндвича», в котором три слоя: хлеб, затем начинка, и потом снова хлеб (рис. 8.31).

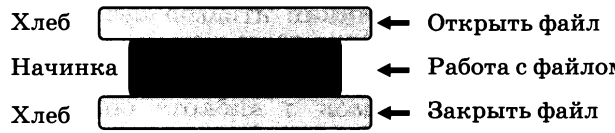


Рис. 8.31

В большинстве языков программирования с файлами работают через вспомогательные переменные (их называют указателями, идентификаторами и т. п.). В Python есть функция `open`, которая открывает файл и возвращает **файловый указатель** — переменную, через которую идёт вся дальнейшая работа с файлом. Функция `open` принимает два параметра: имя файла (или путь к файлу, если файл находится не в том каталоге, где записана программа) и режим открытия файла:

```
"r" – открыть на чтение;
"w" – открыть на запись;
"a" – открыть на добавление.
```

Метод `close`, определённый для файлового указателя, закрывает файл:

```
Fin = open ("input.txt")
Fout = open ("output.txt", "w")
   # здесь работаем с файлами
Fout.close()
Fin.close()
```

При первом вызове функции `open` режим работы с файлом не указан, но по умолчанию предполагается режим `"r"` (чтение).

Если файл, который открывается на чтение, не найден, возникает ошибка. Если существующий файл открывается на запись, его содержимое уничтожается.

После окончания работы программы все открытые файлы закрываются автоматически.

Чтение одной строки из текстового файла выполняет метод readline, связанный с файловой переменной:

```
s = Fin.readline()
```

Если нужно прочитать несколько значений, записанных в одной строке через пробел, используют метод split. Этот метод разбивает строку по пробелам и строит список из соответствующих «слов»:

```
s = Fin.readline().split()
```

Если в прочитанной строке файла были записаны числа 1 и 2, список s будет выглядеть так:

```
["1", "2"]
```

Элементы этого списка — символьные строки. Поэтому для того, чтобы выполнять с этими данными вычисления, их нужно перевести в числовой вид с помощью функции int, применив её для каждого элемента списка:

```
a, b = int(s[0]), int(s[1])
```

То же самое можно записать с помощью генератора

```
a, b = [int(x) for x in s]
```

или с помощью функции map, которая применяет функцию int ко всем элементам списка:

```
a, b = map( int, s )
```

Запись map(int,s) означает «применить функцию int ко всем элементам списка s».

Для вывода строки в файл применяют метод write. Если нужно вывести в файл числовые данные, их сначала преобразуют в строку, например так:

```
Fout.write("{:d} + {:d} = {:d}\n".format(a, b, a+b))
```

Таким образом, мы записали в файл результат сложения двух чисел. Обратите внимание, что, в отличие от функции print, при таком способе записи символ перехода на новую строку \n автоматически не добавляется, и мы добавили его в конце строки вручную.

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, можно закрыть его, а потом снова открыть.

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 и сразу его обрабатывать:

```
while не конец файла:
  прочитать число из файла
  добавить его к сумме
```

Для того чтобы определить, закончились данные или нет, будем использовать особенность метода `readline`: когда файловый курсор указывает на конец файла, метод `readline` возвращает пустую строку, которая воспринимается как ложное логическое значение:

```
while True:
  s = Fin.readline()
  if not s: break
```

В этом примере при получении пустой строки цикл чтения заканчивается с помощью оператора **break**.

Возможны и другие варианты. Например, метод `readlines` позволяет прочитать все строки сразу в список:

```
Fin = open("input.txt")
lst = Fin.readlines()
for s in lst:
  print(s, end = "")
Fin.close()
```

Строки файла читаются в список `lst` и выводятся на экран. Обратите внимание, что переход на новую строку в функции `print` отключён (`end = ""`), потому что при чтении символы перевода строки `\n` в конце строк файла сохраняются.

В Python есть ещё один способ работы с файлами, при котором закрывать файл не нужно, он закроётся автоматически. Это конструкция `with-as`:

```
with open("input.txt") as Fin:
  for s in Fin:
    print(s, end = "")
```

В первой строке файл input.txt открывается в режиме чтения и связывается с файловым указателем Fin. Затем в цикле перебираются все строки в этом файле, каждая из них по очереди попадает в переменную s и выводится на экран. Закрывать файл с помощью close не нужно, он закроется автоматически после окончания цикла.

Наконец, приведём ещё один способ в стиле Python:

```
for s in open("input.txt"):
  print(s, end = "")
```

Этот вариант обычно рекомендуют к использованию, в этом случае файл также не нужно закрывать.

Вернёмся к исходной задаче сложения чисел, записанных в файле в столбик. Далее будем считать, что файловая переменная Fin связана с файлом, открытым на чтение. Основная часть программы (без команд открытия и закрытия файлов) может выглядеть, например, так:

```
sum = 0
while True:
  s = Fin.readline()
  if not s: break
  sum += int(s)
```

или так:

```
sum = 0
for s in open("input.txt"):
  sum += int(s)
```

Обработка массивов

В текстовом файле записаны целые числа. Требуется вывести в другой текстовый файл те же числа, отсортированные в порядке возрастания.

Особенность этой задачи в том, что для сортировки необходимо удерживать в памяти все числа, т. е. для их хранения нужно выделить массив (список).

В большинстве языков программирования память под массив выделяется заранее, поэтому необходимо знать наибольшее возможное число элементов массива. Поскольку список в Python может расширяться, у нас этой проблемы нет. Цикл чтения может выглядеть так:

```
A = []
while True:
  s = Fin.readline()
  if not s: break
  A.append(int(s))
```

Есть и другой вариант — в стиле Python. Метод `read` для файлового указателя читает (в отличие от `readline`) не одну строку, а весь файл. Затем мы разобьём получившуюся символьную строку на слова с помощью метода `split`:

```
s = Fin.read().split()
```

и преобразуем слова в числа, составив из них список:

```
A = list(map(int, s))
```

Теперь нужно отсортировать массив *A* (этот код вы уже можете написать самостоятельно) и вывести их во второй файл, открытый на запись:

```
Fout = open("output.txt", "w")
Fout.write(str(A))
Fout.close()
```

В этом варианте мы использовали функцию `str`, которая возвращает символьную запись объекта, такую, которая выводится на экран. Если нам нужно форматировать данные по своему, например вывести их в столбик, можно обработать каждый элемент вручную в цикле:

```
for x in A:
  Fout.write(str(x)+"\n")
```

К символьной записи очередного элемента массива мы добавляем символ перехода на новую строку, который обозначается как `\n`. В этом случае числа выводятся в файл в столбик.

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 строке — кличка собаки, её возраст (целое число) и порода, например:

```
Мухтар 4 немецкая овчарка
```

Все 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):

```
while не конец файла (Fin):
  прочитать строку из файла Fin
  разобрать строку — выделить возраст
  if возраст < 5:
    записать строку в файл Fout
```

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 для разбора очередной прочитанной строки s можно использовать метод split, который вернёт список, где второй по счёту элемент (он имеет индекс 1) — это возраст собаки. Его и нужно преобразовать в целое число:

```
s = Fin.readline()
data = s.split()
sAge = data[1]
age = int(sAge)
```

Эти операции можно записать в краткой форме:

```
s = Fin.readline()
age = int(s.split()[1])
```

Полная программа (без учёта команд открытия и закрытия файлов) выглядит так:

```
while True:
  s = Fin.readline()
  if not s: break
  age = int(s.split()[1])
  if age < 5:
    Fout.write(s)
```

Её можно записать несколько иначе, используя метод readlines, который читает сразу все строки в список:

```
lst = Fin.readlines()
for s in lst:
  age = int(s.split()[1])
  if age < 5:
    Fout.write(s)
```

или конструкцию **with-as**:

```
with open("input.txt") as Fin:
  for s in Fin:
    age = int(s.split()[1])
    if age < 5:
      Fout.write(s)
```

Вот ещё один вариант в стиле Python, возможно, наилучший:

```
for s in open("input.txt"):
  age = int(s.split()[1])
  if age < 5:
    Fout.write(s)
```

Выводы

- Файл — это набор данных на диске, имеющий имя. С точки зрения программиста, бывают файлы двух типов: текстовые и двоичные. В текстовых файлах не может быть никаких специальных символов, кроме символов перехода на новую строку.
- До выполнения операций с файлом нужно открыть файл (сделать его активным), а после завершения всех действий — закрыть (освободить). После завершения работы программы все файлы закрываются автоматически.
- После открытия файла для обращения к нему используют переменную специального типа — файловый указатель.
- Файл — это «устройство» последовательного доступа к данным. Текущее место в файле определяется положением файлового курсора.

Нарисуйте в тетради интеллект-карту этого параграфа.

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. Как вы думаете, 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 самого начала?
9. Как определить, что данные в файле закончились?
10. В каких случаях нужно открывать одновременно несколько файлов?

Проекты

а) Программа для анализа текстовых файлов
б) Программа для поиска слов в файле
в) Программа для замены слов в файле

ЭОР к главе 8 на сайте ФЦИОР (http://fcior.edu.ru)

- Понятие алгоритма, его свойства. Линейный алгоритм
- Основные структуры данных
- Реализация основных алгоритмических конструкций
- Алгоритмы сортировки
- Алгоритмы поиска
- Организация работы с текстовыми файлами

Практические работы к главе 8

Работа № 39. «Знакомство со средой программирования»
Работа № 40. «Вычисления»
Работа № 41. «Случайные числа»
Работа № 42. «Ветвления»
Работа № 43. «Сложные условия»
Работа № 44. «Циклические алгоритмы»
Работа № 45. «Циклы по переменной»
Работа № 46. «Процедуры»
Работа № 47. «Процедуры-2»
Работа № 48. «Функции»
Работа № 49. «Логические функции»
Работа № 50. «Рекурсия»
Работа № 51. «Заполнение массивов»
Работа № 52. «Перебор элементов массива»
Работа № 53. «Линейный поиск в массиве»
Работа № 54. «Поиск максимального элемента в массиве»

Работа № 55. «Алгоритмы обработки массивов (реверс, сдвиг)»
Работа № 56. «Отбор элементов массива по условию»
Работа № 57. «Простые методы сортировки»
Работа № 58. «Сортировка слиянием»
Работа № 59. «Быстрая сортировка»
Работа № 60. «Двоичный поиск»
Работа № 61. «Символьные строки»
Работа № 62. «Функции для работы со строками»
Работа № 63. «Преобразования "строка-число"»
Работа № 64. «Строки в процедурах и функциях»
Работа № 65. «Рекурсивный перебор»
Работа № 66. «Сравнение и сортировка строк»
Работа № 67. «Матрицы»
Работа № 68. «Алгоритмы обработки матриц»
Работа № 69. «Файловый ввод и вывод»
Работа № 70. «Обработка массивов из файла»
Работа № 71. «Обработка смешанных данных из файла»

Глава 9
РЕШЕНИЕ ВЫЧИСЛИТЕЛЬНЫХ ЗАДАЧ НА КОМПЬЮТЕРЕ

§ 69
Точность вычислений

Ключевые слова:

- погрешность
- абсолютная погрешность
- относительная погрешность
- вычислительно устойчивый метод

«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 половина **цены 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 (рис. 9.1). Линейкой можно измерить высоту кружки снаружи и глубину внутри. Погрешность простых измерительных приборов, таких, как линейка, принято считать равной половине цены деления, т. е. 0,5 мм (0,05 см). Такой подход имеет простое объяснение: всегда легко «на глаз» указать ближайшее из двух делений, между которыми находится интересующая нас точка. Например, если измеренная высота

кружки оказалась примерно 8,2 см, на самом деле она может быть от 8,15 до 8,25 см. Если измеренная глубина равна 7,8 см, фактическая может быть от 7,75 до 7,85 см. 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 $8{,}15 - 7{,}85 = 0{,}3$ см, так и $8{,}25 - 7{,}75 = 0{,}5$ см. Таким образом, реальная толщина может быть от 0,3 до 0,5 см, и в полученном ответе (0,4 см) нет ни одной верной значащей цифры! Обычно в этом случае пишут ответ в виде $0{,}4 \pm 0{,}1$ см.

Рис. 9.1

В приведённом примере погрешность 0,1 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 измерений[1]. В данном случае относительную погрешность можно оценить как

$$\delta_x = \frac{0{,}1}{0{,}4} \cdot 100\% = 25\%.$$

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 (рис. 9.2). По известной формуле площади круга получаем (например, на калькуляторе):

$$S = \frac{\pi \cdot D^2}{4} = 1{,}130973355292325565846551617 9806....$$

D

Рис. 9.2

[1] На практике вместо x^* обычно используют среднее значение, полученное в результате серии измерений одной и той же величины.

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, на самом деле, равен как 1,1 см, так и 1,3 см. В этих «крайних» случаях получаем площадь

$$S = \frac{\pi \cdot 1{,}1^2}{4} = 0{,}950\ldots \quad \text{и} \quad S = \frac{\pi \cdot 1{,}2^2}{4} = 1{,}327\ldots$$

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 из-за неточности задания иррационального числа π.

Все практические расчёты выполняются неточно. Погрешность результата вычислений определяется 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:

$$a = 1000 \pm 0{,}001;\ b = 0{,}002 \pm 0{,}001;$$
$$c = 1000 \pm 0{,}001;\ d = 0{,}002 \pm 0{,}001.$$

Легко проверить, что вычисленное значение x может находиться в интервале от $-166\,667$ до $750\,001$, т. е. относительная погрешность превышает 300%! Такой метод расчёта x **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 приближенного вычисления некоторых стандартных функций (например, $\sin x$ или $\cos x$);
- накопление погрешностей при арифметических действиях с неточными данными;
- собственная погрешность используемого метода (например, 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.

Выводы

- 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, погрешностями метода и погрешностью вычислений (действий с приближёнными числами).
- При использовании вычислительно устойчивых методов малые погрешности в исходных данных не могут привести к большим погрешностям результата.

Нарисуйте в тетради интеллект-карту этого параграфа.

Вопросы и задания

1. Как вы понимаете приведённое в параграфе высказывание К. Ф. Гаусса?
2. Какие величины можно измерять? Какие приборы для этого используются?
3. Как определить цену деления для приборов с цифровыми индикаторами? Приведите примеры.
4. Чем различаются абсолютная и относительная погрешности? Какое из этих значений более важно в практических задачах?
5. Что такое вычислительно неустойчивый метод?

6. Из-за чего могут возникать ошибки при компьютерных вычислениях?
7. Какие задачи изучает вычислительная математика?

Подготовьте сообщение

а) «Абсолютная и относительная погрешность»
б) «Вычислительная устойчивость методов»

Проект

Экспериментальное исследование накопления ошибок при вычислениях

§ 70
Решение уравнений

Ключевые слова:

- аналитическое решение
- приближённый метод
- начальное приближение
- итерационный метод
- отделение корней уравнения
- уточнение корней уравнения
- метод перебора
- метод деления отрезка пополам
- целевая ячейка
- изменяемая ячейка

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, уменьшая шаг при построении графика до получения требуемой точности.

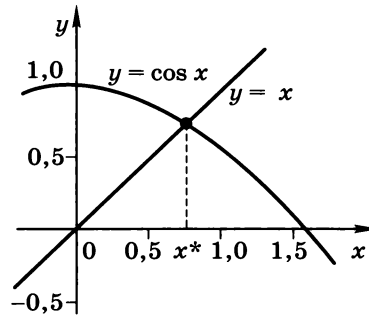


Рис. 9.3

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 малой.

Здесь не совсем ясно, что значит «пока длина отрезка не станет достаточно малой». Обычно задаётся нужная точность ε: это означает, что отклонение полученного решения от истинного x^* не должно быть больше ε.

Если установлено, что 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 решение изменяется на величину, бóльшую, 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. *iteratio* — 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. Во-первых, часто получение аналитического решения невозможно или требует значительных усилий, тогда как приближённые методы позволяют достаточно быстро решить задачу с заданной точностью. Во-вторых, 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 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).

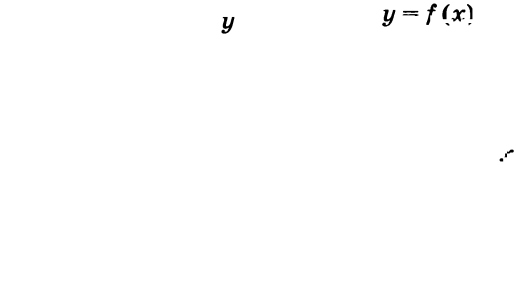


Рис. 9.4

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 всего ответить на второй вопрос: лучше всего взять в качестве решения середину полосы, т. е. точку $x_0 = x + \varepsilon$ (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*.

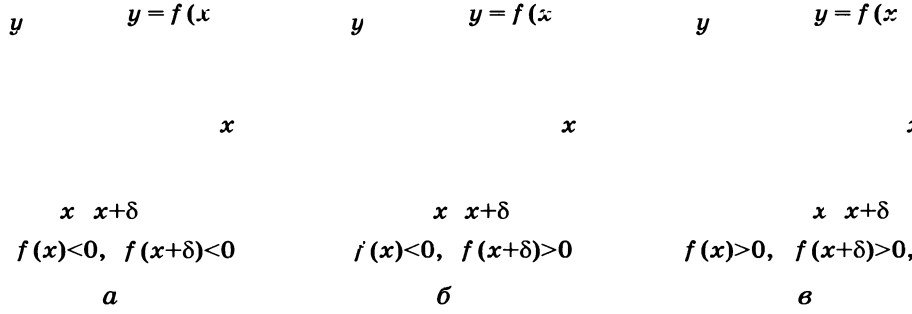


Рис. 9.5

Несложно сообразить, 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 алгоритмическом языке, а справа — на языке Python):

```
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
```

```
import math
def f(x):
  return x - math.cos(x)
eps = 0.001
x = 0
delta = 2*eps
while f(x)*f(x+delta) > 0:
  x = x + delta
print("x = ",
  "{:6.3f}".format(x+eps))
```

Здесь заданная точность ε хранится в виде константы `eps`, а вычисление функции 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 $f(x) \cdot f(x + \delta) \le 0$. Это означает, что одно из значений функции на концах отрезка положительное, а второе — отрицательное, или на каком-то конце отрезка значение функции равно нулю.

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}.

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 интервалы, 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. Эта шутка иллюстрирует метод **деления отрезка пополам** (метод **бисекции**), с помощью которого можно найти решение уравнения на некотором интервале (если оно там есть).

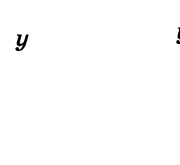


Рис. 9.6

Пусть некоторая функция $y = f(x)$ непрерывна на интервале $[a, b]$ 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 интервала

$$c = \frac{a + b}{2};$$

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 п. 2 нам нужно ответить на вопрос, есть ли решение на интервале $[a, c]$. 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
  все
кц
вывод "x = ", (a+b)/2:6:3
```

```
delta = 2*eps
while b - a > delta:
  c = (a + b) / 2
  if f(a)*f(c) <= 0:
        b = c
  else: a = c
print ("x = ",
  "{:6.3f}".format((a+b)/2))
```

На каждом шаге этого цикла длина 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 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 ему нужно бросить мяч, чтобы попасть в мишень на высоте 4 м на расстоянии 10 м от Васи? В момент броска мяч находится на высоте 2 м.

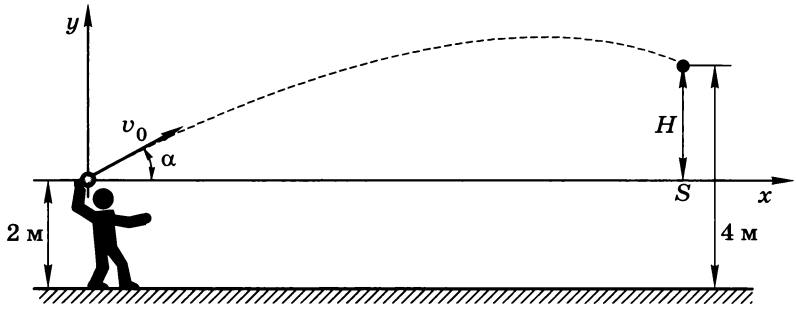


Рис. 9.7

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 шарик 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:

$$t = \frac{S}{v_0 \cdot \cos\alpha}.$$

Подставляя этот результат во второе уравнение, получаем уравнение с одним неизвестным α, которое можно привести к стандартному виду $f(\alpha) = 0$, где:

$$f(\alpha) = S \cdot \operatorname{tg}\alpha - \frac{g \cdot S^2}{2v_0^2 \cdot \cos^2\alpha} - H.$$

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 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
```

1) Хотя можно, например, с помощью замены $z = \frac{1}{\cos^2\alpha}$.

и на языке Python:

```
import math
u = 0
while u < math.pi/2:
  if f(u)*f(u + delta) <= 0:
    alpha = (u + eps)*180/math.pi
    print ("Угол: ",
           "{:4.1f}".format(alpha), " градусов")
  u += delta
```

В переменной u хранится угол в радианах, а в переменной alpha — угол в градусах. Если запустить эту программу, мы увидим, что уравнение имеет два решения — для углов, примерно равных 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 решения уравнений можно использовать табличный процессор, например *OpenOffice 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*, указывая на незначительные отличия *Excel*.

Введём исходные данные, 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 — см. рис. 9.8).

Имя ячейки или диапазона

Рис. 9.8

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. 9.9

Добавим столбцы, в которых для каждого угла с помощью стандартной функции RADIANS (в русской версии *Excel* — РАДИАНЫ) будет вычисляться его значение в радианах, а также время полёта, координата y и значение функции $f(\alpha)$ (рис. 9.10).

Рис. 9.10

Формулы в ячейках B6:E6 можно просто «растянуть» (скопировать) вниз за маркер заполнения. Обратите внимание, что в этих 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 B1, но это было бы менее понятно.

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*). График функции пересекает ось OX в двух точках, т. е. уравнение $f(\alpha) = 0$ имеет два решения, одно — около 35°, второе — около 65°.

Теперь уточним решение, используя возможности табличного процессора, в котором реализован один из приближённых методов 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. 9.11).

	H	I	J	K	L
1	**Угол**	**Угол(рад)**	**Время**	**y**	**f(alpha)**
2	35				

Рис. 9.11

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 (нуль). Ячейка H2 — это **изменяемая ячейка**. В главном меню выбираем пункт *Сервис* → *Подбор параметра* и вводим эти данные (рис. 9.12).

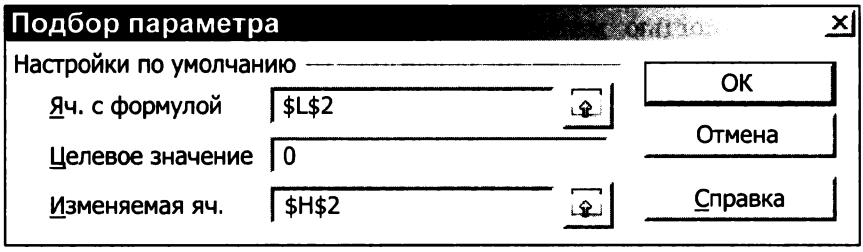


Рис. 9.12

После нажатия кнопки *OK* найденное решение уравнения будет записано в ячейку H2.

Как же найти второе решение? Для этого нужно выбрать другое начальное приближение, например $\alpha = 70°$, 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.

Выводы

- 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 некото-

рого алгоритма, который позволяет на каждом шаге приближаться к точному решению. Процедура заканчивается, когда разница между результатами двух последовательных шагов становится меньше заданной величины ε.

- При численном решении уравнений сначала нужно выполнить отделение корней (например, методом перебора), а затем уточнить решения.
- Подбор параметра в табличных процессорах использует численные методы.

Нарисуйте в тетради интеллект-карту этого параграфа.

Вопросы и задания

1. Сравните приближённые и аналитические методы решения уравнений. В чём достоинства и недостатки каждого подхода?
2. Приведите примеры методов решения задач, которые нельзя назвать итерационными.
3. Сравните метод перебора и метод деления отрезка пополам. В чём достоинства и недостатки каждого из них?
4. Как с помощью математических операций определяют, есть ли решение уравнения на заданном отрезке? В каких случаях такой подход не сработает?
5. Предложите, как избежать зацикливания в методе перебора.
6. Объясните, почему при ширине полосы $\delta = 2\varepsilon$ методом перебора можно найти решение с точностью ε. В каком случае погрешность будет наибольшей?
7. Чем различается отделение корней и уточнение корней?
8. Объясните изменения, сделанные в первоначальной программе для решения уравнения методом перебора, которые позволили в одном цикле найти все решения на заданном отрезке.

Подготовьте сообщение

а) «Метод хорд»
б) «Метод касательных (метод Ньютона)»
в) «Учёт сопротивления воздуха в моделях движения»

Проект

Сравнение численных методов решения уравнений

Интересные сайты

numpy.org — пакет *NumPy* для научных вычислений на языке Python

scipy.org — свободное программное обеспечение для решения математических и инженерных задач на Python

excelworld.ru — «Мир Excel»
planetaexcel.ru — «Планета Excel»
wiki.openoffice.org/wiki/RU/kb/module/calc — справка по *OpenOffice Calc*
help.libreoffice.org/Calc/Welcome_to_the_Calc_Help/ru — справка по LibreOffice Calc

§ 71
Дискретизация

Ключевые слова:

- дискретизация
- шаг дискретизации
- длина кривой
- площадь фигуры
- метод прямоугольников
- метод трапеций

Вычисление длины кривой

Определим длину траектории L, по которой летит мяч, брошенный под углом к горизонту (см. задачу в предыдущем параграфе). Это оказывается 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. 9.13 равна

$$L = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} + \sqrt{(x_3 - x_2)^2 + (y_3 - y_2)^2}.$$

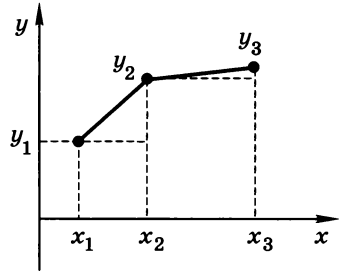


Рис. 9.13

Используя эту идею, разделим кривую линию на небольшие участки и заменим каждый участок отрезком так, чтобы получилась ломаная (штриховая линия на рис. 9.14).

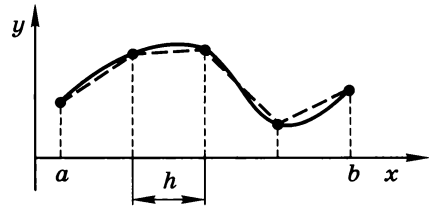


Рис. 9.14

Обычно разбивают исходный отрезок $[a, b]$ (на котором нужно определить длину кривой) на равные участки длиной 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 возрастет 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. главу 4).

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 алгоритмическом языке:

```
x:=a; L:=0
нц пока x < b
  y1:=f(x)
  y2:=f(x+h)
  L:=L + sqrt(h**2 + (y1 - y2)**2)
  x:=x + h;
кц
вывод "Длина кривой ", L:10:3
```

или так на языке Python:

```
import math
x = a; L = 0
while x < b:
  y1 = f(x)
  y2 = f(x + h)
  L += math.sqrt(h**2 + (y2 - y1)**2)
  x += h
print("Длина кривой", "{:10.3f}".format(L))
```

Возвращаясь к задаче с полётом мяча, 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:

$$y = f(\alpha) = x \cdot \operatorname{tg}\alpha - \frac{g \cdot x^2}{2v_0^2 \cdot \cos^2\alpha}.$$

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).

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. 9.15.

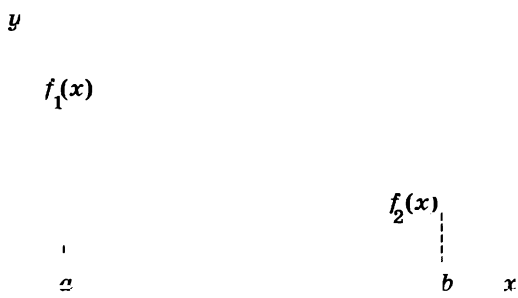


Рис. 9.15

В некоторых простых случаях (но далеко не всегда!) площадь такой фигуры можно вычислить аналитически с помощью мето-

дов 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 каждую полоску на другую фигуру, для которой мы можем легко найти площадь, например на прямоугольник (рис. 9.16).

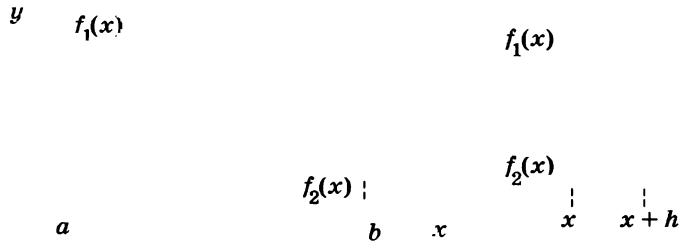


Рис. 9.16

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 9.16 высота прямоугольника рассчитывается как разность 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:8:3
```

```
S = 0; x = a
while x < b:
  S += f1(x+h/2)- f2(x+h/2)
  x += h
S *= h;
print("Площадь",
      "{:8.3f}".format(S))
```

Этот метод называется **методом прямоугольников.**

Вместо прямоугольника для замены удобно применять ещё одну известную фигуру, для которой легко считается площадь, — **трапецию** (рис. 9.17).

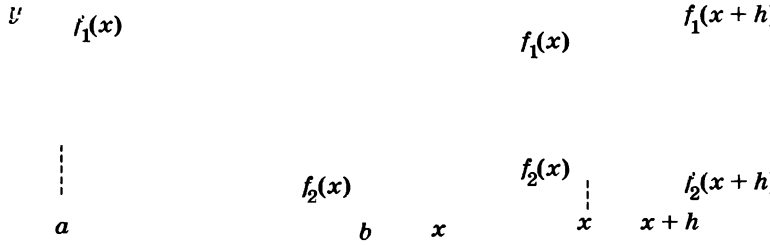


Рис. 9.17

Площадь трапеции равна произведению полусуммы её оснований на высоту, т. е. для элементарной трапеции с левой границей в точке x получаем выражение для площади:

$$\frac{h}{2} \cdot [f_1(x) - f_2(x) + f_1(x+h) - f_2(x+h)].$$

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:8:3
```

```
S = 0; x = a
while x < b:
  S += f1(x)- f2(x) \
     + f1(x+h) - f2(x+h)
  x += h
S *= h/2;
print("Площадь",
      "{:8.3f}".format(S))
```

Этот метод называют **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.

Выводы

- Дискретизация — необходимый этап подготовки вычислительных задач для решения на компьютере.
- Чтобы повысить точность, нужно уменьшать шаг дискретизации, но это увеличивает объём вычислений.
- При вычислении длины кривой нужно приближённо представить её в виде ломаной линии и сложить длины всех звеньев.
- Для вычисления площади фигуры нужно приближённо разбить её на элементы, площадь которых мы умеем вычислять, например на прямоугольники или трапеции.

Нарисуйте в тетради интеллект-карту этого параграфа.

Вопросы и задания

1. Что даёт дискретизация в рассмотренных задачах?
2. Что такое шаг дискретизации? Как должна быть связана его величина с длиной отрезка $[a, b]$, на котором выполняются расчёты?
3. Как зависит точность и время вычислений от выбора шага дискретизации?
4. Почему методы, основанные на дискретизации, всегда дают приближённый результат?
5. Подумайте, можно ли выполнять дискретизацию с переменным шагом. Ответ обоснуйте.

Проект

Сравнение приближённых методов вычисления площадей фигур

§ 72
Оптимизация

Ключевые слова:

- оптимизация
- оптимальное решение
- целевая функция
- ограничения
- локальный минимум
- глобальный минимум
- начальное приближение

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** $f(x)$, которая позволяет сравнивать решения. Оптимальным называется такое решение, при котором целевая функция достигает максимума (если это «доходы», «прибыль») или минимума («расходы», «потери»):

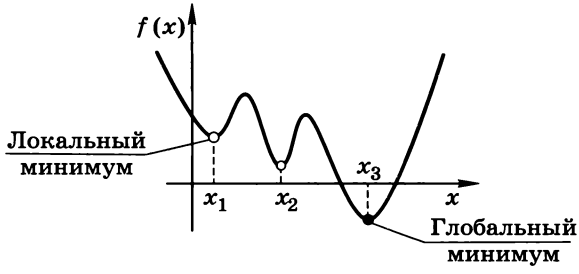
$$f(x) \to \max \text{ или } f(x) \to \min.$$

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») минимум. В точках x_1, x_2 и x_3 функция, график которой показан на рис. 9.18, *а*, имеет **локальные минимумы**. Это значит, что слева и справа от этих точек функция возрастает.



а *б*

Рис. 9.18

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[1)] предназначены именно для поиска локальных минимумов вблизи заданной начальной точки (**начального приближения**). Можно представить себе, что график функции — это срез поверхности (рис. 9.18, *б*), на которую устанавливается шарик в некоторой

---

1) Для некоторых типов функций существуют методы глобальной оптимизации, но они сложны и выходят за рамки школьного курса.

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^* (рис. 9.19). Нужно найти это значение x^* с заданной точностью ε.

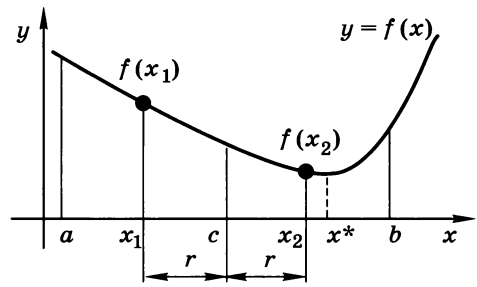


Рис. 9.19

Как и в методе бисекции для решения уравнений, мы последовательно «сжимаем» отрезок, пока его ширина не будет достаточно мала (меньше, чем 2ε). На каждом шаге (см. рис. 9.19):

1) вычисляется середина отрезка $c = \dfrac{a+b}{2}$;
2) вычисляются координаты двух точек, симметричных относительно середины: $x_1 = c - r$ и $x_2 = c + r$, где r — некоторое число, такое что $0 < r < (b-a)/2$;
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.

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
  иначе b:= x2
  все
кц
вывод "x = ", (a+b)/2
```

```
k = 0.01
delta = 2*eps
while b - a > delta:
  r = k*(b - a)
  x1 = (a + b)/2 - r
  x2 = (a + b)/2 + r
  if f(x1) > f(x2):
        a = x1
  else: b = x2
print ("x = ",
       "{:10.3f}".
       format((a+b)/2) )
```

В качестве упражнения можно исследовать работу этой программы при разных значениях k, подсчитав для каждого варианта количество шагов цикла, которое потребовалось 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 $0{,}5 + k = \frac{1}{\varphi}$, где φ — отношение «золотого сечения»:

$$\varphi = \frac{1+\sqrt{5}}{2}.$$

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 развертку (по штриховым линиям на рис. 9.20), сваривают швы, и таким образом получается бак.

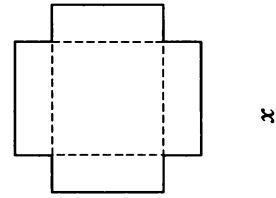


Рис. 9.20

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, а его высота равна x. Однако величина z зависит от x и равна $z = 1 - 2x$, поэтому объём бака вычисляется по формуле $V(x) = x(1 - 2x)^2$, 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 $V(x) = x(1 - 2x)^2$ 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 *OpenOffice Calc* встроенный модуль оптимизации работает только для линейных функций. Для того чтобы решить рассмотренную выше задачу с баком, нужно использовать дополнительный модуль *Solver for Nonlinear Programming* (он входит в стандартную поставку *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. 9.21).

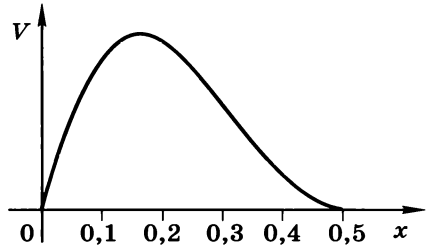


Рис. 9.21

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 при этом значении x (рис. 9.22).

Рис. 9.22

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 её максимально возможное значение), 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 щелкнуть на ячейке), а также ограничения (рис. 9.23).

После щелчка на кнопке *Решить* в ячейке E2 будет записано оптимальное значение x, а в целевой ячейке — соответствующее (максимальное) значение объёма.

Надстройка *Поиск решения* позволяет:

- находить максимум или минимум целевой функции;
- решать уравнения, задавая желаемое значение целевой функции;

Рис. 9.23

- 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.

Выводы

- 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.
- Табличные процессоры содержат модули, позволяющие выполнять оптимизацию.

Нарисуйте в тетради интеллект-карту этого параграфа.

Вопросы и задания

1. Какую роль играют целевая функция и ограничения в задачах оптимизации?
2. Почему выражение «самый оптимальный» безграмотно?
3. Что можно сказать о рекламной фразе «Этот крем обеспечивает оптимальный цвет лица»?
4. В чём разница между понятиями «локальный минимум» и «глобальный минимум»?
5. Почему результат решения задачи оптимизации чаще всего зависит от выбора начального приближения?
6. Объясните принцип работы метода дихотомии.
7. Обязательно ли при использовании метода дихотомии брать пробные точки симметрично относительно середины отрезка? Ответ обоснуйте.
8. Когда метод дихотомии не будет работать (может выдать неверный ответ)?

*9. Подумайте, можно ли задачу решения уравнения сформулировать как задачу оптимизации.

Проекты

а) Случайный поиск в задачах оптимизации
б) Генетические алгоритмы в задачах оптимизации
в) Моделирование отжига в задачах оптимизации
г) Поиск глобального минимума

§ 73
Статистические расчёты

Ключевые слова:

- ряд данных
- импорт данных
- среднее значение
- среднеквадратическое отклонение
- коэффициент корреляции

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.

Данные для обработки могут быть получены из внешних источников (файлов других форматов, баз данных и т. п.) с помощью операции **импорта**. Например, в программе *Excel* для этого используется вкладка *Данные* на ленте. Можно установить связь электронной таблицы с этими данными так, что при изменении исходного файла данные в таблице тоже будут обновляться.

Свойства ряда данных

Простейшая задача статистики — изучить свойства одного ряда данных. **Ряд данных** X длиной n — это множество значений x_1, x_2, ..., 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 (СЧЁТЕСЛИ) можно подсчитать число элементов ряда, удовлетворяющих некоторому условию. Например, если в диапазоне A1:A20 записаны школьные отметки, формула

=COUNTIF(A1:A20;">3")

вычисляет общее число четвёрок и пятёрок (всех ячеек, значения которых больше 3), а формула

=COUNTIF(A1:A20;"=5")

— число пятёрок. При поиске чисел, равных заданному, можно указывать число без кавычек и знака «=»:

=COUNTIF(A1:A20;5)

Более сложная характеристика ряда — **среднеквадратическое отклонение (стандартное отклонение)** σ_x, с помощью которого оценивается «разброс» значений x_1, x_2, ..., 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. 9.24).

Рис. 9.24

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, то записать в ячейку 0, иначе записать значение B2*0,2». В программе на языке Python мы бы записали

```
if B2 > 500:
      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 всех частей: 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:
  C2 = 0
elif B2 > 200:
  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

$$x_1,\ x_2,\ \ldots,\ x_n \text{ и } y_1,\ y_2,\ \ldots,\ y_n.$$

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 $|\rho_{xy}|$ 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)

Обратите внимание, что у этой функции два аргумента (два ряда данных одинаковой длины), адреса двух диапазонов разделяются 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.

[1)] Попробуйте доказать это самостоятельно.

Выводы

- Статистика — это наука, которая изучает методы обработки и анализа больших массивов данных.
- Цель статистических расчётов — выявить закономерности путем математической обработки больших массивов данных.
- Среднеквадратическое отклонение показывает величину разброса данных ряда относительно среднего значения.
- Коэффициент корреляции служит для определения связи между двумя рядами данных. Если его значение по модулю близко к единице, ряды связаны линейной зависимостью.
- Используя условные вычисления, в табличных процессорах можно изменять формулы, по которым выполняются расчёты, в зависимости от данных в других ячейках.

Нарисуйте в тетради интеллект-карту этого параграфа.

Вопросы и задания

1. Как вы думаете, в чём задача статистики?
2. Как влияют пустые ячейки 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?

Проекты

а) Анализ статистических данных по выбранной теме
б) Исследование взаимосвязи данных по выбранной теме

§ 74
Обработка результатов эксперимента

Ключевые слова:

- метод наименьших квадратов
- восстановление зависимости
- тренд
- линия тренда
- прогнозирование

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.

Например, с помощью динамометра и линейки можно экспериментально определить жёсткость пружины. 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 (рис. 9.25). Тогда $k = F/\Delta l$.

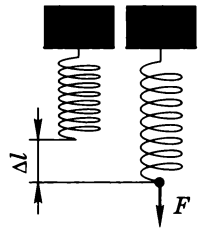


Рис. 9.25

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 **целевую 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. 9.26).

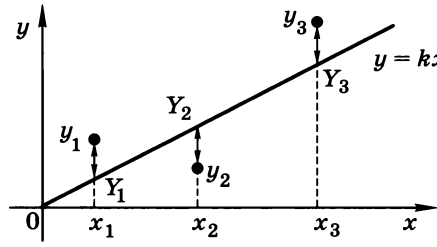


Рис. 9.26

Отклонение полученной линии от исходных данных определяется разностями $Y_i - y_i$ (см. рис. 9.26): 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 массивы x и y, 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
```

```
xx = [d*d for d in x]
xy = [x[i]*y[i]
      for i in range(len(x))]
B = sum(xy)
A = sum(xx)
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 с координатами (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. 9.27).

Рис. 9.27

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 $y = f(x)$. Требуется найти функцию $y = f(x)$ для того, чтобы вычислять значения y в тех точках, где они 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. 9.28).

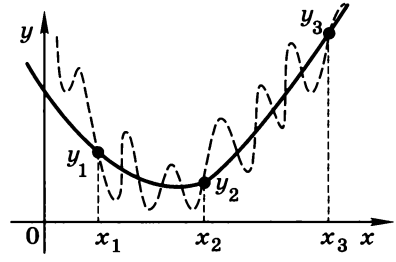


Рис. 9.28

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:

- *линейную* $y = a \cdot x + b$;
- *логарифмическую* $y = a \cdot \ln x + b$;
- *показательную* (экспоненциальную) $y = a \cdot b^x$;
- *степенную* $y = a \cdot x^b$.

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. Заметим, что можно не вводить данные вручную, а **импортировать** их — загрузить из внешнего источника: базы данных, электронной таблицы или даже текстового файла в формате *CSV* (англ. *comma-separated values* — значения, разделённые запятыми[1]). Например, в *OpenOffice Calc* можно импортировать целый лист из электронной таблицы или *CSV*-файла (пункт *Вставка* → *Лист из файла* в главном меню).

При импорте можно установить связь между источником данных и электронной таблицей, где выполняется обработка этих данных. Пусть исходные данные записаны в электронной таблице *data.ods*. Выделим эти данные и присвоим диапазону какое-то имя, например *Данные* (рис. 9.29).

1

Рис. 9.29

Чтобы вставить эти данные в новую таблицу и установить с ними связь, используем пункт главного меню *Вставка* → *Ссылка на внешние данные*. В диалоговом окне нужно выбрать файл *data.ods*, далее — диапазон *Данные* из этого файла и установить флажок *Обновлять* (рис. 9.30).

После этого данные появляются в таблице, причём они будут автоматически обновляться при любом изменении данных в исходном файле *data.ods*.

---

[1]) На практике чаще всего разделителем служит не запятая, а точка с запятой.

Рис. 9.30

Итак, данные введены. Следующий этап — построение диаграммы типа *Диаграмма XY* (в *Excel* — диаграмма *Точечная*) по этим данным. 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 (рис. 9.31).

Рис. 9.31

Если установить флажок *Показать уравнение*, то 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 лучше 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** на будущее. 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. 9.32), оказывается непригодной для прогноза.

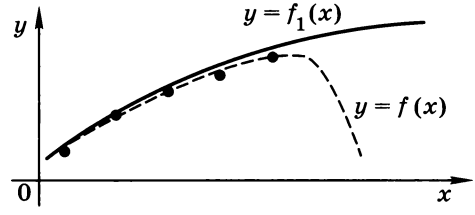


Рис. 9.32

В таких случаях для решения задачи прогнозирования нужно выбирать другую функцию, которая даёт меньшее значение R^2, но лучше показывает закономерность изменения величины (например, возрастающий характер функции).

Выводы

- Метод наименьших квадратов состоит в том, чтобы найти такую функцию, для которой сумма квадратов отклонений её значений от результатов измерений минимальна.
- Для того чтобы задача восстановления зависимости по точкам была корректной, необходимо заранее выбрать вид функции, так что останется определить лишь её неизвестные коэффициенты.
- Табличные процессоры содержат модули для решения задач восстановления зависимостей и прогнозирования.

Нарисуйте в тетради интеллект-карту этого параграфа.

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?
3. Почему задача восстановления зависимости некорректна, если не задан вид функции?

Подготовьте сообщение

а) «Интерполяция»
б) «Экстраполяция»
в) «Аппроксимация»

Проекты

а) Программа для обработки данных цифровой лаборатории
б) Прогнозирование изменения курса валют
в) Подбор зависимостей и прогнозирование на основе данных по выбранной теме

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

Практические работы к главе 9

Работа № 72 «Решение уравнений методом перебора»
Работа № 73 «Решение уравнений методом деления отрезка пополам»
Работа № 74 «Решение уравнений в табличных процессорах»
Работа № 75 «Дискретизация»
Работа № 76 «Оптимизация»
Работа № 77 «Статистические расчёты»
Работа № 78 «Обработка результатов эксперимента»

Глава 10
ИНФОРМАЦИОННАЯ БЕЗОПАСНОСТЬ

§ 75
Основные понятия

Ключевые слова:

- информационная безопасность
- защита информации
- доступность
- целостность
- конфиденциальность
- инсайдер

Что такое информационная безопасность?

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 нуждается государственная и военная тайна, коммерческая тайна, юридическая тайна, врачебная тайна. 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 по сравнению 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.

Средства защиты 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), чтобы сохранить 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).

Информационная безопасность в мире

Впервые требования к информационной безопасности компьютерных систем были сформулированы в 1983 г. в документе министерства обороны США под названием **«Критерии оценки надёжных компьютерных систем»** (его также называют «Оранжевая книга» по цвету обложки). В конце 1980-х гг. подобные документы появились во многих европейских странах.

В этих документах выделены основные угрозы информационной безопасности:

- кража данных и информации из сетей и баз данных;
- подмена информации;
- подделка документов в электронном виде;
- промышленный шпионаж (кража чертежей, технологий и т. п.)

и предлагаются различные варианты обеспечения минимального, но достаточного уровня защиты данных.

Международная организация по стандартизации (англ. **ISO**: *International Organization for Standardization*) утвердила несколько стандартов по информационной безопасности компьютерных систем, в том числе **«Практические правила менеджмента информационной безопасности»** и **«Системы управления информационной безопасностью»**.

В начале XXI века появилась новая угроза для всего мирового сообщества — **кибервойны**, т. е. использование Интернета и связанных с ним информационных технологий одним государством с целью причинения вреда военной, технологической, экономической, политической, информационной безопасности и суверенитету другого государства.

Кибератаки могут привести к очень тяжёлым последствиям, сравнимым с результатом действия обычного оружия. Например, в результате внедрения вредоносного кода в программное обеспечение систем управления ядерным реактором этот реактор может взорваться. При этом урон будет нанесён не только военным объектам, но и в большей степени мирному населению.

Опасны кибератаки не только против военных объектов. Атака на финансовую, транспортную или энергетическую систему тоже может привести к хаосу в государстве, и его последствия могут быть столь же тяжёлыми, как и вооружённое нападение.

К сожалению, международные нормы в области информационной безопасности ещё только начинают разрабатываться. В Организации Объединённых Наций работает Группа правительственных экспертов (ГПЭ) по международной информационной безопасности. В докладе ГПЭ 2015 года государствам — членам ООН было предложено соблюдать набор добровольных и необязательных норм ответственного поведения в киберпространстве. Участники ГПЭ согласились, что:

- государство отвечает за работу всех информационных систем и компьютерных сетей на своей территории;
- в киберпространстве необходимо соблюдать общие принципы международного права, закреплённые в Уставе ООН: суверенитет государств, невмешательство в их внутренние дела, мирное разрешение споров.

Информационная безопасность в России

Доктрина информационной безопасности Российской Федерации была утверждена в 2000 году. В ней информационная безопасность определяется как состояние защищённости национальных интересов в информационной сфере с точки зрения интересов личности, общества и государства. Согласно Доктрине, национальные интересы России включают:

- соблюдение конституционных прав и свобод человека в области получения и использования информации;
- информационное обеспечение государственной политики РФ;

- развитие современных отечественных средств информатизации, телекоммуникации и связи; обеспечение внутреннего рынка и выход на внешний рынок;
- защиту информационных ресурсов от несанкционированного доступа, обеспечение безопасности информационных и телекоммуникационных систем.

Для выполнения этих задач в Доктрине предусмотрены:

- *правовые методы* — разработка документов по информационной безопасности;
- *организационно-технические методы*:
  - создание системы информационной безопасности РФ;
  - защита сведений, составляющих государственную тайну;
  - привлечение к ответственности лиц, совершивших преступления в сфере информационной безопасности;
  - создание систем для предотвращения несанкционированного доступа к информации;
  - контроль за выполнением требований по защите информации;
- *экономические методы* — финансирование работ, связанных с обеспечением информационной безопасности РФ.

В Конституции РФ определено, что доступ к отдельным видам информации (например, к информации о состоянии окружающей среды) не может быть ограничен. В случае угрозы жизни и здоровью людей должностные лица обязаны информировать население о ней под страхом привлечения к уголовной ответственности.

Основные принципы правового регулирования отношений в сфере информации, информационных технологий и защиты информации закреплены в **Федеральном законе «Об информации, информационных технологиях и о защите информации»**, принятом в 2006 году. В нём определяются:

- порядок распространения информации в компьютерных сетях;
- обязанности организаторов распространения информации в сети Интернет;
- перечень информации, запрещённой для распространения (например, призывы к массовым беспорядкам и терроризму);
- меры по соблюдению авторских прав при распространении информации;
- порядок ограничения доступа к информации, которая распространяется с нарушением закона;
- ответственность провайдеров, операторов связи и владельцев сайтов за нарушение законов при распространении информации.

Информация, составляющая государственную, коммерческую или банковскую тайну, а также персональные данные граждан, — это информация ограниченного доступа. В то же время **в законах РФ «О Государственной тайне» и «Об информации, информационных технологиях и о защите информации»** перечислены сведения, не подлежащие засекречиванию (например, информация о работе государственных и местных органов управления, а также об использовании бюджетных средств).

Выводы

- Информационная безопасность — это защищённость информации от любых действий, в результате которых владельцам или пользователям информации может быть нанесен недопустимый ущерб.
- 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.
- Основные угрозы информационной безопасности — выход оборудования из строя, неправомерный доступ к информации, вредоносные программы.
- Слабое звено любой системы информационной защиты — это человек.
- Правовые вопросы информационной безопасности в Российской Федерации регулируются Конституцией РФ, Доктриной информационной безопасности РФ и Федеральным законом «Об информации, информатизации и защите информации».

Нарисуйте в тетради интеллект-карту этого параграфа.

Вопросы и задания

1. Что такое информационная безопасность?
2. Как вы понимаете выражение «недопустимый ущерб»?
3. Какие меры безопасности обычно применяются в организациях?
4. Почему при объединении компьютеров в сеть безопасность снижается?
5. Кто такие инсайдеры?
6. Узнайте, какие меры и средства защиты информации используются в вашей школе.
7. Почему в законах РФ установлено, что доступ к некоторой информации не может быть ограничен?
8. Сравните кибервойны с обычными войнами между государствами.

Подготовьте сообщение

а) «Защита информации в компьютерных сетях»
б) «Информационные войны»

Проект

Коллективная презентация по выбранной теме, связанной с информационной безопасностью

Интересные сайты

itsec.ru — сайт по информационной безопасности
securitylab.ru — сайт, посвящённый компьютерной безопасности
scrf.gov.ru/security/information/document5/ — Доктрина информационной безопасности РФ
security.ru/legislation.php — Законодательство РФ в области защиты информации

§ 76
Вредоносные программы

Ключевые слова:

- компьютерный вирус
- вредоносная программа
- ботнет
- червь
- троянская программа
- патч
- эксплойт

Что такое вредоносная программа?

Вредоносные программы (англ. *malware, malicious software* — злонамеренное программное обеспечение) — это программы, предназначенные для незаконного доступа к информации, для скрытого использования компьютера или для нарушения работы компьютера и компьютерных сетей.

Исторически первыми вредоносными программами были **компьютерные вирусы** — программы, способные при запуске создавать свои копии (необязательно точно совпадающие с оригиналом) и внедрять их в файлы и системные области компьютера. При этом копии могут распространяться дальше самостоятельно.

Очень часто словом «вирус» для краткости называют любую вредоносную программу.

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 переводить на свой счёт незаконно полученные деньги. Некоторые программы блокируют компьютер и для продолжения работы требуют отправить платное SMS-сообщение.

Заражённые компьютеры, подключённые 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 **DDoS-атакой**, или **распределённой DoS-атакой** (от 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. Иногда преступники продают доступ к ботнету или сдают его в аренду, например для распространению спама.

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, некоторые вирусы пишутся на коммерческой основе для полулегального («серого») бизнеса. Эти вирусы заставляют пользователей смотреть нежелательную рекламу, заманивают на платные сайты, предлагают скачать фальшивые антивирусы.

Иногда вирусы пишутся ради самоутверждения программистами, которые по каким-то причинам не смогли применить 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 вредоносных программ — это **уголовное преступление**, которое предусматривает (в особо тяжких случаях) наказание до 7 лет лишения свободы (Уголовный кодекс РФ, статья 273).

Заражение вредоносными программами

Признаки заражения вредоносной программой:

- замедление работы компьютера;
- уменьшение объёма свободной оперативной памяти;
- зависание, перезагрузка или блокировка компьютера;
- ошибки при работе ОС или прикладных программ;
- автоматический запуск неизвестных программ, открытие окон;
- изменение длины файлов, появление новых файлов (в том числе «скрытых»);
- невозможность запускать некоторые программы и изменять настройки компьютера;
- блокирование доступа к некоторым сайтам в Интернете (например, к сайтам антивирусных компаний);
- 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 (в ОС *Windows* — файлы с расширениями *exe*, *com*);
- загрузочные секторы дисков;
- пакетные командные файлы (*bat*);
- драйверы устройств;
- библиотеки динамической загрузки (*dll*), которые используются прикладными программами;
- документы, которые могут содержать *макросы* — небольшие программы, выполняющиеся при нажатии клавиш 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, «зашитая» в рисунок или видеофайл, всё-таки запустится.

Сейчас существуют два основных источника заражения вредоносными программами — флэш-накопители и компьютерные сети. **Компьютер может быть заражён при**:

- запуске заражённого файла;
- загрузке с заражённого CD (DVD)-диска или флэш-накопителя;
- автозапуске заражённого CD (DVD)-диска или флэш-накопителя (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 во вложении 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.

Большинство существующих вредоносных программ написано для ОС *Windows*, которая установлена более чем на 90% персональных компьютеров.

Известны также вирусы для *macOS* и *Linux*, но не каждому удаётся их запустить. Дело в том, что обычный пользователь (не администратор) в этих операционных системах не имеет права на изменение системных файлов, поэтому *macOS* 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.

---

[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.

Типы вредоносных программ

К вредоносным программам относятся компьютерные вирусы, черви, троянские программы, рекламное ПО (Adware) и др.

Среди **вирусов** обычно выделяют следующие типы:

- *файловые* — внедряются в исполняемые файлы, системные библиотеки и т. п.;
- *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*).

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).

Черви, как и вирусы, тоже способны самостоятельно распространяться, проникая на другие компьютеры через компьютерные сети. Они могут передаваться:

- в виде файла-вложения к сообщению электронной почты (почтовые черви);
- в виде ссылки на веб-страницу или другой ресурс в Интернете;
- через пиринговые сети (P2P, см. главу 7).

Червь заражает компьютер только тогда, когда пользователь запустит полученный файл или перейдёт по вредоносной ссылке.

Почтовые черви — это 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.

В Книгу рекордов Гиннеса занесён почтовый червь ILOVEYOU, заразивший в 2000 году более 3 миллионов компьютеров. Ущерб от его действий оценивается в 10–15 миллиардов долларов.

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, которая приводит к запуску вируса.

Иногда файл, пришедший как вложение в письмо, имеет двойное расширение, например:

```
СуперКартинка.jpg          .exe
```

На самом деле это программа (расширение имени файла *exe*), но пользователь может увидеть только первые две части имени и попытаться открыть такой «рисунок».

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**, от 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. Зараженные компьютеры используются для рассылки *спама* или массовых *DDoS-атак* на сайты в Интернете.

Существуют черви, которые могут распространяться через файлообменные сети, чаты и системы мгновенных сообщений, 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 (которые включаются, если некоторое время не работать на компьютере). В отличие от вирусов и червей, они не могут распространяться самостоятельно и часто «путешествуют» вместе с червями. «Троянцы» часто сопровождают файлы с нелицензионными программами.

Среди «троянцев» встречаются:

- *шпионские программы* — передают «хозяину» все данные, вводимые с клавиатуры (в том числе коды доступа к банковским счетам и т. п.), снимки экрана, список работающих приложений;

- *похитители паролей* — передают пароли, запомненные, например, в браузерах;
- *банковские троянцы* — крадут данные банковских карт, систем электронных платежей, пароли для подключения к интернет-банкам;
- *DDoS-троянцы* — создают из заражённых компьютеров ботнеты для проведения DDoS-атак на сайты;
- *фальшивые антивирусы* — вымогают деньги у пользователя, обещая обнаружить и удалить вирусы, которых на самом деле не существует;
- *вымогатели* — блокируют работу компьютера и требуют выплаты денег за разблокирование данных;
- *утилиты удалённого управления* — позволяют злоумышленнику управлять компьютером через Интернет (например, загружать и запускать любые файлы); они часто используются для создания ботнетов или зомби-сетей;
- *логические бомбы* — при определённых условиях (дата, время, команда по сети) уничтожают информацию на дисках.

Рекламное ПО (англ. *adware*, от слов *advertisement* — реклама и *software* — программное обеспечение) — это программы, которые предназначены для показа рекламы на компьютере, перенаправления поисковых запросов на рекламные сайты и сбора информации о вас (например, какие сайты вы постоянно посещаете). Часто такая реклама заманивает людей на платные сайты.

Рекламные программы могут попасть на компьютер двумя способами:

- вместе с бесплатным или условно-бесплатным программным обеспечением; как правило, от их установки можно отказаться; доходы от рекламы дают авторам возможность продолжать разработку и совершенствование этих программ;
- через заражённые веб-сайты без вашего ведома; для проникновения на ваш компьютер они могут использовать ошибки (уязвимости) в браузерах или загружаться с помощью троянской программы.

Вирусы для мобильных устройств

В последние годы появляется всё больше вредоносных программ для операционных систем мобильных устройств — смартфонов и планшетных компьютеров. Такие программы предназначены для незаконного получения персональной информации пользователя или его денег.

Наибольшее количество вредоносных программ для мобильных устройств создаётся для ОС *Android*, и это не случайно. *Android* — самая распространённая мобильная ОС для смартфонов (её доля составляет более 70% рынка), с открытым исходным кодом. Для неё достаточно просто писать программы, которые разрешено устанавливать в обход официального магазина *Google Play*.

Вредоносные программы могут, например:

- блокировать смартфон с целью вымогательства денег;
- рассылать (незаметно для пользователя) SMS- и MMS-сообщения;
- скрытно звонить на платные номера (в этом случае со счёта списывается значительная сумма денег);
- уничтожать данные на телефоне;
- похищать пароли учётных записей, например в интернет-магазинах *Google Play* или *AppStore*; эти данные передаются злоумышленникам, которые затем могут использовать их для покупки приложений за ваш счёт;
- ускорять разрядку аккумуляторной батареи;
- распространяться без участия человека по электронной почте, беспроводным сетям *Wi-Fi* и *Bluetooth*;
- обеспечивать «хозяину» возможность удалённо управлять вашим телефоном.

С помощью вредоносных программ воры стремятся получить доступ к мобильным приложениям для работы с банком. Если вы согласитесь на установку неизвестной программы, ссылка на которую придёт на ваш телефонный номер в SMS-сообщении, они смогут выполнять все банковские операции от вашего имени. Чтобы защитить пользователя, банки внедряют различные системы проверки, например, подтверждение всех операций с помощью кодов, отправляемых на телефон.

Иногда вредоносные программы работают в паре — троянская программа ворует имя пользователя и пароль для входа в интернет-банк, а вторая программа использует их для выполнения банковских операций, обманывая при этом систему защиты банка.

Вирусы могут попасть на ваш смартфон после установки программ (например, файлов с расширением *apk* для ОС *Adnroid*) не из магазина *Google Play*, а со сторонних сайтов и форумов, или в результате щелчка на каком-нибудь баннере с броской надписью. Обычно при этом владелец смартфона должен сам согласиться на установку и запуск ранее неизвестной программы. Если вы не уверены в безопасности, лучше не разрешать такие операции.

Кроме того, вирусы-черви, использующие ошибки (уязвимости, «дыры») в программном обеспечении, могут попасть в телефон и без участия человека (например, через связь по каналу *Bluetooth*, который по умолчанию включен).

Выводы

- Вредоносные программы — это программы, предназначенные для незаконного доступа к информации, для скрытого использования компьютера или для нарушения работы компьютера и компьютерных сетей.
- Заражённые компьютеры могут объединяться в ботнет — сеть зомби-компьютеров, которая управляется из некоторого центра и используется для рассылки спама, подбора паролей или атак на веб-сайты.
- Для того чтобы выполнить какие-то действия, вредоносная программа должна получить управление компьютером.
- Наиболее распространённые типы вредоносных программ — компьютерные вирусы, черви, троянские программы, рекламное ПО.
- В последние годы вредоносные программы активно атакуют мобильные операционные системы, особенно ОС *Android*. Цель таких программ — получить данные пользователя или его деньги.

Нарисуйте в тетради интеллект-карту этого параграфа.

Вопросы и задания

1. Что такое вредоносные программы? Какие вредоносные программы вы знаете?
2. Что такое компьютерный вирус? Чем он отличается от других программ?
3. Перечислите признаки заражения компьютера вирусом.
4. Какие объекты могут быть заражены вирусами? Какие не заражаются вирусами?
5. При каких действиях пользователя возможно заражение вирусом?
6. Чем опасны загрузочные вирусы?
7. Что такое макровирусы? Какие файлы они поражают?
8. Что могут заражать скриптовые вирусы?
9. Почему полиморфные вирусы сложно обнаруживать?
10. Сравните червей и троянские программы.

11. Почему необходимо сразу устанавливать обновления для операционных систем?
12. С какими целями могут быть использованы компьютеры, заражённые сетевым червем?
13. Почему многие почтовые серверы запрещают пересылку исполняемых файлов?
14. Что такое социальная инженерия? Как она используется авторами вирусов?
15. Какие операционные системы лучше защищены от вирусов? Почему?

Подготовьте сообщение

а) «Ботнеты»
б) «Социальная инженерия»
в) «Вредоносные программы для операционной системы Linux»
г) «Вредоносные программы для мобильных устройств»

Проект

Коллективная презентация «Типы вредоносных программ»

Интересные сайты

kaspersky.ru/internet-security-center — классификация вредоносных программ
vms.drweb.ru/classification — классификация вирусов от *DrWeb*

§ 77
Защита от вредоносных программ

Ключевые слова:

- антивирус
- сигнатура
- сканер
- монитор
- брандмауэр

Антивирусные программы

Антивирус — это программа, предназначенная для борьбы с вредоносными программами.

Антивирусы выполняют три основные задачи:

1) не допустить заражения 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 свой 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 зараженных файлов извне. Для этого мониторы:

- проверяют «на лету» все файлы, которые копируются, перемещаются или открываются в различных прикладных программах;
- проверяют используемые флэш-накопители;
- 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 мессенджеров).

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-накопителей.

Кроме вредоносных программ современные антивирусы частично защищают компьютер 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 (*shareware*), пробные версии с ограниченным сроком действия можно свободно загрузить из Интернета. Наиболее известны антивирусы *Антивирус Касперского* (**www.kaspersky.ru**), *DrWeb* (**www.drweb.com**), *Nod32* (**www.eset.com**), *McAfee* (**home.mcafee.com**).

На многих сайтах (**kaspersky.ru, 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*, *Avast Home*, *Antivir Personal*, *AVG Free*. Антивирус *ClamAV* распространяется свободно с исходным кодом.

На сайтах некоторых компаний можно найти **онлайновые антивирусы**. 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 *файрволами* (от англ. *firewall*). Брандмауэры запрещают передачу данных по каналам связи, которые часто используют вирусы и программы для взлома сетей.

Брандмауэр

Рис. 10.1

На рисунке 10.1 показана защита одного компьютера с помощью брандмауэра, точно так же защищаются от угроз из Интернета локальные сети.

Брандмауэр входит в состав современных версий ОС *Windows*, в ядро *Linux* также включен встроенный брандмауэр *Netfilter*. Иногда устанавливают дополнительные брандмауэры, например, *Agnitum Outpost* (**www.agnitum.com**), *Kerio Winroute Firewall* (**kerio.ru**) или бесплатную программу *Comodo Personal Firewall* (**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 CD (DVD)-дисках или флэш-накопителях.

Если вы работаете в сети, желательно включать антивирус-монитор и брандмауэр. Монитор также сразу сообщит об опасности, если вставленный флэш-накопитель 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 Интернета 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* не требует установки и может быть запущена с флэш-накопителя. Даже если удалить вирус не удалось, 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.

Выводы

- Антивирус — это программа, предназначенная для борьбы с вредоносными программами.
- Существуют два типа антивирусов: антивирусы-сканеры («защита по требованию») и антивирусы-мониторы (постоянная защита).
- Главные вред, который могут нанести вредоносные программы, — это потеря данных или паролей доступа к закрытой информации.
- Брандмауэры (сетевые экраны, файерволы) запрещают передачу данных по каналам связи, которые часто используют вредоносные программы для взлома компьютеров через сеть.

Нарисуйте в тетради интеллект-карту этого параграфа.

Вопросы и задания

1. Какие задачи решают антивирусы?
2. Что такое сигнатура? Почему нужно регулярно обновлять базы сигнатур антивирусов?

3. Сравните антивирус-сканер и антивирус-монитор.
4. Что значит «защита по требованию»?
5. Что такое эвристический анализ? В чём его достоинства и недостатки?
6. Какие ограничения есть у пробных версий коммерческих антивирусов?
7. Что такое онлайновый антивирус?
8. Зачем нужен брандмауэр?
9. В чём заключается основной вред, наносимый вирусами? Как можно уменьшить возможные потери?
10. Как можно улучшить безопасность компьютера при работе в сети Интернет?
11. Какие меры безопасности необходимы при работе с электронной почтой?
12. Какие действия можно предпринять, если компьютер заражён вирусом?

Подготовьте сообщение

1) «Бесплатные антивирусы»
2) «Что такое брандмауэр?»
3) «Аппаратные антивирусы»

Проект

Сравнение бесплатных антивирусных программ

Интересные сайты

kaspersky.ru — Антивирус Касперского
drweb.com — антивирус «Доктор Веб»
free.drweb.ru — бесплатная лечащая программа *Dr.Web CureIt*
avast.com — бесплатный антивирус *Avast Home*
avira.com — бесплатный антивирус *Avira*
free.grisoft.com — бесплатный антивирус *AVG Free*
clamav.net — свободно распространяемый антивирус *ClamAV*
kaspersky.ru/kfa — бесплатный антивирус Касперского
virustotal.com — служба проверки файлов и ссылок на вирусы

§ 78
Шифрование

Ключевые слова:

- шифрование
- криптография
- криптоанализ
- ключ шифра
- криптостойкость

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 любую информацию — 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 сообщений изучает наука криптология, история которой насчитывает около четырёх 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 показан алгоритм замены букв в шифре Цезаря со сдвигом 3.

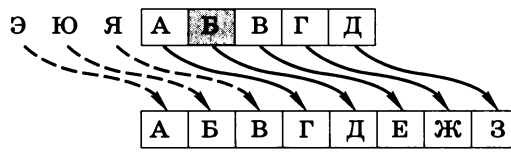


Рис. 10.2

Знаменитая фраза ПРИШЕЛ УВИДЕЛ ПОБЕДИЛ при использовании шифра Цезаря со сдвигом 3 будет закодирована так:

ТУЛЫИО ЦЕЛЗИО ТСДИЗЛО

1) Назван в честь римского императора Гая Юлия Цезаря, использовавшего его для секретной переписки.

Если первая буква алфавита имеет код 0, вторая — код 1 и т. д., то алгоритм шифрования может быть выражен формулой

$$y = (x + k) \mod n,$$

где x — код исходного символа, k — величина сдвига, y — код символа-замены, n — количество символов в алфавите, а запись $(x + k) \mod n$ 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[1]) для буквы «Я» (код 31) получаем код заменяющего символа $(31 + 3) \mod 32 = 2$, это буква «В».

Ключом для шифра Цезаря служит сдвиг k, если его знать, то сообщение легко расшифровать. Для этого используется формула

$$x = (y - k + n) \mod n.$$

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**[2], который стал естественным развитием шифра Цезаря. Для использования шифра Виженера используется ключевое слово, которое задаёт 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	Й	...

Рис. 10.3

Получаем: «З» — 7, «А» — 0, «Б» — 1, «Е» — 5, «Г» — 3. Это значит, что для кодирования первой буквы используется сдвиг 7, для кодирования второй — 0 (символ не меняется) и т. д. Для пятой буквы используется сдвиг 3, а для шестой — снова 7 (начали «проходить» кодовое слова с начала). Фраза ПРИШЕЛ УВИДЕЛ ПОБЕДИЛ при использовании шифра Виженера с ключом ЗАБЕГ будет закодирована в виде ЦРЙЭИТ УГНЗМЛ РУДМДЙР.

---

[1] Будем считать, что буквы Е и Ё совпадают.

[2] Назван по имени Блеза Виженера, швейцарского дипломата XVI века.

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.

Выводы

- Шифрование — это преобразование открытой информации в зашифрованную, недоступную для понимания посторонними.
- Криптография — это наука о способах шифрования информации. Криптоанализ — это наука о методах и способах вскрытия шифров.
- Ключ — это параметр алгоритма шифрования (шифра), позволяющий выбрать одно конкретное преобразование из всех вариантов, предусмотренных алгоритмом.
- 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.
- Криптостойкость шифра — это устойчивость шифра к расшифровке без знания ключа.
- Существуют системы шифрования с открытым ключом, при использовании которых все данные могут передаваться по незащищённому каналу связи.

Нарисуйте в тетради интеллект-карту этого параграфа.

Вопросы и задания

1. Чем различаются понятия «шифрование» и «кодирование»?
2. Что такое ключ?
3. Поясните разницу между криптографией и криптоанализом.
4. Что такое симметричный шифр? Какая проблема возникает при использовании симметричного шифра, если участники переписки находятся в разных странах?
5. Что такое несимметричные шифры?
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»

Проекты

а) Программа для шифрования с помощью шифра Цезаря
б) Программа для шифрования с помощью шифра Виженера
в) Программа для взлома шифра Цезаря (Виженера)

Интересные сайты

cryptography.ru — сайт «Математическая криптография»
pgpru.com — стандарт шифрования электронной почты *openPGP*

§ 79
Хэширование и пароли

Ключевые слова:

- хэширование
- хэш-функция
- хэш-сумма
- коллизия
- метод грубой силы

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. В то же время 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

$$215 = 65 \text{ (код «A»)} + 49 \text{ (код «1»)} + 50 \text{ (код «2»)} + 51 \text{ (код «3»)}.$$

Фактически мы определили функцию $H(M)$, которая сообщение M любой длины превращает в короткий код m. Такая функция называются **хэш-функцией** (от англ. *hash* — «мешанина», «крошить»), а само полученное число — **хэш-кодом, хэш-суммой**

или просто **хэшем** исходной строки. Важно, что, даже 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 же 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 уйдут 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 (прописные и строчные) и цифры, перебор вариантов (англ. *brute force* — **метод «грубой силы»**) со скоростью 10 млн паролей в секунду займёт 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. Нельзя использовать 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 прописных и строчных букв, цифр и других знаков[1)].

Сегодня для хэширования в большинстве случаев применяют алгоритмы *MD5*, *SHA-1*, *SHA-3* и российский алгоритм, изложенный в ГОСТ Р 34.11-2012 (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.

Выводы

- Хэширование — это преобразование массива данных в некоторое число.
- При хэшировании неизбежно возникают коллизии, поскольку большее множество отображается на меньшее.
- Хэширование используется для вычисления контрольных сумм при передаче данных.

Нарисуйте в тетради интеллект-карту этого параграфа.

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? Как выбрать надёжный пароль?
8. Какие алгоритмы хэширования сейчас чаще всего применяются?

*9. Предложите какой-нибудь свой метод хэширования. Подумайте, как часто при его использовании могут происходить коллизии.

---

1) Однако такой пароль сложно запомнить.

Подготовьте сообщение

а) «Ассоциативные массивы»
б) «Контрольные суммы при передаче данных»
в) «Хэширование с «солью»»
г) «Хэширование и Bitcoin»

Проект

Программа для хэширования паролей

§ 80
Современные алгоритмы шифрования

Ключевые слова:

- криптостойкость
- блочный шифр
- алгоритм RSA
- несимметричный шифр
- цифровая подпись

Государственным стандартом шифрования в России является алгоритм, зарегистрированный как **ГОСТ 28147-89**. Он является **блочным шифром**, т. е. шифрует не отдельные символы, а 64-битные блоки. В алгоритме предусмотрено 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 году по результатам проведенного конкурса. Шифр AES используется также в защищённых беспроводных сетях (*Wi-Fi*).

Алгоритм RSA

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 которого основана на том, что перемножить два очень больших простых числа достаточно просто, а вот разложить такое произведение на простые сомно-

жители 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 алгоритм 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 диапазоне от 0 до $n - 1$. Для шифрования используют формулу

$$y = x^e \bmod n,$$

где x — число исходного сообщения, (e, n) — открытый ключ, y — число закодированного сообщения, 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, состоящее из чисел 1, 2 и 3, 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.

Для того чтобы расшифровать сообщение, нужно знать секретный показатель степени d. А для этого, в свою очередь, нужно найти сомножители 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 году 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, сторонам достаточно обменяться открытыми ключами, которые можно показывать всем желающим.

Электронная цифровая подпись

С давних времён для того, чтобы установить авторство бумажного документа, на нём проверяли подпись и печать. В современном мире всё чаще обмен документами между людьми и организациями происходит в электронном виде. При этом проверить авторство достаточно сложно: изображение подписи и печати на отсканированном изображении может быть подделано! Решить эту проблему удалось с помощью электронной цифровой подписи.

Электронная цифровая подпись (ЭЦП) — это набор символов, который получен в результате шифрования сообщения с помощью личного секретного ключа отправителя.

ЭЦП служит для доказательства авторства документов, защиты сообщений от подделки и умышленных изменений.

Особенность алгоритма шифрования ЭЦП состоит в том, что сообщение шифруется одним ключом (секретным, который знает только отправитель), а расшифровывается — другим, открытым, который знают все. Такое шифрование называется **асимметричным**. Для создания электронной цифровой подписи можно использовать алгоритм RSA.

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.

Специальные удостоверяющие центры, честность которых не вызывает сомнения, выдают **сертификат на электронную подпись** — документ, подтверждающий, что открытый ключ действительно принадлежит организации или частному лицу. Свой секретный ключ владелец сертификата должен хранить в тайне.

Электронная подпись в России используется при обмене цифровыми документами между бухгалтериями предприятий, банками, Пенсионным фондом, торговыми организациями, судами. Согласно **Федеральному закону «Об электронной подписи»**, электронный документ, подписанный электронной цифровой подписью, признается равнозначным бумажному документу, подписанному собственноручной подписью.

Программное обеспечение

Во многих современных программах есть возможность шифровать данные с паролем. Например, офисные пакеты *OpenOffice* 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 ОС *Linux*, *Windows* и *Android*. Программы *TrueCrypt* (**www.truecrypt.org**), *BestCrypt* (**www.jetico.com**) и *FreeOTFE* (**freeotfe.org**) созда-

ют логические диски-контейнеры, информация на которых шифруется. Свободно распространяемая программа *DiskCryptor* (**diskcryptor.net**) позволяет шифровать разделы жёстких дисков и даже создавать шифрованные флэш-накопители и CD/DVD диски.

Программа *GnuPG* (**gnupg.org**) также относится к свободному программному обеспечению. В ней поддерживаются симметричные и асимметричные шифры, а также различные алгоритмы электронной цифровой подписи.

Выводы

- Государственным стандартом шифрования в России является блочный алгоритм ГОСТ 28147-89.
- Алгоритм RSA — это алгоритм с открытым ключом, стойкость которого основана на том, что перемножить два очень больших простых числа достаточно просто, а разложить такое произведение на простые сомножители очень трудно.
- Электронная цифровая подпись — это набор символов, который получен в результате шифрования сообщения с помощью личного секретного кода отправителя.

Нарисуйте в тетради интеллект-карту этого параграфа.

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 чём 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) «Алгоритм шифрования RC4»
д) «Цифровая подпись в Российской Федерации»

Проект

Обмен сообщениями по открытому каналу (алгоритм RSA)

Интересные сайты

inssl.com/standart-of-cipher.html — алгоритм ГОСТ 28147-89
crypt-online.narod.ru/crypts/rsa/ — алгоритм RSA онлайн
iecp.ru — портал «Электронная подпись»

§ 81
Стеганография

Ключевые слова:

- стеганография
- метод наименьших значащих битов
- цифровой водяной знак

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 определённых 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 (метод **наименьших значащих битов**). Причём это нужно сделать так, чтобы разница между исходным и полученным рисунками была неощутима для человека. Например, в полутоновом рисунке (256 оттенков серого) яркость каждого пикселя кодируется 8 битами. Если поменять 1–2 младших бита этого кода, «встроив» туда текстовое сообщение, то фотография, в которой нет чётких границ объектов, почти не изменится. При замене одного бита каждый байт исходного текстового сообщения хранится в младших битах кодов 8 пиксе-

лей. Например, пусть первые 8 пикселей рисунка имеют такие коды (рис. 10.4).

10101101	10010100	00101010	01010010	10101010	10101010	10101011	10101111

Рис. 10.4

Чтобы закодировать в них код буквы «И» (11001000_2), нужно изменить младшие биты кодов (рис. 10.5).

10101101	10010101	00101010	01010010	10101011	10101010	10101010	10101110
1	1	0	0	1	0	0	0

Рис. 10.5

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 свое 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 (рис. 10.6).

Рис. 10.6

Выводы

- Стеганография — это наука о скрытой передаче информации путём скрытия самого факта передачи информации.
- Цифровые водяные знаки — это внедрённая в файл информация об авторе изображения.

Нарисуйте в тетради интеллект-карту этого параграфа.

Вопросы и задания

1. Какие методы стеганографии существовали до изобретения компьютеров?
2. Как можно добавить текст в закодированное изображение?
3. На чём основаны методы стеганографии для звуковых данных и видеоданных?
4. Что такое цифровые водяные знаки? Зачем они используются?

Подготовьте сообщение

а) «История стеганографии»
б) «Метод наименьших значащих битов»
в) «Методы стеганографии»

Проект

Программа для встраивания сообщения в рисунок

§ 82
Безопасность в Интернете

Ключевые слова:

- сетевые угрозы
- фишинг
- мошенничество
- выбор пароля

Несмотря на многие достоинства Интернета, нужно помнить о том, что виртуальная жизнь — это продолжение реальной, и в ней тоже есть нечестные люди. Для того чтобы не навредить себе и своим знакомым, нужно соблюдать правила информационной безопасности.

Сетевые угрозы

Вредоносные программы, распространяющиеся через Интернет, представляют серьезную угрозу безопасности данных. Многих проблем можно избежать, если работать в Интернете только из-под ограниченной учётной записи (без прав администратора) — это не позволит многим вирусам и «троянцам» проникнуть в вашу систему. Кроме того, желательно своевременно обновлять программное обеспечение: особенно важно устанавливать «заплатки», связанные с безопасностью.

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, логинов и паролей для входа на почтовые серверы, в социальные сети, платёжные системы;
- *мошенничество* — хищение денег 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.

В результате атаки вредоносных программ может быть удалена важная информация на вашем компьютере: архив фотографий, документы, финансовые данные. Для пользователя кража данных может привести к вторжению в его личную жизнь и к потере денег с банковского счёта; для организации — утечку секретной финансовой и технической информации.

Иногда случается, что почтовые ящики или учётные записи на сайтах «взламывают» для того, чтобы как-то их использовать, например для рассылки спама или для публикации от вашего имени какой-то порочащей вас информации. Чаще всего это происходит потому, что ваш пароль был очень простой, и его удалось подобрать. Помните, что такие действия — уголовное преступление, их расследованием занимается специальный отдел полиции.

Абсолютной защиты от таких атак нет (кроме как отключить Интернет!), но во многих случаях самых простых мер оказывается достаточно:

- используйте антивирусную программу-монитор, которая постоянно находится в памяти и отслеживает подозрительную активность других программ;

- не открывайте электронные письма от неизвестных пользователей, без темы сообщения или без текста; не запускайте файлы-приложения к ним;
- не отвечайте на письма-спам;
- придумывайте сложные пароли.

Мошенничество

Мошенничество процветает потому, что многие пользователи Интернета очень доверчивы и неосторожны. Для того чтобы обманом получить ваши деньги, пароли, данные банковских карт и т. д., мошенники используют спам. Нередко спам — это часть фишинга.

Фишинг (англ. *phishing* — искажение слова *fishing* — рыбная ловля) — это выманивание паролей. Для этого чаще всего используются сообщения, рассылаемые по электронной почте 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.

Один из видов спама называется **«нигерийские письма»**, потому что большое количество таких писем приходило из Нигерии. 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.

Иногда в письме сообщается, что получатель письма якобы может получить большое наследство, а отправитель может ему в этом помочь. За «услуги по оформлению» отправитель письма просит перевести ему «немного» денег.

Существуют фирмы, которые за деньги организуют рассылку рекламы по всем адресам, попавшим в их базу данных. Если вы написали свой адрес электронной почты на каком-либо общедоступном форуме, он может попасть в такую базу, и тогда на ваш

почтовый ящик будет поступать спам. Часто в спам-сообщения вставляют ссылку, якобы позволяющую отписаться от рассылки, но переход по ней может только увеличить поток спама.

Мошенничество может быть связано и с вредоносными программами. В 2010 году несколько миллионов компьютеров в России было заражено троянской программой *Winlock*, которая блокировала компьютер и требовала отправить платное SMS-сообщение для снятия блокировки.

Некоторые сайты в Интернете предлагают платные услуги. Например, для того чтобы скачать какой-то файл, нужно отправить платное SMS-сообщение на указанный номер. Во многих случаях — это обман, и никакого файла вы не получите, а деньги со счёта будут списаны.

Пароли

Пароль — это ваш «ключ» для входа в некоторую закрытую от посторонних зону Интернета, например в почтовый ящик, учётную запись в социальной сети или личный кабинет интернет-банка.

Нужно выбирать такой пароль, чтобы его было сложно (практически невозможно) узнать. Длина пароля должна быть не менее 6–7 символов, он не должен состоять из одних цифр (такие пароли легко подобрать). Желательно, чтобы пароль содержал как буквы (прописные и строчные), так и цифры. Плохо, когда пароль — это слово, которое можно найти в словаре, потому что есть программы, подбирающие пароли по словарю.

Нельзя сообщать другим людям пароль от своей учётной записи, иначе они смогут зайти на сайт и сделать что-то от вашего имени. Не оставляйте бумажку с записанным на ней паролем около компьютера, лучше всего вообще нигде не записывать пароли.

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 человек, открывший эту страницу после вас, 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.

Шифрование данных

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 защищённый протокол HTTPS (англ. *Hypertext Transfer Protocol Secure* — безопасный HTTP), который предусматривает шифрование данных (например, с помощью алгоритмов RSA и RC4). Поэтому нужно проверять, чтобы адрес на странице ввода пароля в таких системах начинался с *https://*, а не с *http://*.

Правила личной безопасности

Общаясь в Интернете, нужно особенно осторожно относиться к публикации личных данных. На многих сайтах просят заполнить **«профиль»** — анкету с данными о вас. Помните, что всё, что вы размещаете в социальной сети, может быть доступно всем, в том числе и злоумышленникам. Наверняка вы никогда не расскажете первому встречному, где вы живёте и сколько зарабатывают ваши родители. Так же нужно вести себя и в Интернете — не сообщайте личную информацию (адрес, номер телефона и т. д.) незнакомым людям, не пишите её на своей страничке в социальной сети.

Нужно понимать, что, размещая какую-то информацию в Интернете, вы делаете её доступной для широкого круга лиц, включая работодателей, полицию, 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.

Когда вы пойдёте устраиваться на работу в какую-то компанию, вашу страничку в сети, скорее всего, будут просматривать сотрудники службы безопасности. Бывает и так, что работников даже увольняют за некорректные комментарии или фотографии.

Иногда, познакомившись через Интернет, люди хотят встретиться и в реальной жизни: сходить вместе в поход, посидеть в кафе и т. д. Помните, что в Интернете каждый может создать

собственный образ, такой, какой он хочет. Недоучившийся студент может представиться успешным бизнесменом, а пожилой дворник — молоденькой девушкой. Поэтому будьте очень осторожны, перенося виртуальные знакомства в реальную жизнь, постарайтесь как можно лучше узнать человека, с которым хотите встретиться, посоветуйтесь с родителями и старшими товарищами.

Выводы

- Фишинг — это выманивание паролей с помощью поддельных сайтов, которые внешне выглядят точно так же, как настоящие.
- Простейшие правила информационной безопасности:
  - используйте антивирусную программу-монитор;
  - не открывайте электронные письма от неизвестных пользователей, без темы сообщения или без текста; не запускайте файлы-приложения к ним;
  - не отвечайте на письма-спам;
  - придумывайте сложные пароли.
- При выполнении денежных операций нужно обязательно проверить, что используется протокол HTTPS (шифрование данных).

Нарисуйте в тетради интеллект-карту этого параграфа.

Вопросы и задания

1. Какие угрозы безопасности существуют при подключении к Интернету?
2. Какие схемы интернет-мошенничества вам известны?
3. Что можно сделать, если вас обманули мошенники в Интернете?
4. Какие меры безопасности нужно соблюдать при работе в Интернете?
5. Как обеспечивается безопасность обмена данными при денежных расчётах в Интернете?

Подготовьте сообщение

а) «Мошенничество в Интернете»
б) «Нигерийские письма»
в) «Фишинг»
г) «Стойкие и нестойкие пароли»
д) «Протокол HTTPS»

Проект

Коллективная работа «Угрозы Интернета»

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

Практические работы к главе 10

Работа № 79 «Использование антивирусной защиты»
Работа № 80 «Шифрование и хеширование»
Работа № 81 «Современные алгоритмы шифрования»
Работа № 82 «Стеганография»

ОГЛАВЛЕНИЕ

Учебное издание

Поляков Константин Юрьевич
Еремин Евгений Александрович

ИНФОРМАТИКА

(базовый и углублённый уровни)

(в 2 частях)

10 класс

Часть 2

Учебник

Ведущий редактор *О. Полежаева*
Ведущие методисты *И. Хлобыстова*
Художники *Н. Новак, Я. Соловцова*
Технический редактор *Е. Денюкова*
Корректор *Е. Клитина*
Компьютерная верстка: *В. Носенко*

Подписано в печать 25.03.19. Формат 70x100/16. Усл. печ. л. 28,6.
Тираж 15 000 экз. Заказ № м7815.

ООО «БИНОМ. Лаборатория знаний»
127473, Москва, ул. Краснопролетарская, д. 16, стр. 3,
тел. (495) 181-5344, e-mail: binom@Lbz.ru
http://www.Lbz.ru, http://metodist.Lbz.ru

Отпечатано в филиале «Смоленский полиграфический комбинат»
ОАО «Издательство «Высшая школа».
Российская Федерация, 214020, Смоленск, ул. Смольянинова, 1
Тел.: +7 (4812) 31-11-96. Факс: +7 (4812) 31-31-70
E-mail: spk@smolpk.ru http://www.smolpk.ru