А. Н. ГУДА, М. А. БУТАКОВА, Н. М. НЕЧИТАЙЛО, А. В. ЧЕРНОВ

ИНФОРМАТИКА

УЧЕБНИК

Издательско-торговая корпорация «Дашков и К°»

А. Н. Гуда, М. А. Бутакова,
Н. М. Нечитайло, А. В. Чернов

ИНФОРМАТИКА
общий курс

Учебник

Под общей редакцией
академика РАН, доктора технических наук,
профессора В. И. Колесникова

3-е издание

Допущено
Учебно-методическим объединением
по образованию в области прикладной
информатики в качестве учебника для студентов
высших учебных заведений, обучающихся
по специальности "Прикладная информатика
(по областям)" и другим экономическим
специальностям

Москва, 2010

УДК 681.3(07) + 06
ББК 32.07
Г93

Рецензенты:

Кафедра "Прикладная информатика" ТРТУ (заведующий кафедрой — академик РАЕН, заслуженный деятель науки и техники Российской Федерации, доктор технических наук, профессор Л. С. Берштейн);

Л. Г. Матвеева — доктор экономических наук, профессор,заведующая кафедрой "Прикладная информатика РТУ.

Руководитель авторского коллектива

А. Н. Гуда — доктор технических наук, профессор.

Гуда А. Н.

Г93 Информатика. Общий курс: Учебник / А. Н. Гуда, М. А. Бутакова, Н. М. Нечитайло, А. В. Чернов; под ред. академика РАН В. И. Колесникова. — 3-е изд. — М.: Издательско-торговая корпорация «Дашков и К°»; Ростов н/Д: Наука-Спектр, 2010. — 400 с.

ISBN 978-5-394-00937-2

Учебник предназначен для изучения курсов "Информатика", "Информатика и программирование" в высших учебных заведениях. Содержание учебника соответствует установленным Министерством образования и науки России дидактическим единицам стандартов высшего профессионального образования.

Рассмотрены основные понятия информатики, аппаратное устройство компьютеров и их программное обеспечение, вопросы функционирования операционных систем и компьютерных сетей, аспекты информационной безопасности. Значительное внимание уделено приобретению практических навыков работы с пакетом офисных программ, а также изучению принципов разработки алгоритмов и программ.

Для студентов вузов.

УДК 681.3(07) + 06
ББК 32.07

ISBN 978-5-394-00937-2

ПРЕДИСЛОВИЕ

Приступая к написанию учебника, авторы осознавали, что информатика является, пожалуй, наиболее интенсивно и динамично развивающейся отраслью знаний. Информатике посвящено множество учебников, учебно-методических пособий, сайтов в сети Internet. Тем не менее авторы, имеющие немалый опыт чтения курсов лекций по информатике в ведущих технических вузах Южного федерального округа, постарались, с одной стороны, изложить материал в соответствии с государственными образовательными стандартами большинства направлений высшего профессионального образования. С другой стороны, хотелось, чтобы учебник помог читателю приобрести достаточные знания и навыки для дальнейшего освоения информационных технологий.

Реалии нашей жизни таковы, что при преподавании информатики даже самый опытный лектор не сможет охватить все области ее применения. Материал, представленный в учебнике, отражает лишь основные разделы дисциплины и является необходимым минимумом, соответствующим дидактическим единицам образовательных стандартов. Авторы постарались, выдержав логическую последовательность изложения дисциплины, сделать тем не менее главы учебника по возможности автономными, чтобы читатель мог при желании более глубоко освоить отдельные разделы в зависимости от конкретных задач, стоящих перед ним.

Изучение любой дисциплины традиционно начинается с определения ее предметной области, введения основных понятий, исторических сведений. Первая глава учебника посвящена обсуждению предмета, задач информатики и ее фундаментальных понятий.

Зарождение информатики как научной дисциплины непосредственно связано с появлением и развитием средств вычислительной техники, среди которых центральное место занимают сейчас персональные компьютеры. Вторая глава учебника познакомит читателя с устройством и основными принципами работы компьютера, функционированием периферийных устройств. В конце главы дается обзор наиболее популярных программных продуктов, который позволит пользователю сориентироваться в огромном количестве существующего программного обеспечения.

Как правило, современный пользователь персонального компьютера может самостоятельно освоить графический интерфейс операционной системы, которая является неотъемлемой частью программного обес-

печения. В третьей главе авторы сконцентрировали внимание на внутреннем устройстве операционных систем семейства *Microsoft Windows*, знание которого придаст уверенности при установке, настройке и эксплуатации программного обеспечения. Не в последнюю очередь возникает вопрос о востребованности специалистов со знанием операционных систем семейства *Unix*, поэтому авторы сочли необходимым включить краткие сведения по этому вопросу.

Не будет преувеличением сказать, что пользователь компьютера не сможет обойтись без офисного программного обеспечения. Последовательно выполнив рекомендации, изложенные в четвертой главе, читатель приобретет практические навыки эффективного использования популярнейшего пакета программ *Microsoft Office*.

Деятельность любого современного предприятия связана со сбором, хранением и обработкой собственных банков данных. О том, какие теоретические принципы лежат в их создании, расскажет пятая глава, которая послужит читателю кратким экскурсом в теорию реляционных баз данных.

В настоящее время наблюдается тенденция интегрирования языков программирования в пакеты быстрой визуальной разработки программ, но следует заметить, что в каком бы внешнем виде они не представлялись программисту, в их основе всегда лежат базовые структуры и алгоритмы. Значение «чистого» языка программирования по сей день не утрачено, поэтому изучению этого вопроса посвящено достаточно внимания в шестой главе. На наш взгляд, первичным является понимание логики программирования, а форма внешнего представления среды разработки программ вторична и не потребует значительных усилий для изучения. Здесь авторы не отошли от традиционной точки зрения большинства преподавателей информатики в том, что начинать изучение программирования удобно на примере языка *Паскаль*. Для читателей, желающий освоить визуальный способ разработки приложений для *Microsoft Windows*, в конце главы даны начальные сведения по программированию в среде *Visual Basic for Application*.

Современное информационное общество стало таковым благодаря появлению и развитию компьютерных сетей. Подробное описание всех существующих сетевых технологий является задачей, далеко выходящей за рамки главы учебника по информатике. В седьмой главе рассматриваются общие принципы построения компьютерных сетей и основы Internet-технологий.

Заключительная, восьмая глава посвящена актуальной, уже выделившейся в самостоятельное научное направление, задаче информационной безопасности.

ГЛАВА 1

ПРЕДМЕТ И ЗАДАЧИ ИНФОРМАТИКИ

1.1. Основные понятия информатики

▶ *Информатика* — это наука, изучающая структуру и наиболее общие свойства информации, её поиск, хранение, передачу и обработку с применением ЭВМ.

Термин «информатика» происходит от французских слов *information* (информация) и *automatique* (автоматика). Широкое распространение получил также англоязычный вариант этого термина — «*computer science*», что означает буквально «компьютерная наука».

Объектами изучения информатики являются технические, программные и алгоритмические средства. Технические средства представляют собой аппаратуру компьютеров, в английском языке они обозначаются словом *hardware*. Для обозначения программных средств, под которыми понимается совокупность всех программ, используемых компьютерами, и область деятельности по их разработке и применению, используется слово *software*, которое подчеркивает способность программного обеспечения модифицироваться, приспосабливаться и развиваться. Программированию задачи всегда предшествует разработка алгоритма ее решения в виде последовательности действий, ведущих от исходных данных к искомому результату. Для обозначения части информатики, связанной с разработкой алгоритмов и изучением методов и приемов их построения, применяют термин *brainware* (от англ. *brain* — интеллект).

Основная *цель* исследований в области информатики состоит в разработке способов решения задач информационной обработки на компьютерах, а также в разработке, модернизации, организации и эксплуатации вычислительных систем. Кроме того, достижения этой «компьютерной науки» имеют практически неограниченный спектр применения в прикладных областях. Вот лишь некоторые научные направления, использующие методы и средства информатики:

- математическое и имитационное моделирование, методы вычислительного эксперимента и их применение к фундаментальным и прикладным исследованиям в различных областях знаний;

- искусственный интеллект — раздел информатики, изучающий алгоритмическую реализацию человеческих способов решения задач. Иными словами, в рамках искусственного интеллекта изучают способы решения компьютером задач, не имеющих явного алгоритмического решения;
- биоинформатика, изучающая информационные процессы в биологических системах;
- социальная информатика, изучающая процессы информатизации общества;
- машинная графика, анимация, средства мультимедиа;
- системы автоматизированного проектирования;
- телекоммуникационные системы и сети, в том числе, глобальные компьютерные сети;
- разнообразные приложения, охватывающие производство, науку, образование, медицину, торговлю, сельское хозяйство и все другие виды хозяйственной и общественной деятельности.

Формирование моделей информатики нацелено на представление определенных структур, взаимодействий и процессов в какой-либо области применения (предметной области) с помощью формальных средств — таких как структуры данных, языки программирования или логические формулы.

Центральным понятием информатики является *информация*. Этот термин происходит от латинского слова «*informatio*», что означает сведения, разъяснения, изложение. Несмотря на широкое распространение этого термина, понятие информации является одним из самых дискуссионных в науке. В настоящее время наука пытается найти общие свойства и закономерности, присущие многогранному понятию *информация*, но пока оно во многом остается интуитивным и получает различные смысловые наполнения в различных отраслях человеческой деятельности:

- в обиходе информацией называют любые данные или сведения, которые кого-либо интересуют, например, сообщение о каких-либо событиях, о чьей-либо деятельности и т. п. «*Информировать*» в этом смысле означает «*сообщить нечто, неизвестное раньше*»;
- в технике под информацией понимают сообщения, передаваемые в форме знаков или сигналов;
- в кибернетике под информацией понимает ту часть знаний, которая используется для ориентирования, активного действия, управления, то есть в целях сохранения, совершенствования, развития системы (Н. Винер).

Клод Шеннон, американский учёный, заложивший основы теории информации — науки, изучающей процессы, связанные с передачей, приёмом, преобразованием и хранением информации, — рассматривает информацию как снятую неопределенность наших знаний о чем-либо.

Норберт Винер, «отец» кибернетики, сформулировал следующее определение: «Информация — это обозначение содержания, полученного из внешнего мира в процессе нашего приспособления к нему и приспособления к нему наших чувств».

Существует еще множество определений.

▶ Применительно к компьютерной обработке данных под *информацией* понимают некоторую последовательность символических обозначений (букв, цифр, закодированных графических образов и звуков и т. п.), несущую смысловую нагрузку и представленную в понятном компьютеру виде.

Информация может быть представлена в виде: текстов, рисунков, чертежей, фотографий; световых или звуковых сигналов; радиоволн; электрических и нервных импульсов; магнитных записей; хромосом, посредством которых передаются по наследству признаки и свойства организмов и т. д.

Среди основных *свойств* информации можно выделить следующие: достоверность; полнота; ценность; своевременность; доступность; краткость и др.

Одну и ту же информацию можно представить и передать по-разному (слова, жесты, сигналы, азбука Морзе (точки и тире), изображения и т. д.). В современной вычислительной технике информация чаще всего кодируется с помощью последовательностей сигналов всего двух видов, что определяется реализацией аппаратуры ЭВМ. В основе схемотехники компьютера лежит использование двоичного элемента для хранения данных — триггера, имеющего два устойчивых состояния. Эти состояния обозначают цифрами «0» и «1», а такое кодирование называют *двоичным кодированием.* В качестве единицы информации принят один *бит* (от *англ. bit* — ***bi**nary digit* — двоичная цифра).

В теории информации битом называют количество информации, необходимое для различения двух равновероятных сообщений.

В вычислительной технике бит — это наименьшая единица измерения памяти компьютера, необходимая для хранения одного из двух знаков «0» или «1», используемых для внутримашинного представления данных и команд.

С помощью одного бита можно представить всего два понятия, например, «истина» или «ложь», «черное» или «белое». На практике приме-

няются более крупные единицы. Восемь бит составляют один *байт* (от англ. *BinarY Term*). Именно восемь бит требуется для того, чтобы закодировать любой из 256 символов алфавита клавиатуры компьютера ($256 = 2^8$).

Широко используются также следующие производные единицы информации:

- 1 килобайт (кбайт) = 1024 байт = 2^{10} байт;
- 1 мегабайт (Мбайт) = 1024 кбайт = 2^{20} байт;
- 1 гигабайт (Гбайт) = 1024 Мбайт = 2^{30} байт.

В последнее время в связи с увеличением объёмов обрабатываемой информации входят в употребление такие единицы, как:

- 1 терабайт = 1024 Гбайт = 2^{40} байт,
- 1 петабайт = 1024 Тбайт = 2^{50} байт

и еще более крупные: экзабайт, зеттабайт, йоттабайт.

С информатикой, как правило, связывают такие понятия, как информационные процессы, технологии и ресурсы. *Информационными процессами* называют процессы передачи, накопления и обработки информации в общении людей, в живых организмах, технических устройствах и жизни общества. Под *обработкой информации* в информатике понимают любое преобразование информации из одного вида в другой, производимое по строгим формальным правилам. Обработка информации на ЭВМ обычно состоит в выполнении огромного числа элементарных технических операций. Технологии накопления, обработки и передачи информации с использованием определенных технических средств (прежде всего ЭВМ) получили название *информационных технологий* (последние иногда выделяют в отдельную дисциплину применительно к конкретной предметной области). *Информационные ресурсы* — это идеи человечества и указания по их реализации, накопленные в форме, позволяющей воспроизводить их. Это книги, статьи, патенты, диссертации, научно-исследовательская и опытно-конструкторская документация, технические переводы, данные о передовом производственном опыте и др.

Вообще говоря, роль информатики в развитии общества огромна. Появление ЭВМ — универсальной машины для обработки информации означало революцию в области информационных технологий. Эта революция, следующая за революциями в овладении веществом и энергией, затрагивает и коренным образом преобразует не только сферу материального производства, но и интеллектуальную, духовную сферы жизни. Прогрессивное увеличение возможностей компьютерной техники, развитие информационных сетей, создание новых информационных технологий приводят к значительным изменениям во всех сферах общества: в производстве, науке, образовании, медицине и т. д.

1.2. История развития вычислительной техники

Людям всегда была свойственна потребность в выражении и запоминании информации об окружающем их мире — так появилась устная речь, письменность, книгопечатание, живопись, фотография, радио, телевидение... Начиная с последней трети XX в. стали говорить об «информационном взрыве», называя этими словами бурный рост объемов и потоков информации. Он произошел на фоне традиционных методов обработки информации с помощью бумаги и ручки, что привело к информационному кризису. Возникло противоречие между быстро возрастающими объемами и потоками информации, потребностями общества в ее обработке для повышения уровня производства и жизни и ограниченными возможностями человека, использующего при работе с информацией традиционные средства. Это противоречие стало негативно сказываться на темпах экономического развития и научно-технического прогресса. Начался постепенный переход к информационному обществу, в котором на основе овладения информацией о самых различных процессах и явлениях можно эффективно и оптимально строить любую деятельность. Важно, что в информационном обществе повышается качество не только потребления, но и производства. Человек, использующий новые информационные технологии, имеет гораздо лучшие условия труда, труд становится творческим и интеллектуальным. Важное место в этом процессе заняла новая научная дисциплина — *кибернетика* — наука об управлении и связи в живом организме, машине, обществе; наука, центральным понятием которой является информация. Кибернетика породила новый системно-информационный взгляд на природу.

В качестве средства для хранения переработки и передачи информации научно-технический прогресс предложил обществу компьютер (электронно-вычислительную машину, ЭВМ). А в качестве критериев развитости информационного общества можно взять три: наличие компьютеров, существование развитого рынка программного обеспечения и функционирование компьютерных информационных сетей. Причем важно наличие таких компьютеров, которые были бы надежны, недороги, с богатыми аппаратными и программными возможностями. Именно к таким компьютерам наиболее приблизились последние модели ЭВМ.

Но вычислительная техника не сразу достигла такого уровня. В ее развитии отмечают предысторию и четыре поколения ЭВМ. Предыстория начинается с глубокой древности, с различных счет, а первая счетная машина появилась лишь в 1642 г. Ее изобрел французский математик, физик, философ и богослов Блез Паскаль. Построенная на основе

зубчатых колес, она могла суммировать десятичные числа. Все четыре арифметических действия выполняла машина, созданная в 1673 г. немецким математиком и философом Готфридом Вильгельмом Лейбницем. Она стала прототипом арифмометров, использовавшихся с 1820 г. до 60-х гг. XX в.

Наряду с устройствами, предназначенными для вычислений, развивались и механизмы для автоматической работы по заданной программе (музыкальные автоматы, шарманки, часы с боем и т. п.). В шарманку, например, помещали диски с расположенными по-разному штырьками — в зависимости от их расположения звучала та или иная мелодия. В ткацком станке Жаккарда узор ткани задавался с помощью дырочек в тонких картонных картах (перфокартах). Для смены узора достаточно было по-другому пробить дырочки в перфокарте.

Впервые соединил идею механической арифметической машины Лейбница с идеей программного управления английский математик Чарльз Беббидж в 1822 г. Он разработал проект программно-управляемой счетной машины, названной им «аналитической», которая имела арифметическое устройство, устройства управления, ввода и печати (использовалась десятичная система счисления). В машине была предусмотрена память для хранения 1000 чисел по 50 десятичных знаков; арифметические операции выполнялись в соответствии с программой, записанной на жаккардовых перфокартах. В программе можно было задавать автоматическое повторение группы арифметических операций, а также выполнение группы операций только при определенном условии. К сожалению, этот проект опережал технические возможности своего времени и не был реализован.

Лишь в 1941 г. ученому Конраду Цузе удалось создать программируемую цифровую счетную машину, причем на основе *электромеханических реле*, которые могут пребывать в одном из двух устойчивых состояний: «включено» и «выключено». Это технически гораздо проще, чем пытаться реализовать десять различных состояний, то есть опираться на обработку информации на основе десятичной, а не двоичной системы счисления. Но работы Цузе так и не были опубликованы. В США в 1943 г. на предприятии фирмы IBM американец Говард Эйкен создал первую программно-управляемую аналитическую машину «МАРК-1», позволявшую проводить вычисления в сотни раз быстрее, чем вручную (с помощью арифмометра) и реально использовавшуюся для военных расчётов. Однако электромеханические реле работали весьма медленно и недостаточно надежно, поэтому, начиная с 1943 г., в США группа специалистов Пенсильванского университета под руководством Джона П.

Эккерта и Джона У. Моукли начала конструировать компьютер ENIAC (*The Electronic Numerical Integrator and Calculator*) на основе электронных ламп. Созданный ими компьютер работал в тысячу раз быстрее, чем «МАРК-1». Однако обнаружилось, что большую часть времени этот компьютер простаивал — ведь для задания метода расчётов (программы) в этом компьютере приходилось в течение нескольких часов или даже дней подсоединять нужным образом провода. Сам расчёт после этого мог занять всего лишь несколько минут или даже секунд.

Для упрощения и ускорения процесса задания программ Моукли и Экерт сконструировали новый компьютер, который мог хранить программу в своей памяти. В 1945 г. к работе был привлечен математик **Джон фон Нейман.** Он сформулировал общие принципы функционирования компьютеров как универсальных вычислительных устройств. Первый компьютер, в котором были воплощены эти принципы, был построен в 1949 г. английским исследователем Морисом Уилксом.

В середине 40 гг. XX в. произошел коренной переворот в вычислительной технике — появились вычислительные машины, в которых применялись уже не электромеханические, а электронные элементы. Предпосылкой послужило изобретение триггеров, способных реализовывать принцип *двоичной системы счисления.* Первая в Европе ЭВМ была создана в СССР в 1951 г. под руководством академика С.А. Лебедева и называлась МЭСМ (*Малая Электронная Счетная Машина*). После появления транзисторов наиболее трудоемкой операцией при производстве компьютеров было соединение и спайка транзисторов для создания электронных схем. Но в 1959 г. Роберт Нойс (будущий основатель фирмы *Intel*) изобрел способ, позволяющий создавать на одной пластине кремния транзисторы и все необходимые соединения между ними. Полученные электронные схемы стали называться *интегральными схемами* или *чипами.* В 1968 г. фирма Burrougrhs выпустила первый компьютер на интегральных схемах, а в 1970 г. фирма *Intel* начала продавать интегральные схемы памяти.

Историю развития ЭВМ принято разделять на поколения. Точной даты смены поколений вычислительных машин сегодня установить невозможно, поскольку черты каждого следующего поколения развивались в недрах предыдущего.

ЭВМ первого поколения (конец 50-х — начало 60-х гг.) — это ламповые машины, используемые для решения научно-технических и инженерных задач, требующих значительного объема вычислений. Быстродействие этих машин составляло сотни и тысячи операций в секунду, их обслуживали десятки инженеров и программистов-математиков. Лам-

повые машины потребляли значительное количество электроэнергии, для них требовались большие помещения. Безотказная работа этих машин исчислялась часами, а иногда лишь десятками минут.

Элементной базой *ЭВМ второго поколения* (начало 60 гг.) были полупроводниковые приборы: транзисторы, диоды и пр. Безотказная работа увеличилась до нескольких сотен часов. Производительность составляла десятки и сотни тысяч операций в секунду.

ЭВМ третьего поколения (конец 60-х — начало 70-х гг.) характеризовались применением электронных микросхем с относительно низкой степенью интеграции активных элементов (до нескольких десятков в одной микросхеме). Благодаря значительному уменьшению количества паек, отрицательно влияющих на надежность электронных устройств, а также применению автоматизированной технологии изготовления, среднее время безотказной работы ЭВМ возросло до нескольких тысяч часов. Быстродействие машин увеличилось до нескольких сотен тысяч операций в секунду. Первая ЭВМ на интегральных схемах, содержащая более 500 схем малой интеграции, была изготовлена в 1961 г. В 1962 г. была выпущена первая серийная ЭВМ третьего поколения. Первенство в разработке подобных машин, начиная с ЭВМ модели ЮМ-360, принадлежало одной из самых известных в мире фирм-производителей вычислительной техники — американской *IBM* (*International Business Machines Corporation*). Помимо значительного улучшения основных параметров ЭВМ, третье поколение характеризовалось дальнейшим расширением области применения и существенным упрощением процедуры общения оператора с вычислительной машиной.

ЭВМ четвертого поколения (середина 70-х гг. и по настоящее время) имеют в качестве элементной базы большие интегральные схемы (БИС) со степенью интеграции от 100 до 1000 активных элементов на одну микросхему и сверхбольшие интегральные схемы (СБИС). Произошло дальнейшее упрощение взаимодействия человека с ЭВМ за счет совершенствования *языков программирования*. Применение БИС и СБИС позволило реализовать некоторые функции программ аппаратными средствами, что способствовало значительному ускорению процесса вычислений. В рамках четвертого поколения были разработаны такие широко известные средства вычислительной техники, как микрокалькуляторы.

К *ЭВМ пятого поколения* принято относить те вычислительные машины, которые, по сравнению с предыдущими поколениями, обладают качественно новыми способами взаимодействия пользователя ЭВМ с помощью речевых сообщений и графических изображений, способностью вычислительных систем к самообучению, логической и ассоциатив-

ной обработке информации, диалогом с пользователем в форме вопросов и ответов; способностью системы «понимать» содержимое баз данных, которые при этом превращаются в «базы знаний», и использовать их при решении задач. Быстродействие ЭВМ пятого поколения — несколько миллиардов операций в секунду.

Появление персональных компьютеров

В августе 1981 г. компания IBM стала выпускать компьютеры, называемые аббревиатурой IBM PC. С того времени название, данное этой компанией для своего электронного устройства (PC — *Personal Computer*, Персональный Компьютер, ПК) начинает прочно закрепляться в научно-технической литературе. Основным отличием ПК от известных высокопроизводительных ЭВМ того времени стала возможность индивидуального взаимодействия научно-технических работников со средствами вычислительной техники. В дальнейшем появилась возможность приобретения IBM PC по доступной цене для персональное использования во всех сферах человеческой деятельности.

Основной предпосылкой для создания персональных компьютеров явилось изобретение интегральных схем. Благодаря развитию технологии производства интегральных схем, количество транзисторов, размещаемых в них, постоянно увеличивалось. Вначале микропроцессоры использовались в различных специализированных устройствах, например, калькуляторах. В 1970 г. был сделан знаменательный шаг на пути к персональному компьютеру — фирма *Intel* сконструировала интегральную схему, аналогичную по своим функциям центральному процессору большого компьютера.

Так появился первый *микропроцессор Intel*-4004, который был выпущен в продажу в 1971 г. Это был настоящий прорыв, так как микропроцессор *Intel*-4004 размером менее 3 см был производительнее гигантской машины ENIAC. Правда, возможности Intel-4004 были куда скромнее, чем у центрального процессора больших компьютеров того времени, так как он работал гораздо медленнее и мог обрабатывать одновременно только 4 *бита* информации за один такт со скоростью 60 000 операций в секунду (процессоры больших компьютеров обрабатывали 16 или 32 *бита* одновременно), но и стоил он в десятки тысяч раз дешевле. В 1973 г. фирма *Intel* выпустила 8-битовый микропроцессор *Intel-8008*. Он требовал, по меньшей мере, 20 микросхем поддержки, зато мог выполнять 45 команд со скоростью 300 000 операций в секунду и адресовать 16 кбайт памяти. В 1974 г. появилась усовершенствованная вер-

сия *Intel-8080* (отечественный аналог — микропроцессорный комплект КР580), которая до конца 70-х годов стала стандартом для микрокомпьютерной индустрии. С точки зрения аппаратуры он был гораздо проще и в применении: требовал лишь 6 микросхем поддержки, выполнял 75 команд, обладал в десять раз большим быстродействием по сравнению с 8008, адресовал 64 кбайт памяти.

В начале 1975 г. появился первый коммерчески распространяемый персональный компьютер *Альтаир-8800* на основе микропроцессора *Intel-8080*. Хотя возможности его были весьма ограничены (оперативная память составляла всего 256 байт, клавиатура и экран отсутствовали), его появление было встречено с большим энтузиазмом: в первые же месяцы было продано несколько тысяч комплектов машин. Покупатели снабжали этот ПК дополнительными устройствами: монитором для вывода информации, клавиатурой, блоками расширения памяти и т. д. Вскоре эти устройства стали выпускаться другими фирмами.

В конце 1975 г. Пол Аллен и Билл Гейтс (будущие основатели фирмы Microsoft) создали для компьютера «Альтаир» интерпретатор языка *Basic*, что позволило пользователям достаточно просто общаться с компьютером и легко писать для него программы. Это также способствовало популярности персональных компьютеров.

Наиболее распространёнными в мире персональными ЭВМ являются *IBM*-совместимые компьютеры.

Открытая архитектура IBM PC. Если бы *IBM PC* был сделан так же, как другие существовавшие во время его появления компьютеры, он бы устарел через два-три года. Однако с компьютерами *IBM PC* получилось по-другому. Фирма *IBM* не сделала свой компьютер единым неразъемным устройством и не стала защищать его конструкцию патентами. Наоборот, она собрала компьютер из независимо изготовленных частей и не стала держать спецификации этих частей и способы их соединения в секрете. Принципы конструкции *IBM PC* были доступны всем желающим. Этот подход, называемый *принципом открытой архитектуры*, обеспечил потрясающий успех компьютеру *IBM PC*, хотя и лишил фирму *IBM* возможности единолично пользоваться плодами этого успеха. Именно открытость архитектуры *IBM PC* повлияла на развитие ПК. Идея открытой архитектуры пришлась по вкусу многим фирмам, которые стали выпускать дополнительные платы и отдельные элементы ЭВМ, быстро улучшая исходную модель ПК.

Существует и другое направление развития персональных ЭВМ, к которому относятся компьютеры, выпускаемые фирмой *Apple Computer* и образующее ряд ПК — *Macintosh.* Несмотря на то, что количество

ежегодно выпускаемой продукции фирмы *Apple Computer* значительно уступает объему производства *IBM*-совместимых компьютеров, в США ПК *Macintosh* является одним из самых популярных.

1.3. Представление информации в компьютере

Современные средства вычислительной техники способны обрабатывать различные виды информации: числовую, текстовую, графическую, звуковую. Каждый из этих видов имеет свои способы кодирования для представления внутри ЭВМ. Тем не менее с точки зрения аппаратуры компьютера любая информация — это последовательность сигналов, кодируемая с помощью нулей и единиц. Таким образом, все вычисления и преобразования информации в компьютере происходят в двоичной системе счисления.

1.3.1. Системы счисления

▶ *Система счисления* — это совокупность приемов обозначения (записи) чисел.

Наиболее часто используются позиционные системы счисления. В них значение каждой цифры в изображении числа зависит от ее положения (позиции) в последовательности цифр, изображающей число. В непозиционных системах счисления значение цифры (знака) не зависит от места, которое она занимает в числе (например, римская система счисления).

▶ *Основание системы счисления* — количество различных цифр (знаков), используемых для представления чисел в данной системе.

Принято основание системы счисления указывать в виде нижнего индекса рядом с числом. Например, запись 34_{10} означает, что число 34 представлено в системе счисления по основанию 10.

Пусть необходимо представить число X (>0) в системе счисления с основанием q в виде полинома:

$$X = a_{n-1} \cdot q^{n-1} + \ldots + a_1 \cdot q^1 + \ldots a_0 \cdot q^0 + \ldots a_{-m} \cdot q^{-m}, \qquad (1.1)$$

где a_i — цифры системы счисления;

n — число цифр в целой части числа X;

m — число цифр в дробной части числа X.

Основание часто указывают в виде индекса либо определенного символа в зависимости от системы программирования. На практике принята сокращенная запись чисел:

$X = a_{n-1} \ldots a_1 \ldots a_0 \ldots a_{-m}$.

Например, число 863,52 (q=10, n=3, m=2) представляется в виде:
$863{,}52 = 8 \cdot 10^2 + 6 \cdot 10^1 + 3 \cdot 10^0 + 5 \cdot 10^{-1} + 2 \cdot 10^{-2}$.

Наиболее распространены двоичная, десятичная и шестнадцатеричная системы счисления.

Двоичная система счисления

Основание q = 2, используются цифры 0 и 1.

Система счисления по основанию 2 аналогична десятичной, за исключением того, что разряды числа в ней соответствуют не степеням 10, а степеням 2. Значения чисел, большие 1, представляются многоразрядными числами точно так же, как в десятичной системе представляются значения, большие 9. Любая цифра двоичного числа является битом. Каждый бит соответствует какой-либо степени 2. Например, разложение двоичного числа 1001,1 по степеням 2 (в виде полинома) имеет вид:

$1001{,}1_2 = 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 + 1 \cdot 2^{-1} = 9{,}5_{10}$.

Арифметические операции в двоичной системе счисления выполняются по следующим правилам:

Сложение:	0 + 0 = 0;	0 + 1 = 1;	1 + 0 = 1;	1 + 1 = 10;
Вычитание:	0 – 0 = 0;	1 – 0 = 1;	1 – 1 = 0;	10 – 1 = 1;
Умножение:	0 · 0 = 0;	0 · 1 = 0;	1 · 0 = 0;	1 · 1 = 1.

Примеры:

11101,01 + 110,11 = 100100,00;

1001 – 110 = 11;

110 · 101 = 11110.

Как уже отмечалось выше, использование только двух символов для кодирования информации дало возможность технической реализации компьютера на основе использования устройств, имеющих всего два устойчивых состояния.

Шестнадцатеричная система счисления

Основание q = 16, используются цифры 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, а также буквы латинского алфавита A, B, C, D, E, F.

Пример:

$247{,}6_{16} = 2 \cdot 16^2 + 4 \cdot 16^1 + 7 \cdot 16^0 + 6 \cdot 16^{-1} = 583{,}375_{10}$.

Каждая цифра в 16-ричном представлении может иметь значение 0...15, каждый разряд соответствует степени 16. 16-ричное представление является компактным и удобным методом записи двоичной информации, так как для перевода из двоичной системы счисления в шестнад-

цатеричную и обратно не требуется специальных процедур. В табл. 1.1 дано представление некоторых целых чисел в различных системах счисления.

Таблица 1.1

Соответствие чисел в различных системах счисления

10	0	1	2	3	4	5	6	7	8
2	0000	0001	0010	0011	0100	0101	0110	0111	1000
16	0	1	2	3	4	5	6	7	8

10	9	10	11	12	13	14	15	16
2	1001	1010	1011	1100	1101	1110	1111	10000
16	9	A	B	C	D	E	F	10

1.3.1.1. Правила перевода чисел из одной системы счисления в другую

Перевод целого числа из системы счисления с основанием p=10 в систему счисления с основанием q

Разделить число X_{10} нацело на основание новой системы q до получения целого частного. Остаток от деления будет разрядом единиц (в новой системе счисления).

Полученное частное вновь делится на основание q до получения в остатке очередного разряда искомого числа.

Процесс продолжается до тех пор, пока частное не станет меньше основания q. Последнее частное и будет старшим разрядом числа в системе с основанием q.

При делении необходимо основание q новой системы записывать в исходной системе с основанием p, и само деление выполнять в исходной системе.

При наличии у числа дробной части сначала выполняется перевод целой части числа, а дробная часть может быть переведена в другую систему счисления с точностью до некоторого числа разрядов. При переводе дробной части числа её разряды умножаются на основание новой системы счисления, и процесс останавливается при достижении заданной точности представления результата.

Пример:

Перевести число 27_{10} в двоичную и шестнадцатеричную системы счисления.

1. p = 10, q = 2.

27 / 2 = 13 и 1 в остатке;
13 / 2 = 6 и 1 в остатке;
6 / 2 = 3 и 0 в остатке;
3 / 2 = 1 и 1 в остатке.
Следовательно, $27_{10} = 11011_2$.

2. p = 10, q = 16.
27 / 16 = 1 и 11 в остатке. Числу 11 в шестнадцатеричной системе счисления соответствует обозначение B, следовательно, $27_{10} = 1B_{16}$.

Перевод целого числа из системы счисления с основанием q в систему счисления с основанием p=10

Пусть известна запись числа X в q-й системе: $X = a_{n-1}... a_1... a_0... a_{-m}$. Число X можно представить в виде полинома (1.1).

Для получения представления числа X в десятичной системе счисления необходимо:

1. Каждый из коэффициентов и основание системы q представить в 10-ричной системе счисления.
2. Выполнить действия по правилам десятичной арифметики.
3. Полученное значение полинома даст значение числа X в десятичной системе счисления.

Пример:

Осуществить проверку предыдущего примера: перевести полученные в двоичной и шестнадцатеричной системах счисления числа в десятичную.

1. $11011_2 = 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 27_{10}$.
2. $1B_{16} = 1 \cdot 16^1 + 11 \cdot 16^0 = 27_{10}$.

Для преобразования шестнадцатеричных чисел в двоичную форму записи и обратно используются более простые правила. Перевод шестнадцатеричного числа в двоичную форму записи осуществляется следующим образом: каждая цифра шестнадцатеричного числа заменяется соответствующим четырехразрядным двоичным числом, при этом отбрасываются ненужные нули (крайние слева в целой части и крайние справа — в дробной).

Пример:

$$\underbrace{6}_{0110}\ \underbrace{A}_{1010}\ \underbrace{2}_{0010},\ \underbrace{E}_{1110}{}_{16} = 011010100010{,}1110_2 = 11010100010{,}111_2.$$

Переход от двоичной к шестнадцатеричной системе осуществляется в таком порядке: вначале, двигаясь от запятой влево и вправо, разбивают двоичное число на группы по четыре разряда (при необходимости крайние левая и правая группа заполняются нулями), затем каждая группа из четырех разрядов заменяется соответствующей шестнадцатеричной цифрой.

Примеры:

$$1110{,}11_2 = \underbrace{1110}_{E},\ \underbrace{1100}_{C} = E{,}C_{16}$$

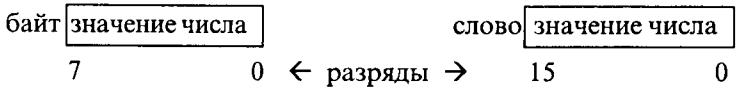
$$101101{,}1_2 = \underbrace{0010}_{2}\ \underbrace{1101}_{D},\ \underbrace{1000}_{8} = 2D{,}8_{16}$$

1.3.2. Формы представления данных

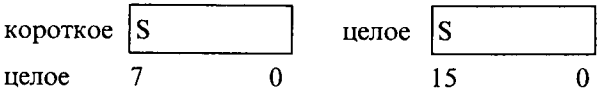
Представление целых чисел

Для представления целых чисел в микропроцессоре выделяется байт, слово, двойное слово и т. д. Эта величина, как правило, не превышает разрядности основного машинного слова микропроцессора.

Для представления беззнаковых типов используются все имеющиеся разряды, например:



Для представления знаковых типов знак кодируется в старшем бите. Если старший бит равен 0 — число положительное, если равен 1 — число отрицательное.



$S = \begin{Bmatrix} 0 \\ 1 \end{Bmatrix}$ — знак числа

Положительные числа хранятся в памяти и вступают в операции в *прямом коде* (т. е. обычном двоичном представлении числа). Отрицательные числа хранятся в памяти компьютера в *дополнительном коде.*

Правило получения дополнительного кода

Модуль отрицательного числа записывается в прямом коде, «прижатым» вправо. В неиспользуемые старшие биты записываются нули.

Формируется обратный код битов (кроме знакового разряда). Для этого 0 заменяется на 1, а 1 — на 0.

К обратному коду числа прибавляется 1.

Таким образом, диапазон представления знаковых чисел с помощью N разрядов определяется как 2^{N-1}, а беззнаковых — 2^N.

Пример:

Получим 8-разрядный дополнительный код числа –52:

00110100 — число |-52|=52 в прямом коде

11001011 — число -52 в обратном коде

11001100 — число -52 в дополнительном коде

Представление символьных данных

В персональных компьютерах и телекоммуникационных системах применяется международный байтовый код *ASCII* (*American Standard Code for Information Interchange*). В справочной литературе имеются таблицы кодов *ASCII*, где каждому символу соответствует код, состоящий из 8 бит (или 1 байта).

Первые 32 кода (от 0 до $1F_{16}$) в *ASCII* являются управляющими. Они служат для представления символов, имеющих специальное назначение. Другие символы (20_{16} ... $7F_{16}$) используются для кодирования цифр, букв латинского алфавита, знаков пунктуации и арифметических операций. Остальные коды (80_{16} ... $0FF_{16}$) означают расширение стандарта *ASCII* и в разных моделях ЭВМ реализуются по-разному, в частности, для кодирования символов кириллицы. Для этой части также имеются стандарты, например, для символов русского языка это *КОИ–8*.

Фирма Microsoft для операционной системы Windows разработала собственную русскую кодировку *ANSI*-1251.

В настоящее время получил распространение также международный стандарт кодировки — *Unicode*. Для представления каждого символа в нем отводится 2 байта. Такая длина кода обеспечивает включение в первичный алфавит 65536 знаков. Это, в свою очередь, позволяет создать и использовать единую для всех распространенных алфавитов кодовую таблицу.

Представление вещественных данных

Различают две формы представления вещественных чисел:

1) представления чисел с плавающей точкой,

2) представления чисел с фиксированной точкой.

Формой представления чисел с фиксированной точкой называется запись числа в виде последовательности цифр, входящих в изображение данного числа, где в качестве разделителя целой и дробной части используется точка (запятая).

Примеры:

11.9943_{10}; 111.01001_2.

Формой представления чисел с плавающей точкой представление числа в виде двух множителей $N = M \times B^p$, где M — мантисса числа; B — основание системы счисления; p — порядок, записываемый в выбранной системе счисления.

Если M удовлетворяет неравенству $1/B \le M < 1$, то число N и само представление *нормализованное*. В любой системе счисления это неравенство означает, что мантисса числа меньше 1, и ее первая цифра после запятой отлична от нуля. При этом порядок числа может быть как положительным, так и отрицательным. Порядок указывает, на какое количество позиций и в каком направлении должна «переплыть», т. е. сместиться десятичная точка в мантиссе. Отсюда название «плавающая точка».

Пример:

Десятичные числа 0.00379; 0.0379; 0.379; 3.79; 37.9 в нормализованном виде записываются соответственно: $0.379 \cdot 10^{-2}$; $0.379 \cdot 10^{-1}$; 0.379; $0.379 \cdot 10^{1}$; $0.379 \cdot 10^{2}$.

В современных компьютерах используются обе формы представления чисел.

Число бит для хранения мантиссы и порядка зависит от типа данных с плавающей точкой.

В отличие от целых чисел, которые всегда представляются в памяти ЭВМ абсолютно точно, значения вещественных данных определяют числа лишь с некоторой конечной точностью, зависящей от внутреннего представления числа.

Алгоритм представления числа с плавающей запятой

1) перевести число из p-ичной системы счисления в двоичную;

2) представить двоичное число в нормализованной экспоненциальной форме;

3) рассчитать смещённый порядок числа;

4) разместить знак, порядок и мантиссу в соответствующие разряды сетки.

Пример:

Представить число — 25,625 в машинном виде с использованием 4-байтового представления (где 1 бит отводится под знак числа, 8 бит — под смещённый порядок, остальные биты — под мантиссу).

$25_{10}=100011_2$
$0{,}625_{10}=0{,}101_2$
$-25{,}625_{10}= -100011{,}101_2$
$-100011{,}101_2 = -1{,}00011101_2 \cdot 2^4$

В разряд S записывается 1_2. В разряды порядка P записывается кодовая комбинация 10000_2 (2^4). В разряды мантиссы M записывается кодовая комбинация 00011101_2.

Представление графической информации

Существует два способа представления графической информации: растровая и векторная графика.

Растровое изображение состоит из множества маленьких точек, у каждой из которых может быть свой цвет и яркость. Точки выстроены как в таблице: по строкам и столбцам. Из них получается изображение (рис. 1.1). Благодаря маленькому размеру, отдельные точки не видны (или малозаметны), и создаётся впечатление однородной картины. Минимальный элемент, из которого состоит растровое изображение, называется *пикселем* (от англ. *picture element*).

Для хранения растрового изображения в памяти компьютера необходимо хранить информацию о цвете каждого пикселя.

Растровый способ представления изображений прекрасно подходит для хранения фотографий и видеофрагментов.

Редактирование растровой графики заключается в изменении цветов пикселей. Это удобно в том случае, когда нужно изменить мелкие детали изображения или применить какой-либо визуальный эффект (например, эффект «размытия» изображения).

Однако изображение, представленное в растровом виде, не хранит никакой информации о форме объектов. Форма получается в мозгу человека за счёт разницы цветов соседних пикселей. Поэтому редактировать форму объектов, представленных растровым способом, достаточно сложно.

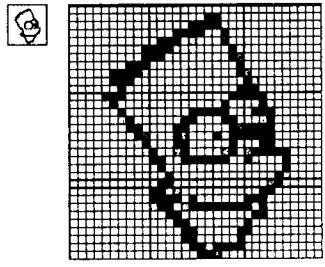


Рис. 1.1

Другой способ представления графической информации — *векторная* графика. Основными элементами векторной графики являются простые геомет-

рические фигуры, которые хранятся в памяти компьютера в виде математических формул и числовых параметров. Например, отрезок задаётся координатами первой и второй точки, а окружность — координатами центра и радиусом. Из простейших фигур складываются более сложные. Каждая фигура может иметь собственный цвет. Область, ограниченная несколькими линиями, может быть закрашена определенным цветом или особым способом (например, заштрихована). Простейшие элементы, из которых состоит векторное изображение, называются *примитивами*.

Обычно примитивами являются самые базовые геометрические объекты: точки и отрезки. Иногда к примитивам относят и другие фигуры: квадрат, окружность, прямоугольник, эллипс и т. д. Отрезок может быть как прямым, так и кривым (рис. 1.2). Прямые отрезки задаются координатами крайних точек, а для кривых отрезков задаются дополнительные параметры, которые определяют степень и форму кривизны. Отрезки могут иметь и другие параметры, например, цвет и толщину. Из отрезков можно получить различные фигуры, в том числе и замкнутые. Замкнутые фигуры могут иметь определённый цвет или стиль внутренней закраски. Совокупность фигур и отрезков может изображать какой-то объект, который тоже может иметь определённые параметры, например, название, размер, угол наклона.

Рис. 1.2

Таким образом, для хранения векторного изображения необходимо хранить координаты и дополнительные параметры примитивов, фигур и объектов и взаимосвязи между ними.

Представление звуковой информации

Приёмы и методы работы со звуковой информацией пришли в вычислительную технику наиболее поздно. К тому же, в отличие от числовых, текстовых и графических данных, у звукозаписей не было столь же длительной и проверенной истории кодирования. В итоге методы кодирования звуковой информации двоичным кодом далеки от стандартизации. Множество отдельных компаний разработали свои корпоративные стандарты, но среди них можно выделить два основных направления.

Метод *FM* (*Frequency Modulation*) основан та том, что, теоретически, любой сложный звук можно разложить на последовательность про-

стейших гармонических сигналов разных частот, каждый из которых представляет собой правильную синусоиду, а следовательно, может быть описан числовыми параметрами, то есть кодом. В природе звуковые сигналы имеют непрерывный спектр (являются аналоговыми). При таких преобразованиях неизбежны потери информации, связанные с методом кодирования, поэтому качество звукозаписи обычно получается не вполне удовлетворительным и соответствует качеству звучания простейших электромузыкальных инструментов с окрасом, характерным для электронной музыки. В то же время данный метод кодирования обеспечивает весьма компактный код, поэтому он нашёл применение ещё в те годы, когда ресурсы средств вычислительной техники были явно недостаточны.

Метод *таблично волнового* (*Wave-Table*) синтеза лучше соответствует современному уровню развития техники. В заранее подготовленных таблицах хранятся образцы звуков множества различных музыкальных инструментов. В технике такие образцы называют *сэмплами.* Числовые коды выражают тип инструмента, номер его модели, высоту тона, продолжительность и интенсивность звука, динамику его изменения, некоторые параметры среды, в которой происходит звучание, а также прочие параметры, характеризующие особенности звучания. Поскольку в качестве образцов используются реальные звуки, то его качество получается очень высоким и приближается к качеству звучания реальных музыкальных инструментов.

ГЛАВА 2


УСТРОЙСТВО ПЕРСОНАЛЬНОГО КОМПЬЮТЕРА

2.1. Принципы функционирования компьютера

2.1.1. Общие принципы

Классические принципы построения и функционирования ЭВМ известны как принципы **Джона фон Неймана**. Согласно им, во-первых, для представления данных должна быть использована двоичная система счисления. Во-вторых, программа также должна храниться в виде последовательности нулей и единиц, причем в той же самой памяти, что и данные, которые ею обрабатываются. Третий принцип — последовательное выполнение команд программы, а также адресация ячеек памяти, для того, чтобы возможен был непосредственный переход к любой из них (*адрес* — номер ячейки памяти, по которому осуществляется доступ к ней). И последний принцип заключается в равноправии ячеек памяти, хранящих как команды программы, так и данные, поэтому над любыми ячейками памяти ЭВМ можно производить одинаковые действия.

Структуру ЭВМ Дж. фон Нейман представлял четырьмя основными блоками:

- АЛУ — устройство, выполняющее арифметические и логические операции;
- УУ — устройство управления для организации выполнения программ;
- ЗУ — память для хранения программ и данных;
- ВУ — внешние устройства для ввода/вывода информации.

УУ управляет процессом обработки информации и содержит:

- генератор тактов;
- информацию о состоянии процесса (в специальных регистрах: операций, адресов, индексном, счетчике команд);
- оборудование для выработки управляющих сигналов для выполнения отдельных команд;
- усилители мощности (необходимы, т. к. нагрузка на шину данных не может быть большой, а к внешней шине может быть подключено множество физических устройств).

Собственно шаги преобразования информации, представленной в виде двоичных слов, реализует АЛУ. В нем выполняется весь перечень простейших команд (сдвиги, сложение, инверсия и др.) и хранение временных результатов. АЛУ содержит схемы выполнения арифметических и логических команд и регистры общего назначения. АЛУ выполняет все операции под воздействием сигналов УУ, то есть является *исполнительным устройством*. В результате работы АЛУ всегда вырабатываются «флаги» (например, был ли перенос в старший разряд, равен или не равен результат нулю, является ли результат отрицательным и т.д.).

Память компьютера должна состоять из некоторого количества пронумерованных ячеек, в каждой из которых могут находиться или обрабатываемые данные, или инструкции программ. Все ячейки памяти должны быть одинаково доступны для других устройств компьютера. На рис. 2.1 показаны связи между устройствами компьютера.

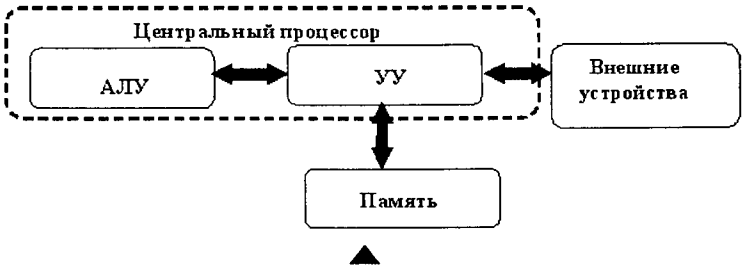


Рис. 2.1

Основные принципы работы компьютера следующие. В память компьютера из внешнего устройства считывается программа и располагается последовательно, согласно адресам загрузки. УУ считывает содержимое ячейки памяти, где находится первая инструкция (команда) программы, и организует ее выполнение. Эта команда может задавать выполнение арифметических или логических операций, чтение из памяти данных или запись результатов операций в память, ввод данных из внешнего устройства в память или вывод данных из памяти на внешнее устройство. Далее УУ может как выполнить команду из следующей ячейки памяти, так и перейти к другой ячейке, например, при выполнении некоторых условий. УУ может обмениваться информацией с оперативной памятью и внешними устройствами компьютера, а также приостанавливать выполнение программы.

Процесс выполнения программ обычно является непрерывным, но может прерываться для выполнения других срочных действий — в ответ на сигналы от внешних устройств, называемые *прерываниями.*

В современных компьютерах АЛУ и УУ объединены в один блок, называемый центральным процессором, либо микропроцессором (МП).

▶ Процессор (*Central Processing Unit — CPU*) является основой компьютера. Именно в нем производятся основные процессы вычислений и других преобразований информации. Одновременно с этим МП управляет ходом процесса вычислений.

Процессор, таким образом, является инструментальным средством, аппаратурой (*hardware* — «жесткое оборудование»), которое автономно управляет ходом вычислений и выполняет предусмотренные в компьютере операции преобразования данных. Степень интеграции определяется размером кристалла и количеством реализованных в нем транзисторов.

▶ Важнейшей характеристикой процессора является *тактовая частота* — величина, показывающая количество импульсов синхронизации начала тактов команд микропроцессора за секунду (каждая машинная команда микропроцессора может выполняться за определенное число тактов). В современных микропроцессорах она измеряется в гигагерцах (ГГц = 1 млрд. тактовых импульсов в 1 с) и определяет быстродействие компьютера.

▶ *Разрядность* процессоров определяется количеством двоичных разрядов, которое может обрабатываться процессором за 1 такт.

В процессоре всегда имеется определенное число внешних входных и выходных сигналов. Эти сигналы интерпретирует и генерирует УУ, связывая внутренние шины, регистры и прочие устройства МП с внешними устройствами (памятью и устройствами ввода/вывода). УУ вырабатывает также управляющие сигналы для АЛУ и переключателя устройств передачи данных.

▶ *Регистры* — запоминающие ячейки с очень малым временем доступа (то есть высоким быстродействием), которые служат для хранения значений и переработки информации.

В принципе, процессор может осуществлять все операции и непосредственно через память, но так как его быстродействие всегда гораздо выше, чем у памяти, такая работа МП была бы неэффективной. Поэтому процессору нужна своя, «внутренняя» память. Регистры — это временная, промежуточная память ограниченного объема. Если какие-то данные нужны для длительного хранения или дальнейшего использования, они могут быть вынесены, например, во внешнюю память. Каждый

из регистров имеет свой внутренний адрес, по которому осуществляется доступ к его содержимому. Содержимое регистров может быть произвольно (как команды, так и данные).

▶ *Шины* — служат для обмена сигналами между устройствами процессора.

В современных ПК принята классическая трехшинная архитектура, включающая:

- *шину данных* (двунаправленная) — по ней передаются данные. Таких шин может быть несколько (локальные шины от различных устройств). И УУ может обращаться одновременно к нескольким блокам за информацией;
- *шину адреса* (однонаправленная) — по ней процессор выставляет адрес памяти или устройств ввода/вывода;
- *шину управления* (имеются входные, выходные и реверсивные сигналы), которая позволяет процессору «разобраться» с данными и адресами. Это своеобразный указатель процессору, как действовать. И, наоборот, через шину управления сам процессор может указать, как действовать.

1.1.2. Начальная загрузка персонального компьютера

Устройство МП не позволяет в начальный момент времени выявить из потока, какая имеется информация: команда или данное. Для того чтобы система начала работу, необходимо какое-то оговоренное исходное ее состояние. При нажатии кнопки включения питания на системном блоке ПК его МП получает сигнал сброса *RESET* (возможно также нажатие кнопки *RESET*), все элементы памяти устанавливаются в логический «0», и первой всегда является команда чтения памяти по определенному адресу. Для «оживления» системы служит внешнее постоянное запоминающее устройство (ПЗУ), в котором содержатся первичные программы управления устройствами — *BIOS* (*Base Input Output System* — базовая система ввода-вывода). Таким образом, *BIOS* — это постоянная память, которая позволяет процессору инициализировать дисководы, клавиатуру и видеосистему. В функции *BIOS* также входит тестирование оперативной памяти (ОП) и других устройств, определение аппаратного состава компьютера и загрузка операционной системы. Далее загружается операционная система с дисковых устройств в ОП ПК. И только после этого машина готова к общению с пользователем.

1.1.3. Логическая структура ПК

Логическая структура ПК, дающая общее представление о входящих в состав компьютера устройствах и функциональных взаимосвязях между ними, изображена на рис. 2.2. Главная отличительная черта архитектуры ПК состоит в наличии *системной шины* (шины ввода-вывода), посредством которой микропроцессор взаимодействует и обменивается информацией с периферийными устройствами.

Архитектура с общей системной шиной обеспечивает простоту и дешевизну ПК, а также унифицирует алгоритмы взаимодействия устройств, облегчая программирование. С другой стороны, общая системная шина является узким местом ПК, потенциально ограничивая его производительность. Это объясняется, прежде всего, тем, что в каждый момент времени посредством системной шины могут обмениваться информацией только два устройства, остальные же вынуждены простаивать.

▶ *Основная память* ПК — совокупность постоянной или полупостоянной (содержащей *BIOS*) и оперативной памяти (ОЗУ, хранящей выполняемые программы и данные, непосредственно участвующие в операциях).

Это единственные устройства, к которым МП способен обращаться непосредственно, без использования системной шины как тракта передачи данных. Обмен информацией между микропроцессором и всеми другими — периферийными — устройствами производится через системную шину.

▶ Под *периферийным устройством* понимают любое устройство, конструктивно отделённое от его центральной части (МП и основной памяти), имеющее собственное управление и выполняющее запросы микропроцессора без его непосредственного вмешательства.

Таким образом, МП осуществляет только общее управление процессом обмена данными, а всю черновую работу выполняет периферийное устройство. Однако непосредственная организация обмена информацией между ОЗУ и периферийными устройствами выполняется МП. Именно через него перекачиваются все данные из ОЗУ в периферийное устройство и обратно. Поэтому МП во время обмена не способен выполнять никакую другую работу. В то же время в ПК имеется система прямого доступа к памяти, отчасти решающая эту проблему в отношении быстродействующих периферийных устройств — прежде всего дисковых накопителей, увеличивая к тому же скорость обмена (МП способен обмениваться только небольшими порциями данных, а система прямого

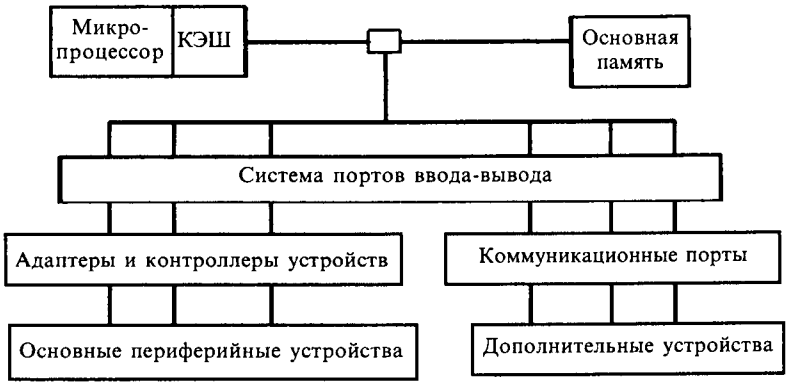


Рис. 2.2

доступа к памяти — целыми блоками). Если обмен происходит с использованием этой системы, МП только инициирует операцию обмена, после чего может перейти к другой работе.

По функциональному признаку периферийные устройства делятся на две основные группы — внешние запоминающие устройства и устройства ввода-вывода. По степени важности периферийные устройства можно разделить на основные (неотъемлемая часть любого ПК — монитор, клавиатура, хотя бы одно внешнее запоминающее устройство) и дополнительные.

Периферийные устройства присоединяются к системной шине не напрямую. Основные периферийные устройства подключаются через цепочку *адаптер* (*контроллер*) периферийного устройства — *порт* ввода-вывода.

▶ *Порт* ввода-вывода представляет собой программное, а зачастую и аппаратное устройство, предназначенное для обмена данными с периферийным устройством и для управления им.

Порт ввода-вывода играет роль «точки» на системной шине, к которой подключается адаптер (контроллер) периферийного устройства. Каждый порт имеет свой адрес, аналогичный адресу в ОЗУ, но содержащийся в другом адресном пространстве. Одному периферийному устройству может быть приписано несколько портов ввода-вывода, каждый из которых имеет своё назначение.

▶ В функции *адаптера* (*контроллера*) периферийного устройства входят:

1. Непосредственное управление периферийным устройством по запросам от микропроцессора.

2. Согласование *интерфейса* (способа взаимодействия) периферийного устройства с системной шиной.

Термины «адаптер» и «контроллер» близки. Однако контроллер несёт большую управляющую нагрузку, чем адаптер, реализующий, в основном, несложные согласования.

Дополнительные периферийные устройства могут подключаться к ПК через свои адаптеры (контроллеры), устанавливаемые в имеющиеся на системной плате гнёзда расширения, либо через коммуникационные порты, играющие роль адаптеров и поддерживающие стандартные интерфейсы. По сути, коммуникационный порт — это универсальный адаптер. Чтобы быть подключенным к коммуникационному порту некоторого типа, периферийное устройство должно обладать тем же интерфейсом.

Имеется два универсальных типа коммуникационных портов: последовательные и параллельные.

▶ *Последовательные* (*serial*) порты обеспечивают побитовый обмен информацией с медленнодействующими (мышь, джойстик, модем) или достаточно удалёнными периферийными устройствами.

▶ *Параллельные* (*parallel*) порты служат для обмена байтами с более быстродействующими периферийными устройствами (принтер, сканер, цифровые фото- и кинокамеры).

МП должен оперативно реагировать на различные события в ПК, происходящие в результате действий пользователя или без его ведома. Такими событиями могут быть: нажатие клавиши на клавиатуре, попытка деления на нуль, переполнение разрядной сетки, сбой питания, иные нарушения в работе оборудования, запланированные в программе обращения к ядру операционной системы и т. п.

▶ Необходимую реакцию на внешние по отношению к МП события обеспечивает *система прерываний*.

Обработка прерываний сводится к приостановке выполнения текущей последовательности команд (программы), вместо которой начинает интерпретироваться другая последовательность инструкций, соответствующая данному типу прерывания и называемая *обработчиком прерывания*. После её реализации выполнение прерванной программы может быть продолжено, если это возможно и/или целесообразно, что зависит от типа прерывания.

Прерывания делятся на следующие категории:

Внешние аппаратные прерывания, возникающие в результате событий, происходящих вне микропроцессора (например, нажатие клавиши на клавиатуре).

Внутренние аппаратные прерывания, вырабатываемые самим микропроцессором при выполнении программы (к прерыванию этой категории приводит, например, попытка деления на нуль).

Программные прерывания, инициируемые выполняемой программой по специальной команде, чтобы получить сервисные услуги операционной системы.

Таким образом, аппарат прерываний используется как для обеспечения асинхронной работы микропроцессора и периферийных устройств, так и для взаимодействия выполняемых программ с операционной системой. Адреса обработчиков прерываний содержатся в таблице векторов прерываний, размещаемой в начальных ячейках ОЗУ.

Сама системная шина представляет собой совокупность одно- и двунаправленных линий, логически объединяемых в следующие в группы:

- *шину данных* — для передачи данных в оба направления;
- *шину адреса*, с использованием которой адресуются порты ввода-вывода;
- *шину управления* — для передачи управляющих сигналов, таких, как «запись в порт», «чтение из порта», сигналов прерываний и т. п.

Физически шины адреса и данных могут мультиплексироваться (совмещаться).

2.2. Состав персонального компьютера и периферийные устройства

2.2.1. Состав ПК

Обычно ПК семейства IBM PC состоят из следующих блоков:

- *системного блока*, в котором размещаются:
  - электронные схемы, управляющие работой ПК (микропроцессор, оперативная память, контроллеры устройств и т. д.);
  - видеоадаптера, подготавливающего информацию для вывода на монитор;
  - блок питания;
  - накопители для гибких магнитных дисков (дисковод, *FDD*);
  - накопитель на жестком магнитном диске (винчестер, *HDD*);
  - устройство чтения/записи компакт-дисков (*CD-ROM, DVD-ROM, CD-RW, DVD-RW*);
- *монитора* — для отображения текстовой и графической информации;
- *клавиатуры* и *мыши* — для управления компьютером и ввода в него информации.

МП ПК осуществляет выполнение программ и управляет работой остальных устройств ПК. Скорость его работы во многом определяет быстродействие всего ПК. Микропроцессоры отличаются друг от друга типом (моделью) и тактовой частотой. Чем выше модель процессора, тем меньше тактов требуется для выполнения одних и тех же операций.

Начиная с МП 80486, используются специальные устройства охлаждения: радиатор, прилегающий к процессору, и вентилятор, образующие единый конструктивный блок. Тип вентилятора (мощность, тип радиатора и тип крепления к МП) должен соответствовать типу охлаждаемого процессора.

▶ *Системная плата*, изображенная на рис. 2.3 (ее также называют материнской, от английского *Mother Board*) — диэлектрическая пластина с многослойным печатным монтажом, на которой, как минимум, имеются:

- микропроцессор;
- модуль *BIOS*;
- модули оперативной памяти;
- системная шина;
- гнёзда расширения ресурсов (слоты);
- вспомогательные микросхемы, в том числе кварцевый резонатор, вырабатывающий синхроимпульсы для работы ПК.

Модуль *BIOS* находится в микросхеме энергонезависимой памяти (*CMOS*-памяти). В *CMOS*-памяти записана конфигурация аппаратных средств компьютера и некоторая другая информация. Она питается от небольшой батареи, и поэтому ее содержимое не стирается даже после выключения питания компьютера. Если системная плата содержит встроенный контроллер накопителей на жестких дисках, на ней также располагается разъем для подключения индикатора жесткого диска. Этот индикатор светится в момент обращения к жестким дискам.

С помощью жгутов электрических проводов (шлейфов) встроенные адаптеры внешних устройств или адаптеры, подключаемые через платы расширения, соединяются с разъёмами, расположенными, как правило, с тыльной стороны корпуса ПК (рис. 2.4) (например, клавиатура, мышь, джойстик, принтер, сканер, внешний модем). Другая часть контроллеров внешних устройств имеет разъёмы непосредственно на самой плате расширения (например, видеоадаптер, звуковая плата, встроенный модем).

▶ ОЗУ (англоязычное название *RAM, Random-Access Memory* — память с произвольной выборкой) хранит выполняемые программы и дан-

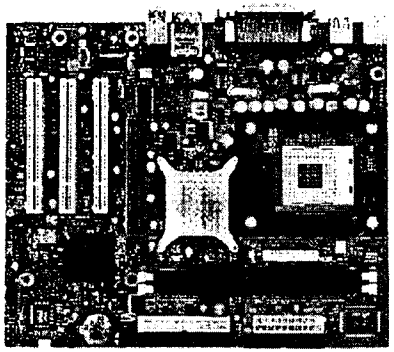


Рис. 2.3

Рис. 2.4

ные, непосредственно участвующие в операциях. Среднее время доступа к ее ячейкам составляет около 10 наносекунд (10^{-9} с). На современных ПК объем ОЗУ находится в пределах 128–4096 Мбайт.

От объёма ОЗУ зависит не только возможность работы с ресурсоёмкими программами, но и его производительность, поскольку при нехватке памяти в качестве её логического расширения используется жесткий диск, время доступа к которому значительно больше. На производительность ПК влияют также быстродействие ОЗУ и используемый способ обмена данными между микропроцессором и памятью.

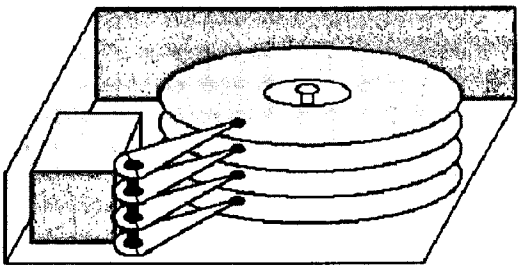
ОЗУ реализуется на микросхемах *DRAM* (*Dynamic RAM* — динамическая память с произвольной выборкой), характеризующихся по сравнению с другими разновидностями памяти низкой стоимостью и высокой удельной ёмкостью, но большим энергопотреблением и меньшим быстродействием. Каждый информационный бит (0 или 1) в *DRAM* хранится в виде заряда конденсатора. Из-за токов утечки этот заряд необходимо с определённой периодичностью обновлять. Регенерация содержимого памяти требует дополнительного времени, а запись информации во время регенерации в память не допускается.

Видеоадаптер ПК позволяет монитору взаимодействовать с процессором. Поэтому видеоадаптер должен иметь специальную память (*видеопамять*), в которую процессор записывает изображение в периоды относительно небольшой загруженности. А уже затем видеоадаптер, независимо от процессора, может выводить содержимое видеопамяти на экран. В современных условиях минимальным объёмом видеопамяти следует считать 32 Мбайт, приемлемым — 128 Мбайт, комфортным —

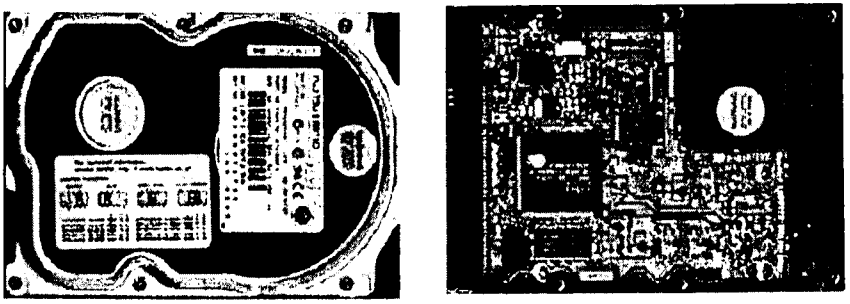
256 Мбайт. Ввиду больших объёмов видеоинформации видеоадаптеры, как правило, подключаются к материнской плате через специализированные шины с наибольшей пропускной способностью.

▶ *Жесткий магнитный диск* (*HDD — hard disk drive*) состоит из пакета жестких магнитных дисков, заключённого вместе с головками чтения-записи в герметичный корпус.

Название «винчестер» закрепилось за жестким диском из-за того, что первый загерметизированный жёсткий диск, разработанный фирмой *IBM* в 1973 г., имел 30 цилиндров (по 30 дорожек на каждой поверхности), а каждая дорожка — 30 секторов. Поэтому первый накопитель получил обозначение 30/30, как калибр винтовки «винчестер». *HDD* первых ПК в России (ЕС1841, ЕС1845) имели объём 5–20 Мбайт. В настоящее время индивидуальные пользователи в основном имеют *HDD* с объёмом в диапазоне от 40 Гбайт до 300 Гбайт, что обусловлено, с одной стороны, требованиями к аппаратной части наиболее распространённого программного обеспечения ПК, а с другой стороны, имею-



▲
Рис. 2.5



▲
Рис. 2.6

щимся торговым предложением и стоимостными характеристиками аппаратуры. На рис. 2.5 показано схематическое устройство *HDD*, а на рис. 2.6 приведён внешний вид *HDD Fujitsu MPG*3204*AT E*.

2.2.2. Внешние накопители информации

▶ *Дисководы* (*FDD — floppy disk drive*) осуществляют чтение-запись информации на гибкие магнитные диски, заключённые в пластмассовые конверты. Другое название таких дисков — дискеты с объёмом 1,4 Мбайт. В настоящее время дисководы практически вышли из употребления.

▶ *Устройства для чтения оптических дисков* (*CD-ROM — compact disk read only memory*) характеризуются большой информационной ёмкостью (до 680 Мбайт), низким по сравнению с *HDD* быстродействием, низкой стоимостью хранения данных в расчёте на 1 Мбайт, высокой надёжностью хранения данных и долговечностью носителей. Принцип работы *CD-ROM* основан на использовании луча лазера (твёрдотельный лазер с длиной волны 650 нм) для записи и чтения информации в цифровом виде. В процессе записи модулированный цифровым сигналом лазерный луч оставляет на активном слое оптического носителя (алюминиевая или золотая пленка на пластмассовом, чаще из поликарбоната, основании) стойкий след, который затем можно считывать, направив на него луч меньшей интенсивности и проанализировав изменение характеристик отражённого луча. В отличие от магнитных дисков, компакт-диски не имеют концентрических дорожек — вместо них на поверхности диска формируется одна спиральная дорожка. В большинстве моделей сам диск в накопителе вращается так, чтобы обеспечить вместо постоянной угловой постоянную линейную скорость. Современные приводы *CD-ROM* достигли более высоких скоростей считывания информации благодаря внедрению технологии *CAV* (*Constant Angular Velocity* — постоянная угловая скорость) — скорость считывания на периферийных участках диска — 4–7,8 Мбайт/с, а на внутренних — 2–3,5 Мбайт/с. Скоростные характеристики накопителей стандартизованы. Накопитель с одинарной скоростью вращения диска способен обеспечить скорость передачи 150 кбайт/с (этот показатель выбран за базовый). Поэтому, кроме фирмы-производителя, конкретный *CD-ROM* характеризуется кратностью превышения скорости передачи данных по отношению к накопителю с одинарной скоростью. Большинство современных *CD-ROM* имеют кратность 32–58.

Спецификация на формат *DVD* (*digital video disk* — цифровой видео диск) является стандартной и соблюдается всеми производителями. С помощью приводов *DVD-ROM* можно считывать данные с лазерных компакт-дисков форматов *CD-ROM, CD-R, CD-RW, DVD-ROM* (односторонних, двусторонних, однослойных, двухслойных, ёмкостью 4,7 – 17 Гбайт), дисков видео *DVD*. Для подключения используются интерфейсы *IDE/ATAPI* или *SCSI* (подробнее см. п. 2.3.5). Чтение осуществляется в режиме постоянной угловой скорости, кратность лежит в диапазоне 2 — 6 (до 7 Мбайт/с), при чтении дисков *CD-ROM* скорость выше — от 10 до 24 (36 Мбайт/с). Существуют также приводы с возможностью записи на носитель *DVD±R* (однократная запись), *DVD-RAM* и *DVD±RW* (оба с многократной записью).

Переносные накопители *USB Flash Drive*

Краткое название — *USB Flash Drive.* При емкости до нескольких гигабайт и необычайной легкости подключения к современным компьютерам, благодаря удобствам интерфейса *USB* (*Universal Serial Bus*) и опознаванию большинством нынешних операционных систем, эти устройства могут заменять собой как обычные дискеты, перезаписываемые компакт-диски, так и небольшие винчестеры. Строго говоря, *USB Flash Drive* не является винчестером, поскольку не содержит в себе главных «винчестерных» атрибутов — вращающихся пластин с магнитным покрытием. Однако, по функциональным характеристикам сходство с винчестерами очень близкое — длительное энергонезависимое хранение данных, быстрая (и в произвольном порядке) перезапись информации, количество циклов перезаписи тоже крайне велико, и достаточно высока скорость работы. По объему и скорости сегодняшние *USB Flash Drive* соответствуют винчестерам примерно шестилетней давности. Выгодные отличия от винчестеров — это малые габариты, очень малое энергопотребление в работе (несколько десятков миллиампер от порта *USB*), повышенная ударопрочность.

Внутри корпуса в два уровня размещаются две миниатюрные печатные платы (рис. 2.7), на одной из которых (основной) расположены сам контроллер памяти и *USB*, разъем порта *USB*, светодиод и вспомогательные элементы, а на другой (второй уровень) — несколько микросхем высоконадежной *Flash*-памяти (с обеих сторон платы).

Скорость записи — не менее 450 кбайт/с, скорость чтения — более 750 кбайт/с (ток потребления при записи/чтении — 36 и 33 мА соответственно). Ударостойкость 1000 G (выше любого современного винчес-

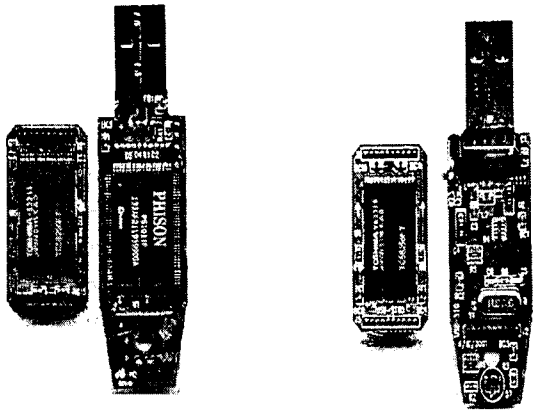


Рис. 2.7

тера). Время хранения данных — не менее 10 лет, и количество циклов записи (стирания) — не менее 1 миллиона (то есть 70 лет, если ежедневно перезаписывать весь диск по 40 раз). Накопитель весьма непритязателен и способен работать при высоких вибрациях, влажности и больших магнитных помехах.

2.2.3. Мониторы

Видеомониторы предназначены для вывода на экран текстовой и графической информации.

По конструкционному исполнению они делятся на мониторы, имеющие электронно-лучевую трубку (ЭЛТ), жидкокристаллические и плазменные мониторы. Подавляющая часть мониторов ПК основана на использовании электронно-лучевых трубок (ЭЛТ), но в настоящее время они постепенно вытесняются жидкокристаллическими мониторами, а плазменные мониторы пока встречаются относительно редко.

Конструкция монитора с ЭЛТ представлена на рис. 2.8.

Основным устройством такого монитора служит кинескоп с электронной пушкой, откуда под действием сильного электростатического поля исходит поток электронов. Через теневую маску кинескопа они попадают на внутреннюю поверхность стеклянного экрана монитора, покрытую люминофорными точками разного цвета. Поток электронов (луч) может отклоняться в вертикальной и горизонтальной плоскости, что обеспечивает последовательное попадание его на все поле экрана.

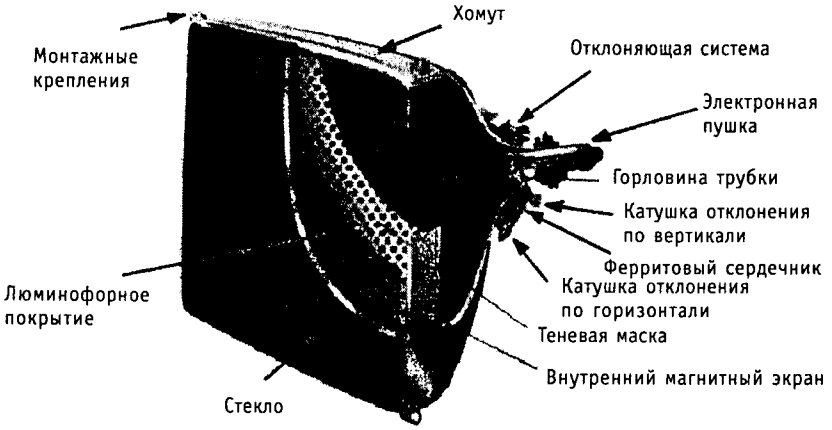


Рис. 2.8

Отклонение луча происходит посредством отклоняющей системы. Отклоняющая система состоит из нескольких катушек индуктивности, размещенных у горловины кинескопа. Изменение магнитного поля возникает под действием переменного тока, протекающего через катушки и изменяющегося по определенному закону (это, как правило, пилообразное изменение напряжения во времени), при этом катушки придают лучу нужное направление.

Частота перехода на новую линию называется частотой строчной (или горизонтальной) развертки. Частота перехода из нижнего правого угла в левый верхний называется частотой вертикальной (или кадровой) развертки. Амплитуда импульсов перенапряжения на катушках строчной развертки возрастает с частотой строк, поэтому этот узел оказывается одним из самых напряженных мест конструкции и одним из главных источников помех в широком диапазоне частот. Электроны попадают на люминофорный слой, после чего энергия электронов преобразуется в свет, то есть поток электронов заставляет точки люминофора светиться. Эти светящиеся точки и формируют изображение. Как правило, в цветном ЭЛТ мониторе используется три электронные пушки и три основных цвета точек люминофора: зеленый красный и синий. Основными характеристиками ЭЛТ мониторов являются: размер экрана по диагонали, разрешающая способность, размер точки (зерна) экрана, частота кадровой развёртки, режим строчной развёртки, глубина цвета.

Размер экрана по диагонали измеряется, как правило, в дюймах. Стандартными размерами являются 13, 14, 15, 17, 19, 21". *Разрешающая способность* — количество точек изображения по горизонтали и вертикали экрана. Стандартным разрешением можно считать 1024S768, то есть мониторы, работающие при таком режиме, способны выводить на экран 1024 точек по горизонтали и 768 точек по вертикали при 24- или 32-битной глубине цвета. *Глубина цвета*, например, 32-битная, означает возможность кодирования 2^{32} различных оттенков. Существуют и более высокие разрешающие способности, например, 1280S1024 точек и др. *Размер точки* (*зерна*) экрана (*шаг точек*) — выраженное в миллиметрах расстояние между центрами двух соседних точек люминофора. Для мониторов 13–14" ранних лет выпуска шаг точек составлял 0,31–0,39 мм. Для большинства современных мониторов шаг точек составляет — 0,21–0,25 мм. *Частота кадровой развёртки* для современных мониторов составляет 50-200 Гц, а основным режимом можно считать 100 Гц. *Режим строчной развёртки* характеризует способ формирования изображения. Если развёртка *чересстрочная*, то кадр изображения выводится на экран за два приёма (сначала прорисовываются все нечётные строки, затем — все чётные). По сути, этот режим снижает частоту кадровой развёртки в 2 раза. Современные мониторы имеют режим *построчной* (нечересстрочной — *non-interlaced*) развёртки.

Жидкокристаллические мониторы

Первый работоспособный жидкокристаллический дисплей, или *LCD*-монитор (*Liquid Crystal Display*), был создан в 1970 г. Жидкие кристаллы (*Liquid Crystal*) — это органические вещества, способные под напряжением изменять величину пропускаемого света. Жидкокристаллический монитор представляет собой две стеклянных или пластиковых пластины, между которыми находится суспензия. Кристаллы в этой суспензии расположены параллельно по отношению друг к другу, тем самым они позволяют свету проникать через панель. При подаче электрического тока расположение кристаллов изменяется, и они начинают препятствовать прохождению света. Жидкокристаллическая (ЖК) технология получила широкое распространение в компьютерах и в проекционном оборудовании. Первые жидкие кристаллы отличались нестабильностью и были мало пригодными к массовому производству. Реальное развитие ЖК технологии началось с изобретением английскими учеными стабильного жидкого кристалла — бифенила (*Biphenyl*). Жидкокристаллические дисплеи первого поколения можно наблюдать в калькуляторах, электронных играх и часах.

Существует два вида ЖК мониторов: *DSTN* (*dual-scan twisted nematic* — кристаллические экраны с двойным сканированием) и *TFT* (*thin film transistor* — на тонкопленочных транзисторах), также их называют соответственно *пассивными* и *активными* матрицами. Такие мониторы состоят из следующих слоев: поляризующего фильтра, стеклянного слоя, электрода, слоя управления, жидких кристаллов, ещё одного слоя управления, электрода, слоя стекла и поляризующего фильтра (рис. 2.9).

В первых компьютерах использовались восьмидюймовые (по диагонали) пассивные черно-белые матрицы. С переходом на технологию активных матриц размер экрана вырос. Практически все современные ЖК мониторы используют панели на тонкопленочных транзисторах, обеспечивающих яркое, четкое изображение значительно большего размера.

Поперечное сечение панели на тонкопленочных транзисторах представляет собой совокупность нескольких «слоев». Крайние слои выполнены из стекла. Между этими слоями расположен тонкопленочный транзистор, панель цветного фильтра, обеспечивающая нужный цвет — красный, синий или зеленый, и слой жидких кристаллов (рис. 2.10). Кро-

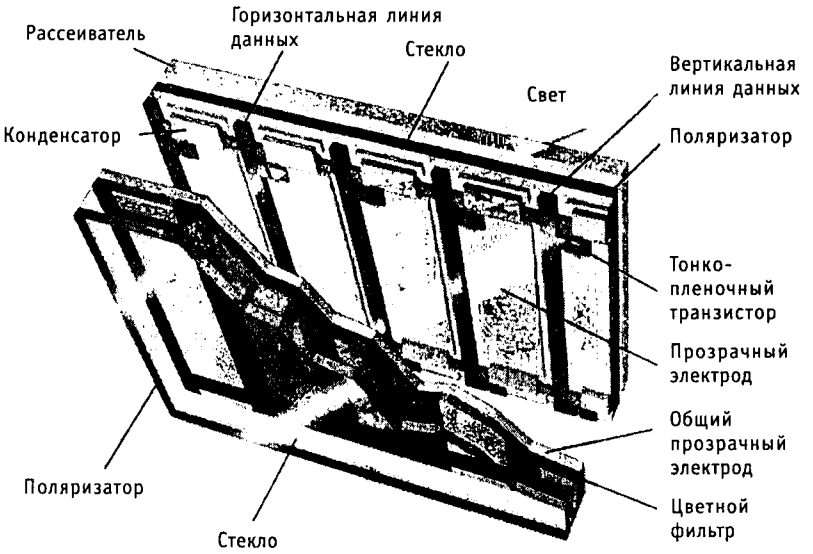


Рис. 2.9

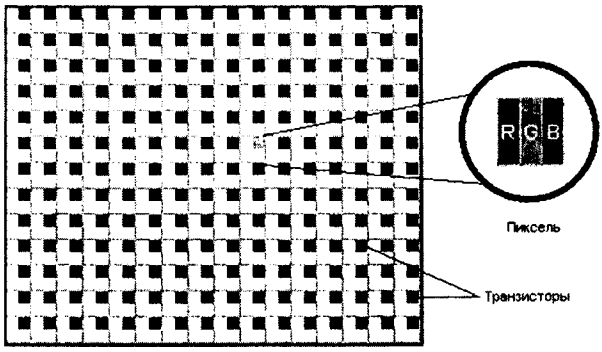


Рис. 2.10

ме того, существует флуоресцентная подсветка, освещающая экран изнутри.

При нормальных условиях, когда нет электрического заряда, жидкие кристаллы находятся в аморфном состоянии. В этом состоянии они пропускают свет. Количеством света, проходящего через жидкие кристаллы, можно управлять с помощью электрических зарядов — при этом изменяется ориентация кристаллов.

Как и в традиционных электроннолучевых трубках, пиксель формируется из трех участков — красного, зеленого и синего. А различные цвета получаются в результате изменения величины соответствующего электрического заряда (что приводит к повороту кристалла и изменению яркости проходящего светового потока).

TFT экран состоит из целой сетки таких пикселей, где работой каждого цветового участка каждого пикселя управляет отдельный транзистор. Для нормального обеспечения экранного разрешения 1024x768 (режим *SVGA*) монитор должен располагать именно таким количеством пикселей.

Большинство ЖК мониторов — цифровые. Это означает, что графической карте с цифровым выходом не придется производить цифроаналоговые преобразования, какие она производит в случае с ЭЛТ-монитором. Теоретически, это позволяет более точно передавать информацию о цвете и местоположении пикселя. В то же время, если подключать ЖК монитор к стандартному аналоговому *VGA* выходу, придется проводить аналого-цифровые преобразования.

ЖК мониторы имеют целый ряд преимуществ:

- в 2–3 раза меньшее потребление электроэнергии (15-30 ватт);
- отсутствие ионизирующего излучения;
- *LCD*-монитор совершенно плоский, он имеет четко определенное число пикселей по горизонтали и вертикали, каждый из них доступен схемам управления напрямую. Поэтому у этих устройств в принципе отсутствуют проблемы с фокусировкой и геометрическими отклонениями типа «бочка», «трапеция» и «параллелограмм»;
- в ЖК мониторах каждый пиксель включается или выключается отдельно, поэтому не возникает проблем со сведением лучей, в отличие от ЭЛТ-мониторов, где требуется безукоризненная работа электронных пушек;
- отсутствие мерцания (каждая точка светится столько времени, сколько нужно пользователю, и эффекта затухания нет, если картинка на экране не меняется из–за отсутствия построчной развёртки);
- меньшие габариты и масса;
- большая видимая область экрана.

Среди других отличий можно выделить следующие.

Разрешение: ЭЛТ-мониторы могут работать с несколькими разрешениями в полноэкранном режиме, тогда как ЖК монитор может работать только с одним разрешением. Меньшие разрешения возможны лишь при использовании части экрана. Так, например, на мониторе с разрешением 1024×768 при работе в разрешении 640×480 будет задействовано лишь 66% экрана.

Измерение диагонали: размер диагонали видимой области ЖК монитора соответствует размеру его реальной диагонали. В ЭЛТ-мониторах реальная диагональ теряет за рамкой монитора более дюйма.

Контрастность — сами по себе пиксели не вырабатывают свет, они лишь пропускают свет от подсветки. И темный экран вовсе не означает, что подсветка не работает — просто пиксели блокируют этот свет и не пропускают его сквозь экран. Под контрастностью *LCD* монитора подразумевается то, сколько уровней яркости могут создавать его пиксели. Обычно, контрастность 250:1 считается хорошей.

Угол обзора LCD монитора составляет от 140 до 180 градусов, что, практически, соответствует аналогичному показателю ЭЛТ мониторов.

2.2.4. Принтеры

Конструкцию современных принтеров принято разделять на *матричную*, *струйную* и *лазерную*. Матричные принтеры практически вышли из употребления из-за чрезвычайно низкой скорости получения отпечатков и значительного шума при распечатывании листов.

Струйные принтеры. Изображение формируется микрокаплями специальных чернил, выдуваемых на бумагу с помощью сопел. Эти принтеры обеспечивают более высокое качество печати по сравнению с матричными принтерами (разрешающая способность — до 600 точек на дюйм). Лучше приспособлены для цветной печати. На распечатку одного листа уходит от 15 до 100 секунд. Время печати цветной страницы — 3–5 минут.

Лазерные принтеры. Обеспечивают наилучшее качество и скорость печати. Используется принцип ксерографии: изображение переносится на бумагу со специального барабана, к которому электрически притягиваются частички красящего порошка. Печатающий барабан электризуется с помощью лазера по командам из ПК. Разрешающая способность — более 1200 точек на дюйм. Скорость печати от 2 до 16 листов в минуту. Все современные принтеры имеют, как правило, устройство автоматической подачи бумаги при печати на отдельных листах, собственное ПЗУ для хранения встроенных шрифтов (наиболее скоростной режим печати) и ОЗУ (0,5–16 Мбайт), используемое для построения выводимых на принтер рисунков и для хранения шрифтов символов, загружаемых в принтер программно. В качестве примера на рис. 2.11 приведён внешний вид принтера *Canon LBP*-810 (А4 8 стр./мин, 2400×600 точек на дюйм, *USB*).

Основными способами подключения принтеров к ПК являются подключение по параллельному (*LPT*) порту и по последовательной универсальной шине *USB*. Второй вид подключения является более перспективным и высокоскоростным.

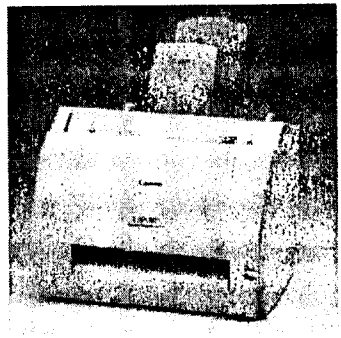


Рис. 2.11

2.2.5. Сканеры

Сканеры позволяют считывать графическую и текстовую информацию в ПК.

Сканеры делятся на настольные (планшетные), обрабатывающие лист бумаги целиком, и ручные (их нужно проводить над графическим объектом). Различные модели отличаются разрешающей способностью, количеством воспринимаемых цветов или оттенков серого цвета. Современные планшетные сканеры имеют разрешение от 2400 точек на дюйм при 24-х и 32-х битном цвете. С помощью специального программного обеспечения ПК (систем оптического распознавания текстов) возможно распознавание стандартных символов во введенной через сканер картинке.

На рис. 2.12 приведен внешний вид двух сканеров фирмы *HP* (*ScanerJet 2300 c, ScanerJet* 4500 c).

▲
Рис. 2.12

2.2.6. Сетевой адаптер (сетевая карта)

Сетевой адаптер — это плата расширения, устанавливаемая в рабочую станцию, сервер или другое устройство сети, позволяющая обмениваться данными между компьютерами. Операционная система через соответствующий драйвер управляет работой сетевого адаптера. Объем задействованных при этом ресурсов адаптера и центрального процессора системы может изменяться от реализации к реализации. На сетевых картах обычно имеется микросхема (либо гнездо для ее установки) перезаписываемой памяти для удаленной загрузки (*Remote Boot*), которая может быть использована для создания бездисковых станций. Как правило, подключаются к системной плате ПК через разъем *PCI* (см. п. 2.3.5). На задней панели расположен разъем для подключения сетевого кабеля. Современные сетевые адаптеры обеспечивают пропускную способность, обычно, 10 Мбит/с, 100 Мбит/с либо 1000 Мбит/с.

2.2.7. Модемы

Модем обеспечивает удаленный доступ ПК к компьютерным сетям по линиям связи (чаще всего — проводные и радиоканалы). Модем представляет собой устройство, имеющее, два интерфейса: с внешней точки зрения — цифровой интерфейс для подключения к ПК (обычно последовательный порт *RS*-232) и аналоговый интерфейс с каналом связи (телефонной линией) — разъем для телефонного кабеля (*RJ*-11). Основное устройство в составе модема — аналого-цифровой и цифро-аналоговый преобразователи, ответственные за преобразование сигнала из аналоговой формы (непрерывный сигнал-напряжение) в цифровую (отдельные отсчеты сигнала, дискретизованные по времени и квантованные по напряжению), и наоборот соответственно. Практически все современные модемы производят обработку информации в цифровой форме, без сколь-либо сложной аналоговой предобработки. По исполнению модемы можно разделить на внутренние и внешние.

Внутренний модем вставляется в разъем *PCI* системной платы ПК. Этот вид модемов делится на контроллерные и бесконтроллерные. Дальнейшим развитием бесконтроллерных модемов являются *Soft*-модемы (иначе *Win*-модемы).

Внешний модем имеет отдельный корпус и размещается рядом с компьютером, соединяясь кабелем с последовательным портом ПК.

2.2.8. Корпус ПК

Предназначен для размещения системной платы ПК с подключенными периферийными устройствами, отсеков для 3,5" и 5,25" устройств, блока питания, поддержания температурного режима размещенных в нем устройств, а также для защиты пользователя и окружающих электронных устройств от опасных напряжений, электромагнитных излучений и шумов, создаваемых системным блоком.

Корпуса существуют в двух основных исполнениях — в горизонтальном (*desktop*) и вертикальном (*mini-*, *middle-*, *big tower*). Чем больше размер корпуса, тем легче обеспечить охлаждение находящихся в нем устройств.

Конструктивно корпуса современных ПК делятся на два типа — *AT* (включение и выключение питания осуществляется пользователем вручную) и *ATX* (имеется возможность программного управления питанием). Отличия не сводятся только к необходимости ручного включения/выключения питания для корпусов *AT*. Разъемы для подачи питания на сис-

темных платах у разных типов корпусов имеют различную спецификацию. Как следствие, ответные части разъемов питания на корпусах ПК также различны. Поэтому тип устанавливаемой системной платы должен соответствовать типу корпуса.

2.3. Аппаратное обеспечение современного ПК

2.3.1. Микропроцессоры

Архитектура линии МП x86 (это сокращенное название моделей, базирующихся на системе команд первого микропроцессора 8086) компании *Intel* основана на концепции *CISC* (*Complex Instruction Set Calculation*) — расширенной системе команд переменной длины, появившейся в 1978 г. Команды x86 могут иметь длину от 8 до 108 бит, и МП должен последовательно декодировать инструкцию после определения ее границ. Когда МП были скалярными устройствами (то есть могли в каждый момент времени выполнять только одну команду), конвейерная обработка практически не применялась (исключение составляли большие ЭВМ).

В 1986 г. появились МП, основанные на архитектуре *RISC* (*Reduced Instruction Set Calculation*) — сокращенном наборе команд фиксированной длины, которая была оптимизирована для суперскалярных (с возможностью выполнения нескольких команд одновременно) конвейерных вычислений. С тех пор обе линии до недавнего времени развивались практически независимо. *Intel* с целью обеспечения совместимости не могла отказаться от архитектуры *CISC* даже в новейших моделях МП x86, а фирма *Apple*, ориентировавшаяся на МП с архитектурой *RISC*, не могла существенно увеличить свою долю на рынке ПК из-за трудностей с использованием программ для x86 на своих компьютерах.

Однако в отдельных модификациях МП *AMD* удалось совместить обе архитектуры. То есть микроядро процессора работает на основе инструкций *RISC*, а специальный блок интерпретирует команды *CISC* для обеспечения совместимости с системой команд x86. Крупнейшие фирмы постоянно увеличивают число функциональных узлов на кристалле МП, что позволяет обрабатывать параллельно больше команд. Это существенно усложняет блоки управления для распределения потоков команд по узлам обработки. На данный момент большинство МП не может выполнять более четырех команд одновременно, при этом управляющая логика занимает на кристалле слишком много места.

Последовательная структура кода программ и большая частота ветвлений делают задачу распределения потока команд крайне сложной. Современные МП содержат большое количество управляющих элементов для минимизации потерь производительности, связанных с ветвлениями (логику предсказания переходов). Они изменяют порядок команд во время исполнения программы, пытаются предсказать, куда необходимо будет перейти в результате очередного ветвления, и выполняют команды до вычисления условий ветвления. Если путь ветвления предсказан неверно, МП должен сбросить полученные промежуточные результаты, очистить конвейеры и загрузить нужные команды, что требует большого числа тактов. Таким образом, МП, теоретически выполняющий 4 команды за такт, реально выполняет менее двух команд за это время. Проблему осложняет и тот факт, что даже современные микросхемы *DRAM* не успевают за тактовой частотой МП.

Важным элементом МП является блок обработки данных с плавающей точкой (*FPU — Floating Point Unit*), встроенный во все модели процессоров, начиная с модели *Intel* 80486. От его эффективности напрямую зависит скорость работы со сложными приложениями (графика, мультимедиа, трехмерные объекты). До недавнего времени наиболее эффективными были *FPU* фирмы *Intel*. С появлением процессора *Athlon* фирмы *AMD* у них появился серьезный конкурент, превосходящий микропроцессоры *Intel* по некоторым показателям.

Тактовая частота и объем установленной на МП памяти команд, данных (*кэш* 1-го уровня) и памяти буфера ввода-вывода (*кэш* 2-го уровня) являются важными факторами, влияющими на его производительность. Имеется ряд специализированных задач, ускоренное решение которых возможно за счет оптимизации операций на аппаратном уровне. Впервые эту проблему пыталась решить *Intel* путем внедрения технологии *MMX* (*MultiMedia Extension* — мультимедийные расширения). И так немалый набор команд x86 был расширен за счет 57 дополнительных инструкций типа *SIMD* (*Single Instruction — Multiple Data* — одна инструкция для многих данных), позволивших распараллелить обработку однородных данных. Технология *MMX* значительно ускорила работу МП с мультимедийными приложениями. Но у нее имелся существенный недостаток — невозможность обработки данных с плавающей точкой.

Впервые технология для обработки данных с плавающей точкой была реализована фирмой *AMD* на МП К6-2 и получила название *3Dnow*. Она включает 21 инструкцию типа *SIMD*, оптимизированных для параллельной обработки данных с плавающей точкой.

С некоторым опозданием похожую технологию под названием *SSE* (*Streaming SIMD Extension*) реализовала *Intel* в МП *Pentium* III. Хотя производительность обеих технологий примерно одинакова (около 2 миллиардов операций/с для 32-разрядных чисел с одинарной точностью при частоте ядра МП 500 МГц), *SSE* опережает *3DNow* за счет использования отдельных регистров для этого типа данных. Фактически *Intel* ввела новый режим работы МП — параллельную обработку инструкций *FPU* и *SSE*.

При производстве МП используются технологические нормы, устанавливающие допустимое расстояние между электрическими цепями на кристалле кремния и минимально возможный размер элементов. К тому же уменьшение размеров приводит и к уменьшению рассеиваемой мощности, что позволяет поднять рабочую частоту, на которой надежно функционируют элементы. Еще недавно стандартом считался показатель 0,35 микрон, сейчас МП изготавливают большей частью по норме 0,13 или 0,09 микрон.

Фирма *Intel* ввела технологическое новшество, разделив внутреннее устройство МП на так называемое *ядро* (то есть сам по себе процессор) и остальные элементы (контроллеры памяти, интерфейс шины, кэш и пр.) путем подачи на них разных напряжений питания, ввела разные обозначения для именования ядра и процессора в сборе. Например, выпускались модели МП *Celeron* с ядром *Covington* (полный аналог *Deschutes*), ядром *Mendocino* и ядром *Coppermine*. МП *Pentium* II выпускались с ядром *Klamath* и *Deschutes*. МП с ядром *Katmai* после окончания разработки получил наименование *Pentium* III. У МП *Pentium* IV ядра получили названия *Willamette*, *NorthWood*, а затем *Prescott*. Фирма *Intel* ввела в ПК-индустрию и понятие *конструктив*. Это не совсем по-русски звучащее слово вместе с тем весьма точно передает суть некоего сооружения, внутри которого находятся и процессорная плата, на которой располагаются кристаллы собственно процессора, и кэш-память второго уровня, и корпус, охватывающий эту плату с разъемами для подключения к системной плате. Все это сооружение (без радиатора) было названо *SECC* (*Single Edge Contact Cartridge* — картридж с односторонними контактами). МП *Celeron* размещались на конструктиве, представляющем собой печатную плату. Затем последовал выпуск *Celeron* в конструктиве *PPGA* (*Plastic Pin Grid Array*), то есть возврат к технологии расположения контактов в виде прямоугольной матрицы. В общем, их называют *сокетами* (*Socket*) с указанием количества контактов для подключения МП. Сравнительно недавно появились конструктивы: *mPGA Socket* 478 и *LGA Socket* 775 для МП *Intel* и *Socket A*, и

OmPGA Socket 754, *Socket* 939 для МП *AMD*. То есть произошел возврат к техническим решениям, характерным для *Socket* 7, но на ином технологическом уровне. Спецификация разъема описывает не только его конструктив, но и электрические параметры, и назначение контактов, предопределяет порядок взаимодействия с шинами данных, особенности работы с основной и кэш-памятью и др. показатели.

Один из самых распространенных ныне конструктивов *Socket* 478 предназначается для подключения МП *Pentium* IV. Для его поддержки требуются микросхемы системного набора *Intel* 845 и выше. Корпус МП — хорошо зарекомендовавший себя корпус *mPGA*2 (рис. 2.13).

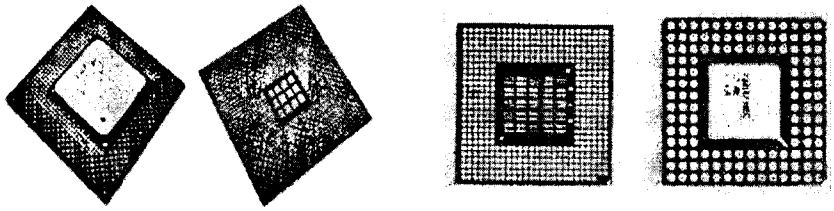


Рис. 2.13

До появления МП *Pentium MMX* все элементы кристалла работали при одном напряжении питания. Затем с ростом числа транзисторов на кристалле, появлением новых блоков, увеличением встроенной кэш-памяти пришлось специально уменьшать напряжение ядра МП. Так появились понятия внешнего и внутреннего (для ядра) напряжений питания МП. В настоящее время практически все МП для ПК выпускаются с двойным напряжением питания. Как правило, более высокоскоростные модели имеют большее напряжение питания ядра. Например, МП *Pentium* IV с технологией *HT* (*Hyper-Threading*) имеет напряжение питания для ядра 1,3 В. Эта технология позволяет производить обработку программного кода двумя АЛУ, расположенными на одном кристалле микропроцессора.

МП Pentium IV фирмы Intel

В ноябре 2000 г. *Intel* приступила к производству 32-разрядного МП, ранее известного как *Willamette*, работающего на частоте 1,5 ГГц.

Технические новшества этого МП были следующие:

- асимметричное ядро с блоками, работающими на различных скоростях;

- значительно улучшенная версия суперскалярного механизма исполнения инструкций;
- новый кэш второго уровня, отслеживающий порядок выполнения инструкций;
- переработанные блоки операций с мультимедийными данными и числами с плавающей точкой;
- огромный набор новых инструкций;
- новая 100 МГц шина, передающая по 4 пакета данных за такт (что эмулирует результирующую частоту 400 МГц);
- конвейер выполнения инструкций из 20 стадий.

Pentium IV при длине конвейера 20 стадий имеет самое меньшее время выполнения такта, позволяющее достичь максимальной тактовой частоты, но и получает самые большие задержки для связанных друг с другом операций (второй операции придется ожидать 20 тактов, пока не завершится первая операция). Эта проблема частично решается за счет буфера, некоторые инструкции из которого можно выполнять независимо от результата предыдущих. Для этого требуется точно предсказывать так называемые переходы. Объем кэш-памяти второго уровня у *Pentium* IV увеличен до 256 кбайт и может достигать 1 Мбайта, а новый механизм предсказания переходов позволил повысить точность «попадания» до 95 %.

Еще один инструмент ускорения работы — *Advanced Dynamic Execution*, улучшенная версия механизма суперскалярного внеочередного выполнения инструкций, когда МП нарушает их естественную последовательность с целью более плотной загрузки исполнительных модулей. Не менее важной для эффективности процессора является производительность модулей, непосредственно выполняющих те или иные операции с целыми числами, числами с плавающей точкой, специфическими данными, когда одна инструкция оперирует сразу несколькими пакетами данных.

В *Pentium* IV блок целочисленных операций работает на удвоенной скорости относительно скорости процессора — то есть при базовой частоте 1,5 ГГц скорость работы целочисленных модулей составляет 3 ГГц за счет выполнения операций за полтакта. В идеальном случае два имеющихся модуля могут выполнять по 4 операции с целыми числами за один такт работы процессора. Модулей обработки чисел с плавающей точкой всего два (против трех у *Athlon*), что обеспечивает пиковую производительность в операциях с плавающей запятой 1,5 *GFLOPS* (*Giga FLoating Operating Per Second*). Для *Athlon* на той же частоте производительность достигает 3 *GFLOPS*.

Важным преимуществом *Pentium* IV является блок обработки *SIMD* (*Single Instruction — Multiple Data*) инструкций. 64-битные инструкции рассчитаны на обработку чисел с плавающей точкой, а 128-битные — на целочисленные данные. Таких модулей у *Pentium* IV также два: один для регистровых операций и другой — для арифметических. Теоретически возможен режим работы, когда за один такт полняется одна *SIMD*-инструкция, состоящая из четырех операций. Таким образом, пиковая производительность *Willamette* в случае использования *SIMD* составляет 6 *GFLOPS*. Набор *SIMD*-инструкций (*SSE2*) включает 76 совершенно новых, оперирующих с широким диапазоном данных, и 68 расширенных для работы с целыми числами.

С выходом *Pentium* IV появляется новая системная шина на 100 МГц, которая должна передавать по 4 пакета данных за такт, эмулируя результирующую частоту 400 МГц, с пиковой про-пускной способностью 3,2 Гбайт (400 МГц, 64 бит). По-видимому, она станет новым стандартом, для микропроцессоров с 64-разрядной архитектурой.

МП Itanium фирмы Intel

Хотя новая 64-разрядная архитектура основана на многолетних исследованиях *Intel* и *Hewlett Packard*, она радикально отличается от всего того, что до сих пор было представлено на рынке. По-видимому, с выходом *Merced* (название проекта по разработке процессора, уже получившего «рыночное» имя *Itanium*) изменится вся компьютерная индустрия.

Архитектура, известная под названием *Intel Architecture*-64 (*IA*-64), не является ни 64-разрядным расширением 32-разрядной архитектуры ×86, ни переработкой 64-разрядной архитектуры *RA-RISC* компании *HP*. В *IA*-64 используются длинные слова команд (*Long Instruction Words — LIW*), предикаты команд (*Instruction Predication*), устранение ветвлений (*Branch Elimination*), упре-ждающая загрузка данных (*Speculative Loading*) и другие новшества для того, чтобы «извлечь больше параллелизма», то есть одновременного выполнения команд. Это стало возможным из-за наличия нескольких устройств арифметико-логической обработки данных, расположенных на одном кристалле микропроцессора.

Команды в формате *IA*-64 упакованы по три в 128-битный пакет для быстрейшей обработки. *Intel* предпочитает называть свою новую *LIW*-технологию *Explicitly Parallel Instruction Computing* или *EPIC* (полностью параллельного выполнения команд). В любом случае формат команд

IA-64 не имеет ничего общего с x86. Каждый 128-битный пакет содержит шаблон (*template*) длиной в несколько бит, помещаемый в него компилятором, который указывает МП, какие из команд могут выполняться параллельно. Наличие параллелизма определяет именно компилятор.

Компиляторы для *IA*-64 будут использовать технологию «помеченных команд» (*predication*) для устранения потерь производительности из-за неправильно предсказанных переходов и необходимости пропуска участков кода после ветвлений. Когда процессор встречает «помеченное» ветвление в ходе выполнения программы, он начинает одновременно выполнять все ветви. После того как будет определена «истинная» ветвь, процессор сохраняет необходимые результаты и сбрасывает остальные. Компиляторы для *IA*-64 будут также просматривать исходный код с целью поиска команд, использующих данные из памяти. Найдя такую команду, они будут добавлять команды упреждающей загрузки (*speculative loading*) и проверки актуальности данных упреждающей загрузки (*speculative check*). Во время выполнения программы первая из команд загружает данные в память, до того как они понадобятся.

2.3.2. Системные платы

▶ *Системная (материнская) плата* (СП) является важнейшим элементом ПК, служит для объединения и организации взаимо-действия других компонентов.

По сути, выбор конфигурации компьютера начинается именно с выбора системной платы. Ее основными параметрами являются: форм-фактор, поддерживаемые процессорный интерфейс, типы и объем системной (оперативной и кэш) памяти, поддерживаемый интерфейсы видеоадаптера, накопителей, интерфейсы общего назначения (ввода-вывода, клавиатуры, *USB*, *IEEE*1394 и др.), тип *BIOS*. В качестве дополнительных часто выступают встроенные графические, звуковые, коммуникационные средства.

Основные параметры системной платы большей частью определяются системным набором (*чипсетом*). Именно он является «ядром» всей компьютерной системы. Форм-фактор, или типоразмер СП, определяет ее размеры, тип разъема питания, расположение элементов крепления (отверстий, клипсов), размещение разъемов различных интерфейсов и т. д.

Сегодня существуют два основных типоразмера (форм-фактора) СП: *AT* (различных модификаций) и *ATX* (также имеющий подклассы). Пол-

номасштабные СП с форм-фактором *AT* сейчас найти практически невозможно. Иногда производители выпускают по две модификации каждой платы — в форматах *AT* и *ATX*. Все *AT*-платы имеют общие черты. Для крепления в корпусе в них сделано три ряда отверстий. В них вставляются пластмассовые клипсы и винты, которыми прикрепляют плату к соответствующим прорезям и резьбовым соединениям. Почти все имеют встроенные контроллеры последовательных и параллельных портов, *IDE*, ввода-вывода. Типовое расположение разъема клавиатуры, гнезда под процессор, разъемов под модули оперативной памяти, а также других элементов показано на рис. 2.14.

Форм-фактор *ATX* был предложен *Intel* еще в 1995 г. К настоящему моменту большинство СП выпускаются в формате *ATX*. К его новым (по сравнению с *AT*) возможностям относятся: размещение портов ввода-вывода на системной плате; встроенный разъем мыши типа *PS/2*; расположение *IDE*-разъемов и разъемов контроллера дисководов ближе к самим устройствам (благодаря развороту платы, по сравнению с *AT*, на 90 градусов); перемещение гнезда процессора с передней части платы на заднюю, рядом с блоком питания; использование единственного 20-контактного разъема питания, вместо двух раздельных у *AT*. Предусмотрена возможность управления режимами работы блока питания со стороны контроллера СП. Вентилятор блока питания является нагнетающим, поэтому на СП попадает меньше пыли, а воздух, поступающий из блока питания, сначала охлаждает МП.

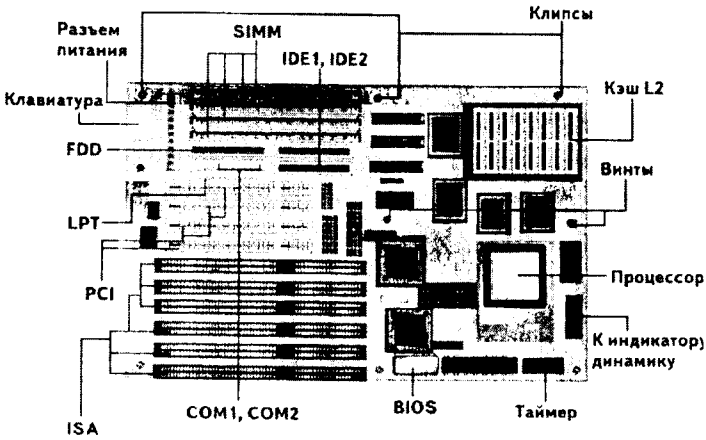


Рис. 2.14

2.3.3. Системный набор

▶ Потенциальные возможности и эффективность компьютерной системы во многом определяются установленным на СП набором *системных микросхем* (*чипсетом*), который обеспечивает работу процессора, системной шины, интерфейсов взаимодействия с оперативной памятью и другими компонентами ПК.

До сих пор лидером в разработке и изготовлении системных наборов является корпорация *Intel*. Прочие производители, а их не так много, в отличие от других секторов рынка компьютерных микросхем, вынуждены ориентироваться на решения *Intel* и обеспечивать совместимость с ее компонентами и стандартными интерфейсами.

Многие современные системные наборы включают две базовые микросхемы, которые в англоязычной компьютерной литературе принято называть соответственно микросхемами *North Bridge* (*северный мост*) и *South Bridge* (*южный мост*). Первая из них обычно обеспечивает управление шиной *AGP*, шиной системной памяти, шиной *PCI* и взаимодействие с системной шиной процессора. Южный мост управляет интерфейсами *IDE*, *USB*, *ACPI*, *IEEE*1294, включает мост *ISA-PCI*, контроллер клавиатуры, мыши, *FDD*. Оба моста соединены шиной *PCI*.

2.3.4. Системная память

Обычно под системной понимают лишь оперативную память. На самом деле работоспособность всей компьютерной системы зависит от характеристик подсистемы памяти в целом. Подсистема памяти охватывает:

- оперативную память как таковую;
- кэш-память первого уровня, расположенную в ядре МП;
- кэш-память второго уровня (в некоторых конфигурациях она выступает как кэш третьего уровня), размещаемую на СП, на картридже МП или в его ядре;
- контроллер памяти;
- шины данных и команд, объединяющие все элементы подсистемы в единое целое.

Динамическая и статическая память

Системная память подразделяется на два типа — с динамической и статической выборкой. В первом случае значение бита информации в ячейке определяется наличием или отсутствием заряда на миниатюрном

конденсаторе, управляемом одним — двумя транзисторами. В статической памяти применены специальные элементы — триггеры, реализованные на 4–6 транзисторах. Естественно, что из-за необходимости ожидания накопления (стекания) заряда на конденсаторе быстродействие *DRAM* ниже. Однако благодаря большему числу транзисторов на ячейку, память *SRAM* существенно дороже. Обычно модули *DRAM* применяют в оперативной и видеопамяти, а модули *SRAM* — в качестве быстрых буферных элементов в процессорах, на СП, в контроллерах дисков, *CD-ROM* и пр.

Статическая память

Ячейкой в статической памяти является триггер — логический элемент с двумя устойчивыми состояниями, в любом из которых он сохраняется до тех пор, пока подается питание. Время срабатывания триггера составляет в современных микросхемах единицы наносекунд. Однако плотность компоновки ячеек *SRAM* существенно ниже, чем в микросхемах *DRAM*, а стоимость производства выше, поэтому статическая память применяется лишь в наиболее ответственных компонентах.

В современных системах обычно используется конвейерный режим с пакетным способом передачи данных (*Pipelined Burst Cache*), организованный на микросхемах статической памяти с синхронным доступом.

Асинхронная динамическая память (DRAM)

Асинхронный интерфейс работы динамической памяти предусматривает наличие отдельного устройства в контроллере памяти для генерации управляющих сигналов. Для операций чтения/записи определяется продолжительность, зависящая от технологии изготовления микросхемы, ширины шины данных, наличия буфера и других параметров.

Каждый цикл операции чтения и записи ячеек памяти может иметь продолжительность, отличную от других циклов. Никакая последующая операция не может начаться до сигнала об окончании предыдущей. Для генерации необходимых импульсов контроллер асинхронной памяти имеет делитель, вырабатывающий сигналы необходимой частоты для каждой операции внутри цикла.

Синхронная динамическая память (SDRAM)

В этом случае все команды и обмен данными по шине памяти проходят синхронно с тактовыми импульсами системной шины, поэтому все циклы одной операции имеют одинаковую продолжительность.

Ячейки в динамической памяти образуют матрицу, состоящую из строк и столбцов. При считывании данных содержимое одной строки целиком переносится в буфер, реализованный на элементах статической памяти. После этого из строки считывается значение (0 или 1) нужной ячейки, и содержимое буфера вновь записывается в прежнюю строку динамической памяти. Такие переносы данных осуществляются путем изменения состояния конденсаторов ячеек, то есть происходит процесс заряда (разряда, если конденсатор был заряжен). Так как конденсаторы чрезвычайно малы, высока вероятность непроизвольного изменения их состояния из-за паразитных утечек и наводок.

Для исключения утраты данных проводятся циклы регенерации с определенной частотой, которые обычно инициализируются специализированными микросхемами. За один такт микропроцессора память может регенерироваться несколько раз.

Без участия МП информация из памяти может считываться блоками устройством прямого доступа к памяти *DMA — Direct Memory Access*. При необходимости оно посылает запрос, содержащий адрес и размер блока данных, а также управляющие сигналы. Так как доступ к памяти по каналам *DMA* одновременно могут иметь несколько устройств (например, процессор, видеокарта с интерфейсом *AGP*, контроллер шины *PCI*, *HDD*), образуется очередь запросов, хотя каждому потребителю ресурсов памяти требуются собственные данные, часто расположенные не только в разных микросхемах, но и в разных банках памяти.

2.3.5. Интерфейсы

▶ *Интерфейс* в компьютерных системах обозначает как физическое аппаратное устройство, так и совокупность алгоритмических правил обмена информацией с возможностью их перепрограммирования. Часто под интерфейсами понимают разъемы подключения дополнительных адаптеров к системным платам и порты ввода-вывода для подключения периферийных устройств.

Peripheral Component Interconnect (*PCI*) — соединение периферийных компонентов. Поддерживает тактовую частоту до 33 МГц (вариант *PCI* 2/1 — до 66 МГц, *PCI-X* — до 100 МГц), имеет пропускную способность до 132 Мбайт/с (264 Мбайт/с для 32-разрядных и 528 Мбайт/с для 64-разрядных данных на частоте 66 МГц). Конструктивно разъем состоит из двух следующих подряд секций по 64 контакта. Разъемы и карты к ним поддерживают уровни сигналов либо 5 В, либо 3,3 В, либо оба уровня. Интерфейс обеспечивает поддержку режима автоматической конфигурации компонентов при установке (*Plug-and-*

Play). Все разъемы *PCI* на плате сгруппированы в сегменты, число разъемов в сегменте ограничено четырьмя. Если сегментов несколько, они соединяются посредством так называемых мостов (*bridge*). В настоящее время является самым распространенным и универсальным интерфейсом.

Universal Serial Bus (*USB*) — универсальная последовательная шина. Предусматривает подключение до 127 внешних устройств к одному *USB*-каналу (по принципу общей шины). Современные СП обычно имеют 2 — 4 канала на контроллер. Обмен данными по шине происходит в пакетном режиме при пропускной способности до 12 Мбит/с (до 480 Мбит/с в версии *USB* 2,0).

Accelerated Graphics Port (*AGP*) — ускоренный графический порт. Предназначен исключительно для подключения видеоадаптеров к отдельной (не связанной с системной шиной) магистрали *AGP*, имеющей выход непосредственно на системную память.

Integrated Drive Electronics (*IDE*) — встроенная электроника накопителя. Предназначен для обеспечения работы *HDD* и других накопителей. Для работы компонентов с интерфейсом *IDE* требуется наличие соответствующего контроллера. В большинстве случаев он выполняется встроенным на СП и поддерживает 2 разъема *IDE* (*Primary* — первичный и *Secondary* — вторичный), к каждому из которых можно подключать по 2 устройства (*Master* и *Slave* — ведущий и ведомый). В настоящее время используется спецификация интерфейса *ATA*-2, который поддерживает режимы *LBA* (*Logical Block Address* — логическая адресация блоков), что обеспечивает работу с дисками большой (более 528 Мбайт) емкости; *PIO* (*Programmed Input-Output* — программный ввод-вывод), *DMA* (*Direct Memory Access* — прямой доступ к памяти). Пропускная способность — до 100 Мбайт/с.

Клавиатура к системным блокам *AT* подключается через разъем *DIN5* к специальному контроллеру *UPI* (*Universal Peripheral Interface*) на СП. В самой клавиатуре имеется микроконтроллер, соединенный последовательным каналом с *UPI*. Последовательный интерфейс клавиатуры не совместим с последовательным интерфейсом *RS*-232. Интерфейс *PS*/2 отличается от *AT* только разъемом и контроллером на СП. В качестве разъема применяют *DIN*6. Клавиатуру можно также подключить и по интерфейсу *USB*.

Мышь подключается к СП по последовательному интерфейсу *RS*-232C, а чаще через разъем *PS*/2. Возможен вариант подключения мыши по интерфейсу *USB*.

2.4. Программное обеспечение ПК

Точное число программ для современного компьютера подсчитать невозможно, однако по традиции выделяется два вида: системное программное обеспечение (ПО) и прикладное ПО.

▶ К *системному ПО* относится операционная система, а также программы-утилиты, обеспечивающие дополнительные сервисные возможности, не входящие в операционную систему.

Операционной системе, как главному элементу программного обеспечения, будет уделено значительное внимание в следующей главе. В каждой группе программных продуктов можно выделить один из наиболее распространенных, хотя мнение авторов в этом случае может не совпадать с мнением других. К программам-утилитам в большинстве случаев относятся: утилиты для определения аппаратного состава ПК и диагностики его узлов (пакет *SiSoftware Sandra Professional*). Очень распространены графические методы тестирования производительности ПК с помощью программ *Futuremark 3Dmark05 Pro*. Прекрасным дополнением к операционной системе Windows любой версии является пакет утилит *Norton Systemworks Pro*, включающий программы для диагностики и резервирования данных, антивирусный пакет, утилиту для восстановления операционной системы после сбоя, восстановление случайно удаленных файлов и др. Выпускаются и отдельные программы для создания, удаления, изменения и конвертации разделов жесткого диска (*PowerQuest PartitionMagic*, *Partition Explorer*). При установке нескольких операционных систем на один жесткий диск потребуется программа-утилита загрузки, например, *BootManager*. В связи с тем, что операционные системы, как правило, не обладают средствами для повышения степени защиты файлов пользователя, существуют программы для полного удаления файлов с разных носителей (*Disk Wiper Pro*), а также *Encrypted Disk PE* — для шифрования файлов и каталогов.

Большую группу программ-утилит составляют антивирусные программы. Наиболее известны российские антивирусные программы *Kaspersky Antivirus Pro*, *Dr. Web for Windows*, среди зарубежных программ-антивирусов выделяются *Norton Antivirus*, *McAfee Internet Security Suite*, *Panda Platinum Internet Security*. Эти программы включают в себя антивирусный сканер, монитор, антиспам (для удаления нежелательной электронной почтовой рассылки) и средства для уничтожения рассылки нежелательной электронной рекламы.

Работа любого пользователя не обходится без средств архивации (сжатия, компрессии) данных. Архиватор позволяет сэкономить место на жестком диске за счет применения методов алгебраического кодиро-

вания информации, которые часто связаны с удалением повторяющихся данных из исходного файла. Современные архиваторы *WinRAR*, *WinZIP* позволяют достичь высокого коэффициента компрессии (в зависимости от типа файла) — отношения длины заархивированного файла к длине исходного. В некоторых случаях, например, при обработке аудио- и видеоданных без методов архивации вообще обойтись невозможно.

Отдельную группу программ-утилит составляют программные средства для записи *CD* и *DVD* дисков. Во многих случаях пользователи предпочитают программный комплекс для записи дисков *Ahead Nero Burning ROM*. Существует также средство для создания точных копий дисков, включая и защищенные *CloneCD*, *CloneDVD*.

При обработке различных мультимедиаданных потребуются графические, аудио-, видеоконвертеры. Значительное число графических форматов файлов можно обработать с помощью пакета программ *ACDSee PowerPack*, а известными кодеками для сжатия видеоданных являются *DivX Pro* и *InterVideo WinDVD*. Не менее популярны и утилиты — мультимедиапроигрыватели, среди которых несомненным лидером является *Winamp Pro*.

▶ *Прикладное ПО* предназначено для решения задач пользователя в различных предметных областях.

Рассмотрим основные виды прикладного программного обеспечения и условно разделим его на несколько групп. Это: пакеты офисных прикладных программ, пакеты профессиональных графических и издательских программ, программы для мультимедиа редактирования, пакет трехмерной машинной графики, пакеты символьной математики и научных вычислений, системы автоматизированного проектирования, системы программирования, прикладные программные продукты для различных отраслей науки, образования, производства. Продолжение классификации программного обеспечения является очень объемной задачей, поэтому ограничимся теми программными продуктами, с которыми сталкивается большинство пользователей в своей повседневной работе.

Невозможно представить работу современного пользователя ПК без пакета программ *Microsoft Office*, изучению которого в данном учебнике посвящена четвертая глава.

Пользователь, желающий грамотно и профессионально оформить свою работу графическими иллюстрациями, вряд ли обойдется без пакета графических программ *Adobe Creative Suite*, включающего «короля» графических редакторов *Adobe Photoshop*. Другими полезными программами из этого пакета являются *Adobe Illustrator* — для работы с векторной графикой, *Adobe GoLive* — макетирование и разработка Web-сайта. Очень популярен формат электронного документа *Portable Document Format* (*pdf*), с которым позволяет работать программа *Adobe Acrobat*.

Как альтернатива, существует пакет программ *CorelDRAW Graphics Suite*, включающий программы аналогичного назначения.

Под мультимедиа редактированием обычно понимают создание небольших видеороликов, включающих как видео-, так и звуковую дорожку. Записать и оцифровать видео в любительском и полупрофессиональном виде (конечно, при наличии адаптеров видеозахвата) можно с помощью программ *Pinnacle Studio*, *Adobe Premiere Pro* или *Ulead Media Studio*. Для редактирования только звуковой дорожки подойдет редактор *Sony Sound Forge*.

Современные домашние ПК настолько производительны, что легко позволяют работать с трехмерной графикой. Визуальное конструирование трехмерных моделей и их анимацию позволяет осуществлять *3DMax Studio*. Другие пользователи среди программ трехмерного моделирования предпочитают программный пакет *Maya*.

Для нынешних студентов теоретические знания в области математики, безусловно, необходимы, но пакеты символьных вычислений могут значительно облегчить постижение ее глубин. Среди основных математических пакетов следует указать *Maplesoft Maple*, так как на его символьном ядре разрабатываются другие, например, *Mathsoft MathCAD* и *Wolfram Research Mathematica*. Множество программ позволят автоматизировать статистические исследования и анализ данных, например, *StatSoft Statistica* или *SPSS*. Выполнять имитационное моделирование можно в программе *GPSS World*.

Системы автоматизированного проектирования, с внешней стороны «похожие» на графические пакеты программ, обладают совсем другим внутренним содержанием. Например, распространенная САПР *Autodesk AutoCAD* представляет собой целую систему программирования и управления *2D* и *3D* графикой на основе языка *AutoLisp*. Кроме этого, *AutoCAD* имеет множество надстроек, которые позволяют автоматизировать процесс проектирования в различных областях производства, например, архитектуре, машиностроении и др.

Студентам, изучающим технические дисциплины, связанные с электротехникой и электроникой будет полезно знакомство с программами электронной лаборатории *Electronical WorkBench* и *MicroCap*, помогающими освоить новый подход, основанный визуальном составлении электрических схем из списков элементов и имитационного моделирования режимов работы.

Огромной популярностью в области решения задач экономического анализа и финансового учета пользуются программы серии 1С: Предприятие. В зависимости от поставляемой конфигурации, программы этой серии могут автоматизировать операции складского, бухгалтерского,

кадрового и др. учета. Масштабы автоматизации с помощью программ 1С: Предприятие различны: начиная от индивидуального предприятия и заканчивая крупной корпорацией (1С: Предприятие + *SQL*).

Современные системы программирования позволяют разрабатывать программы по технологии быстрой разработки приложений *RAD* (*Rapid Application Development*). Эта технология состоит из этапа визуального конструирования интерфейса приложения и этапа разработки программного кода, который будет обрабатывать действия пользователя с компонентами интерфейса. Наиболее мощным и включающим значительное число языков программирования является *Microsoft Visual Studio.Net*. Эта среда разработки программ для операционной системы Windows является стандартом де-факто, и другие разработчики вынуждены подстраиваться под программирование в среде *Microsoft Net Framework*. Примером может служить изначально ориентированная на язык Паскаль, а теперь включающая и другие языки программирования *RAD Borland Delphi* (версии 2005). *Visual Studio.Net* существует в четырех версиях: *Professional*, *Enterprise Developer*, *Enterprise Architect* и *Academic*. Пользователь может разрабатывать программу на одном из четырех основных языков программирования: *Visual Basic*, *Visual C++*, *Visual C#* и *Visual J#*. Следует, однако, заметить, что в пакет программ *Microsoft Office* встроен язык программирования *Visual Basic for Applications* (*VBA*). В большинстве случаев, связанных с автоматизацией офисной деятельности, пользователь может обойтись его применением.

Значительное число прикладных программ появляется в связи с ростом интернет-технологий и электронных телекоммуникаций. Работая с *Web*-серверами Internet, пользователь прибегает к услугам программы-браузера: *Internet Explorer*, *Opera*, *Netscape Navigator*. Какой из них лучше — зависит от вкуса самого пользователя, но объективно можно утверждать, что для удобной работы с электронной почтой понадобится более совершенная программа, например, *The Bat!*. Лучшим вариантом, чем браузер, для «скачивания» файловых архивов, наверняка будет специализированная программа типа *Reget*, *Flashget*, *Download Master*. Не следует забывать и о программах фильтрации интернет-трафика, которые также называют брандмауэрами или файерволами (*firewall*). Эта «огненная стена» защитит мирного пользователя Internet от нежелательного воздействия злоумышленников на его компьютер, а также избавит от ненужной рекламы и «всплывающих» окон, которые отнимают время соединения или перегружают трафик пользователя. Среди них стоит упомянуть о программе *OutPost Firewall*. Пожалуй, эта группа программ на сегодняшний день пополняется наиболее интенсивно, являясь одной из самых востребованных.

ГЛАВА 3

ОПЕРАЦИОННЫЕ СИСТЕМЫ

3.1. Основные сведения из теории операционных систем

▶ *Операционная система* (ОС) — это совокупность программных средств, осуществляющих управление ресурсами персонального компьютера (ПК), запуск прикладных программ и их взаимодействие с внешними устройствами и другими программами, а также обеспечивающих диалог пользователя с компьютером.

ОС классифицируются:

- по количеству одновременно работающих пользователей: одно- и многопользовательские;
- по числу процессов, одновременно выполняемых под управлением системы: одно- и многозадачные;
- по количеству поддерживаемых процессоров: одно– и многопроцессорные;
- по разрядности кода ОС: 32– и 64-разрядные;
- по типу интерфейса: командные (текстовые) и объектно-ориентированные (графические);
- по типу доступа пользователя к ПК: с пакетной обработкой, с разделением времени, реального времени;
- по типу использования ресурсов: локальные и сетевые.

▶ *Операционная среда* — набор функций и сервисов ОС и правила обращения к ним. Это также набор интерфейсов, необходимый программам и пользователям для обращения к ОС с целью получить определенные сервисы. Операционная система в общем случае может содержать несколько операционных сред. Операционная среда может включать несколько интерфейсов: *пользовательские* и *программные*. Она предоставляет системное программное окружение, в котором могут выполняться программы, созданные по правилам работы этой среды.

▶ *Приложение ОС* — прикладная программа, которая не относится к компонентам ОС, но работает под ее управлением.

Пользователь взаимодействует с ОС на уровне пользовательского интерфейса (*UI — User Interface*). Программист взаимодействует с ОС на уровне программного интерфейса (*API — Application Program*

Interface). На рис. 3.1 представлено взаимодействие пользователей, программистов и операционной системы. ОС, как интерфейс между пользователем и аппаратными средствами компьютера, на прикладном уровне, предоставляет возможности исполнения программ, доступ к периферийным устройствам ввода-вывода и файловой системе. На программном уровне пользователи ОС могут разрабатывать свои прикладные и системные приложения, пользуясь интерфейсами программирования, например, в ОС Windows XP базовым является программный интерфейс *Win32 API*. Существуют также интерфейсы *MAPI*, *TAPI*, *CryptoAPI* и др. Программный интерфейс ОС представляет собой документированные разработчиком, вызываемые на языках программирования высокого уровня подпрограммы. В современных ОС наборы вызываемых подпрограмм *API* сходного назначения объединяются в один двоичный файл, который приложение может использовать, динамически загружая в процессе своей работы — динамически подключаемую библиотеку (*DLL*).

ОС, играя роль посредника, служит двум *целям*:

- эффективно использовать ресурсы вычислительной системы;
- создавать условия для эффективной работы пользователя.

Основные *функции* ОС:

- распределение ресурсов;

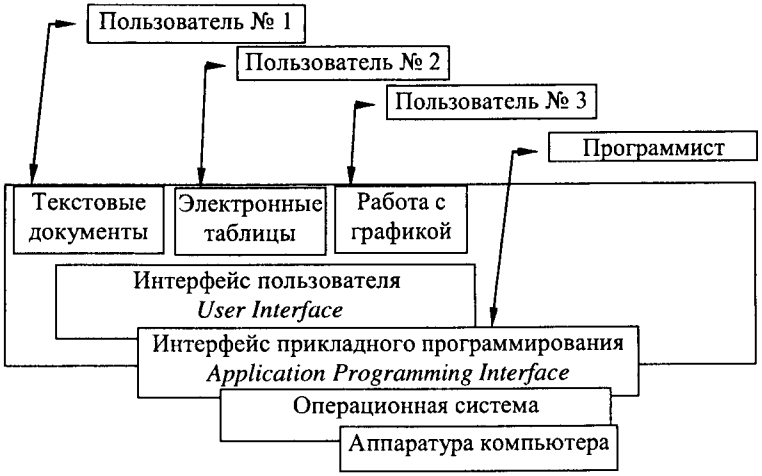


Рис. 3.1

- управление заданиями, задачами, данными, памятью, процессорами, устройствами ввода-вывода;
- организация режимов работы (пакетного, однопрограммного, мультипрограммного, реального);
- выполнение программ;
- разработка программ;
- отладка программ.

Компонентный состав ОС определяется набором функций, для выполнения которых она предназначена. Все программы, входящие в её состав, можно разбить на две группы: управляющие программы и системные обрабатывающие программы (рис. 3.2).

Управляющая программа (*микроядро* ОС) — обязательный компонент, выполняющий следующие функции:

- планирование прохождения непрерывного потока заданий;
- управление распределением ресурсов;
- реализация принятых методов организации данных;
- управление операциями ввода-вывода;

Рис. 3.2

- организация мультипрограммной работы;
- управление работоспособностью системы после сбоев.

При управлении статическими ресурсами (заданиями) микроядро ОС осуществляет предварительное планирование потока заданий для выполнения и статическое распределение ресурсов между одновременно выполняемыми заданиями в процессе подготовки к выполнению (инициализации). К таким ресурсам относятся разделы памяти (оперативной, дисковой, внешней) и устройства, допускающие только монопольное использование. Такие ресурсы закрепляются за заданием или его частью с момента его инициализации до момента завершения. Обычно такое управление носит название мониторинга, а программы, его осуществляющие — *мониторы*.

Управление динамическими ресурсами (задачами) осуществляет динамическое распределение ресурсов между несколькими задачами, решаемыми одновременно в мультипрограммном режиме для выполняемого потока заданий. Динамическое управление выполняет программа *супервизор* (планировщик).

Управление данными обеспечивает все операции ввода-вывода (обмена между оперативной памятью и периферийными устройствами) на физическом и логическом уровнях. К таким функциям относят, например, организацию необходимых различных информационных структур, таблиц во время работы ОС, управление файловой системой, каталогами, режимом прямого доступа к памяти, обработку ошибок ввода-вывода и т.д.

Управление восстановлением регистрирует машинные сбои и отказы и восстанавливает работоспособность системы после их возникновения, если это возможно.

В качестве основных аппаратных ресурсов компьютера, подлежащих распределению, обычно рассматривают:

- время работы процессора;
- адресное пространство основной памяти;
- оборудование ввода — вывода;
- файлы, хранящиеся во внешней памяти.

Основными подсистемами, обеспечивающими распределение указанных ресурсов ОС, являются:

1) подсистема управления процессами (распределяет ресурс «процессорное время»);

2) подсистема управления памятью (распределяет ресурс «адресное пространство основной памяти»);

3) подсистема управления устройствами (распределяет ресурсы «оборудование ввода – вывода»;

4) подсистема управления данными (распределяет ресурс «данные или файлы»).

Функционирование ОС было бы бесполезным, если не было бы возможности выполнять пользовательские программы, представляющие собой статические наборы команд центрального процессора — *процессы*. Дать исчерпывающее определение процесса труднее, чем выделить его характеристики:

- адресное пространство памяти, которое доступно процессу;
- код программы и данные, помещаемые в адресное пространство для исполнения и обработки;
- идентификатор процесса и его контекст, то есть информация о состоянии аппаратных средств, выделенных процессу, и стадии его выполнения;
- список ресурсов компьютера, которые требуются процессу;
- средства защиты памяти процесса от постороннего вмешательства в код и данные процесса;
- минимум один поток, то есть программу, выполняемую процессором.

Одно из основных условий функционирования ОС — стабильность и устойчивость — приводят к четкому разделению процессов на две группы: *системные* и *прикладные*. Первая группа, системные процессы, выполняются в режиме ядра ОС. К ней относится сам код ОС и программы, взаимодействующие с устройствами компьютера на уровне машинных команд — драйверы устройств. Этим процессам разрешается доступ ко всей физической системной памяти ПК (адресному пространству памяти ОС), и выполнение любых машинных команд микропроцессора, в том числе привилегированных. Пользовательские процессы работают только в своем адресном пространстве, не имея возможности повлиять на ход функционирования системных процессов, и даже могут быть принудительно завершены последними. С точки зрения программирования, пользовательским процессам доступны библиотеки подпрограмм и функции *API*, которые при необходимости переходят на уровень системных процессов.

При исполнении программ на процессоре различаются следующие характерные отдельные состояния:

- *порождение* — подготавливаются условия для первого исполнения на процессоре;

- *активное* состояние — программа исполняется на процессоре;
- *ожидание* — программа не исполняется на процессоре по причине занятости какого-либо требуемого ресурса;
- *готовность* — программа не исполняется, но для исполнения предоставлены все необходимые в текущий момент ресурсы, кроме процессора;
- *окончание* — нормальное или аварийное окончание исполнения программы, после которого процессор и другие ресурсы ей не предоставляются.

Процесс находится в каждом из своих допустимых состояний в течение некоторого времени, после чего переходит в какое-либо другое допустимое состояние. Состав допустимых состояний обычно задается в виде графа (рис. 3.3).

Процессы определяются рядом временных характеристик. В некоторый момент времени процесс может быть порожден, а через некоторое время закончен. Интервал между этими моментами *называется интервалом существования процесса.*

Процессы можно разделить на следующие классы:

1) обычные (пакетные) — длительность интервала существования не определенная;

2) интерактивные — интервал существования не более времени реакции компьютера на запросы пользователя;

3) реального времени — интервал существования процесса менее некоторого определенного момента времени (гарантированного интервала).

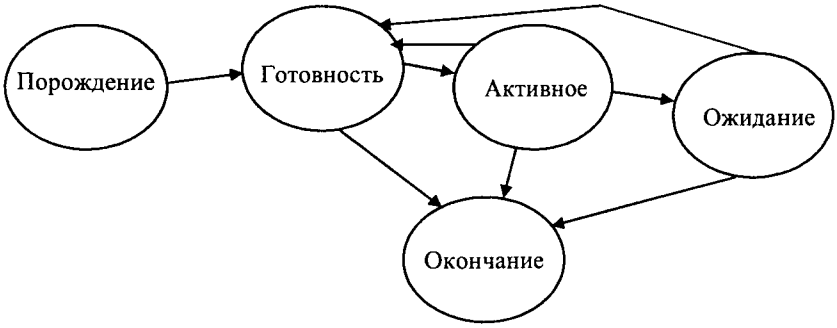


Рис. 3.3

Один процесс может порождать следующий, при этом порождающий процесс будет «предком», а порожденный — «потомком». Порядок длительности и пребывания процесса в допустимых состояниях на интервале существования называется *трассой процесса*.

По способу достижения конечного результата обработки информации процессы делятся на:

1) эквивалентные — при одинаковых исходных данных, различных программах обработки информации и трассах процессов достигается одинаковый конечных результат;

2) тождественные — при одинаковых входных данных, одинаковых программах обработки информации и трассах процессов достигается одинаковый результат;

3) равные — трассы процессов и все остальное совпадают.

Во всех остальных случаях процессы различны.

По времени существования друг относительно друга процессы делятся на:

1) последовательные — интервалы существования процессов не пересекаются во времени;

2) параллельные — на рассматриваемом интервале времени процессы существуют одновременно;

3) комбинированные — на рассматриваемом интервале времени есть точка, в которой существует один процесс и не существует другой, и точка, в которой два процесса существуют одновременно.

По принадлежности к ОС процессы могут быть:

1) системные;

2) пользовательские.

По связности:

1) изолированные — нет связей друг с другом;

2) информационно-независимые — процессы совместно используют некоторые ресурсы, но не обмениваются информацией;

3) взаимодействующие — процессы взаимосвязаны обменом информацией;

4) конкурирующие — процессы взаимосвязаны по ресурсам.

Порядок выполнения процессов определяет ряд отношений между ними, которые в совокупности называются синхронизирующими правилами.

Отношение предшествования. Означает, что первый процесс должен переходить в активное состояние всегда раньше второго.

Отношение приоритетности. Процесс с некоторым приоритетом *P* может быть переведен в активное состояние только при соблюдении двух

условий: в состоянии готовности этого процесса не существует процессов с большим приоритетом, и процессор либо свободен, либо используется процессом с меньшим, чем P, приоритетом.

Отношение взаимного исключения. Если несколько процессов используют один общий ресурс, то совокупность действий над этим ресурсом в составе одного процесса называют *критической секцией.* Другими словами, критическая секция одного процесса не должна выполняться одновременно с критической секцией другого процесса над одним и тем же ресурсом.

С процессом связано понятие ресурса. Термин *ресурс* относится к используемым, относительно стабильным и часто недостающим объектам аппаратной части компьютера, которые запрашиваются, используются и освобождаются процессами в период их активности. Некоторые ресурсы могут быть разделяемыми, например, оперативная память, диски, процессор и использоваться либо одновременно (в один и тот же момент времени), либо псевдопараллельно (в течение некоторого отрезка времени процессы используют ресурс попеременно).

В общем случае ресурс — это запас неких материальных предметов в составе некоторого объекта.

▶ *Ресурс* вычислительной системы — это средство компьютера, которое может быть выделено процессу на определенный интервал времени.

Ресурсы можно классифицировать по следующим признакам.

По реальности существования:

1) *физические;*
2) *виртуальные.*

Главным виртуальным ресурсом можно считать память компьютерной системы, причем следует учесть, что объем физической оперативной памяти гораздо меньше, чем виртуальной оперативной памяти. Виртуальная оперативная память организуется за счет дисковой памяти в результате выгрузок (*swapping*) программ из первой и замещения программами из второй.

По характеру использования:

1) *параллельно-используемый;*
2) *последовательно-используемый.*

По возможности расширения свойств:

1) *жесткий;*
2) *расширяемый.*

По активности:

1) *активный;*
2) *пассивный.*

По времени существования:
1) *постоянный;*
2) *временный.*
По степени важности:
1) *главный;*
2) *второстепенный.*
По функциональной избыточности:
1) *дорогой;*
2) *дешевый.*
По структуре:
1) *простой;*
2) *составной.*
По восстанавливаемости:
1) *воспроизводимый;*
2) *потребляемый.*

Физический ресурс реально обладает всеми физическими характеристиками. Виртуальный ресурс — некоторая модель конкретного физического ресурса. Он реализуется в какой-либо программно-аппаратной форме, но при работе виртуальный ресурс предоставляет пользователю не только часть свойств, которые присущи физическому ресурсу, но и часть свойств, которые ему не присущи. Например, виртуальная оперативная память компьютера. Объем виртуальной оперативной памяти может значительно превосходить реальный физический объем памяти компьютера путем имитации её за счет механизма подкачки страниц памяти.

Жесткие ресурсы не допускают виртуализации, иначе ресурс — расширяемый. Активные ресурсы способны выполнять действия по отношению к другим ресурсам или процессам, которые приводят к изменению последних. Если ресурс существует до момента порождения процесса, то он рассматривается как постоянный. Временный ресурс может появляться и уничтожаться динамически в течение времени существования рассматриваемого процесса.

Процесс выполняет в отношении ресурсов три типа действий:

1) запрос — в ответ на него ОС выделяет процессу некоторый ресурс, либо отказывает в его выделении;

2) использование — процесс использует некоторый ресурс в соответствии со своей программой;

3) освобождение — выполняется по требованию процесса и переводит ресурс в состояние «свободен».

Если ресурс допускает многократное выполнение действий ЗАПРОС-ИСПОЛЬЗОВАНИЕ-ОСВОБОЖДЕНИЕ, то он называется воспроизводимым, иначе — потребляемым. Примером потребляемого ресурса может служить буфер ввода-вывода, при записи в который предыдущая информация теряется, что эквивалентно повторному выделению этого ресурса.

В целях повышения эффективности обработки данных ОС обладают свойством многозадачности. Многозадачность позволяет выполнять несколько программ одновременно (но не параллельно, так как большинство персональных компьютеров однопроцессорные) в режиме разделения времени. Это приводит к появлению в составе ОС подсистем, отвечающих за временное планирование и управление памятью, выделяемой исполняемым программам с условиями защиты её от вмешательства со стороны других программ.

Многозадачный режим работы вычислительной системы заключается в том, что пока одна программа (процесс, задача) ожидает завершения очередной операции ввода/вывода, другая программа (задача) может быть поставлена на выполнение. При многозадачности повышается пропускная способность системы, но отдельный процесс никогда не может быть выполнен быстрее, чем если бы он выполнялся в однопрограммном режиме. Операционная система старается эффективно использовать ресурсы путем организации очередей запросов. При необходимости использовать какой-либо ресурс процесс обращается к *супервизору*. ОС сообщает ему свои требования (вид ресурса, объем и т.д.). Эта директива переводит процессор в привилегированный режим, если он есть.

▶ *Супервизор* — часть операционной системы, отвечающая за перераспределение ресурсов между процессами.

Обычно супервизор работает по следующим правилам. Получив управление, супервизор действует в соответствии с запрограммированной стратегией выполнения процессов. В зависимости от приоритета процесса, супервизор может выделять ему необходимые для успешного выполнения ресурсы. Ресурс будет выделен обратившемуся за ним процессу, если: a) он свободен, и нет задач с более высоким приоритетом, обратившимся за этим ресурсом; b) текущий запрос и ранее выданные запросы допускают совместное использование ресурсов; c) ресурс используется задачей с более низким приоритетом и может быть временно отобран. Если ресурс занят, супервизор ставит процесс в очередь к ресурсу, переводя его в состояние ожидания. Очередь к ресурсу может быть организована различными способами, но обычно с помощью списко-

вой структуры. Если требуемые ресурсы готовы для использования процессом, то супервизор передает ему управление, и процесс начинает выполняться. После завершения работы с ресурсом процесс с помощью системного вызова супервизора сообщает об отказе от ресурса и освобождает его для других процессов.

За время функционирования ОС один и тот же процесс может выполняться многократно. Это обусловлено обращениями к операционной системе с запросами ресурсов, выполнением системных функций, взаимодействием с другими процессами, появлением сигналов прерывания от таймера, каналов и устройств ввода/вывода и других устройств. Основных *состояний процесса* два: *активное*, когда процесс участвует в конкуренции с другими процессами за ресурсы компьютера, и *пассивное*, когда процесс известен ОС, но в отношении ресурсов не проявляет никаких действий. *Активный процесс* может находиться в одной из следующих стадий обработки данных:

- *выполнения*, когда все затребованные ресурсы выделены, ведется обработка данных процесса центральным процессором (в однопроцессорных системах в этой стадии может находиться только один процесс);
- *готовности к выполнению*, когда готовы к обработке данных процесса все ресурсы, кроме процессора;
- *блокирования или ожидания*, когда затребованные ресурсы не могут быть предоставлены процессу или не завершена операция ввода/вывода;
- *завершения*, когда обработка данных процесса полностью завершена.

Существует комплекс правил перехода между стадиями выполнения процессов, связанных либо с системными событиями, либо с действиями пользователя. Процесс из стадии *блокирования* в стадию *готовности* может перейти в следующих случаях:

- по команде оператора (пользователя);
- при выборе из очереди супервизором;
- по вызову из другой задачи (посредством обращения к супервизору один процесс может создавать, инициировать, приостановить, остановить или уничтожить другой процесс);
- по прерыванию от внешнего «инициативного» устройства;
- при наступлении запланированного времени запуска программы.

Из стадии *выполнения* процесс может выйти по одной из следующих причин:

- процесс завершается, при этом он посредством обращения к супервизору передает управление ОС и сообщает о своем завершении. Супервизор уничтожает процесс или переводит его в список *бездействующих* процессов. В состояние *бездействия* процесс может быть переведен принудительно по команде оператора или путем обращения к супервизору другой задачи, требующей остановить данный процесс;
- процесс переводится супервизором ОС в состояние *готовности к исполнению* в связи с появлением более приоритетной задачи или в связи с окончанием выделенного ему кванта времени;
- процесс *блокируется* из-за невозможности предоставить ему ресурс или вследствие запроса ввода/вывода, а также по команде оператора на приостановку задачи.

Процесс деблокируется и переводится в стадию *готовности к выполнению* при наступлении соответствующего события:

- завершение операции ввода/вывода;
- освобождение затребованного ресурса;
- загрузка в оперативную память страницы виртуальной памяти и т. д.

Предпосылками изменения состояния процесса являются события. Один из основных видов событий — прерывания.

3.2. ОС семейства Windows

В настоящее время существует несколько различающихся направлений в семействе ОС Windows:

- Windows NT/2000;
- Windows XP;
- Windows 2003 Server.

Каждое направление состоит из некоторого числа модификаций версий Windows, что позволяет выделить две области применения:

1) версии персональные, такие как Windows XP Home Edition;

2) версии для рабочих станций сетей, Windows 2000 Professional, Windows XP Professional;

3) версии для серверов сетей, Windows 2000 Server, Windows 2000 Advanced Server, Windows 2000 Datacenter Server, Windows 2003 Server.

Среди общего числа версий ОС Windows есть преемники принципиально нового направления, технологии NT (*New Technology*, разрабатываемой *Microsoft* с 1989 г.). К ОС технологии NT предъявляется ряд

повышенных, в сравнении с потребительскими версиями Windows, требований, таких как поддержка многопроцессорных систем, вытесняющая многозадачность, работа с виртуальной памятью, защищенная файловая система и др. К ним относятся все версии операционных систем Windows 2000 и Windows XP. Системные файлы этих модификаций Windows одинаковы, как и ядро ОС. Они различаются между собой по числу поддерживаемых процессоров, объему поддерживаемой физической памяти, одновременному числу сетевых подключений и наличием дополнительных сетевых сервисов.

В сравнении с предыдущими версиями (Windows 95/98/ME), у ОС технологии *NT* имеется ряд существенных отличий. Первое принципиальное отличие технологии *NT* состоит в повышении надежности работы ядра системы и исключении возможности зависания ОС из-за зависания или некорректной работы пользовательской программы. Второе принципиальное отличие технологии *NT* — возможность использования не только файловых систем из предыдущих версий Windows, называемых *FAT16* и *FAT32*, но и значительно более безопасной и надежной файловой системы *NTFS* (*NT File System*). Использование *NTFS* повышает безопасность компьютера, так как эта файловая система допускает защиту данных (файлов и папок) путем их шифрования и возможности установки запрета на доступ к ним. Кроме этого, *NTFS* обеспечивает более высокую степень сжатия информации и полную поддержку разделов и файлов большого размера. Третье существенное отличие технологии *NT* заключается в том, что на одном компьютере можно запускать не только версии Windows 2000/XP, но и более ранние версии Windows 95/98/ME. Для этого используется конфигурация с двойной загрузкой, и на этапе загрузки компьютера можно выбрать ту операционную систему, которая требуется в этом сеансе работы.

3.2.1. Общая структура ОС Windows XP

Windows XP имеет модульную структуру, в которой код ОС и драйверы выполняются в привилегированном режиме процессора (режиме ядра), обеспечивающим полный доступ ко всей аппаратной части компьютера, а пользовательские приложения выполняются в непривилегированном режиме процессора, называемом пользовательским режимом ОС без прямого доступа к оборудованию компьютера. Упрощенная структура Windows XP представлена на рис. 3.4.

В режиме ядра работают следующие компоненты.

Рис. 3.4

1. *Уровень абстрагирования от оборудования* (*Hardware Abstraction Layer, HAL*). Его задачей является отделение ОС от особенностей конкретных реализаций в аппаратном обеспечении ПК, т.е. от различий в материнских платах, модификациях процессоров, наборах микросхем и др. Благодаря этому уровню управление подсистемами прерываний, прямого доступа к памяти, системными шинами, таймерами для ядра ОС является одинаковыми. Уровень *HAL* реализован в системном файле *Hal.dll*.

2. *Ядро* содержит наиболее часто вызываемые низкоуровневые функции ОС, планирование и распределение ресурсов между процессами, их переключение и синхронизацию. В обязанности ядра входит также управление прерываниями и обработка ошибочных ситуаций при функционировании ОС. Код ядра Windows XP не разделяется на потоки, находится только в оперативной памяти и не может быть выгружен на диск. Код ядра Windows XP находится в системном файле *Ntoskrnl.exe*.

3. *Драйверы устройств* представляют собой подпрограммы, транслирующие вызовы, поступившие от пользовательских программ в запросы обработки данных для конкретных устройств. Значительное число драйверов устройств входит в состав Windows XP (они располагаются в подкаталоге */system32/drivers* системного каталога и имеют тип файла **.sys*, например, драйвер дисковой подсистемы находится в файле

disk.sys), а для нестандартных периферийных устройств драйверы находятся в комплектах поставки.

4. *Исполняющая подсистема* (*NT Executive*) состоит из *микроядра* и подсистем *диспетчеризации управления программами* и доступом к виртуальной памяти, окнам и графической подсистеме. Виртуальная память предоставляет пользовательским программам виртуальные адреса адресного пространства процессов и соответствующие физические страницы оперативной памяти ПК. Графическая подсистема предназначена для создания оконного интерфейса, рисования элементов управления, расположенных в окнах. К исполняющей подсистеме относятся системные файлы *Ntkrnlpa.exe*, *Kernel32.dll*, *Advapi32.dll*, *User32.dll*, *Gdi32.dll*.

ОС Windows XP в значительной мере использует возможности процессоров, совместимых с семейством *Intel* x86. В их аппаратной архитектуре предусматривается четыре уровня привилегий выполнения кода программ от 0-го наивысшего привилегированного, до 4-го пользовательского режима с ограниченным набором команд процессора. Программы режима ядра ОС Windows XP функционируют в нулевом, защищенном и привилегированном режиме, а все остальные пользовательские программы работают в менее привилигированных режимах, таким образом, находясь под контролем у программ режима ядра. Операции, недоступные в пользовательском режиме, приложения (а также многие подсистемы самой Windows XP, которые не работают в режиме ядра) обращаются к системным вызовам ядра ОС, которые также называют *Win32 API*. В состав этого *API* входит более 250 функций, обращение к которым осуществляется при помощи системных вызовов, основанных на подпрограммах ядра ОС. Все вызовы *Win32 API* обслуживаются как системными службами *NT*, так и модулем *NT Executive* — исполняющей системы Windows XP. Модуль *NT Executive* представляет собой несколько программных потоков, которые выполняются в режиме ядра. Код практически всех подсистем этого модуля находится в файле *ntoskrnl.exe*, кроме подсистемы *win32*, код которой расположен в файле *win32k.sys*, и уровня абстрагирования от оборудования *HAL*, который содержится в файле *hal.dll*. *NT Executive* сосредоточивает все самые важные части ОС, которые следует рассмотреть подробнее.

Микроядро отвечает за выделение памяти для приложений и распределение процессорного времени, то есть, фактически, за реализацию многозадачности. Для этого в состав микроядра входит так называемый планировщик потоков (*threads scheduler*), который назначает каждому из потоков один из 32 уровней приоритета. Уровень 0 зарезервирован для системы. Уровни от 1 до 15 назначаются исполняемым програм-

мам, а уровни от 16 до 31 могут назначаться только администраторами. Планировщик делит все процессорное время на кванты фиксированного размера. При этом каждый программный поток выполняется только в течение отведенного ему времени, и если к окончанию кванта он не освобождает процессор, планировщик в принудительном порядке приостанавливает этот поток и меняет программное окружение процесса, настраивая его на выполнение другого потока, обладающего тем же приоритетом. Микроядро также осуществляет всю работу, связанную с обработкой программных и аппаратных прерываний.

Диспетчеризация управления программами состоит из следующего набора системных программ. *Диспетчер ввода-вывода* — интегрирует добавляемые в систему драйверы устройств в операционную систему Windows XP. *Диспетчер объектов* — служит для управления всеми разделяемыми ресурсами компьютера. В момент обращения приложения или службы к какому-либо ресурсу диспетчер объектов сопоставляет этому ресурсу объект (к примеру, окно) и отдает приложению дескриптор (№ окна) этого объекта. Используя дескриптор, приложение взаимодействует с объектом, совершая в его отношении различные операции. Монитор системы безопасности следит при этом за тем, чтобы с объектом выполнялись только разрешенные действия. *Диспетчер процессов* — предоставляет интерфейс, при помощи которого другие компоненты Windows *NT Executive*, а также приложения пользовательского режима могут манипулировать процессами и потоками. Во время работы диспетчер процессов сопоставляет каждому процессу и потоку идентификатор процесса (*PID — Process Identifier*) и потока (*TID — Thread Identifier*) соответственно, а также таблицу адресов и таблицу дескрипторов. *Диспетчер виртуальной памяти* — служит для управления и организации подсистемы памяти, позволяет создавать таблицы адресов для процессов и следит за корректностью использования адресного пространства приложениями (то есть, следит за общим доступом к памяти и осуществляет защиту страниц в режиме копирования при записи). Диспетчер виртуальной памяти также обеспечивает возможность загрузки в оперативную память исполняемых файлов и файлов динамических библиотек. Диспетчер виртуальной памяти представляет физическую память для пользовательских приложений таким образом, что каждому процессу выделяются отдельные 4 Гбайта виртуального адресного пространства, из которых младшие 2 Гбайта используются непосредственно процессом по своему усмотрению, а старшие 2 Гбайта отводятся под нужды системы, причем они — общие для всех процессов. Каждый про-

цесс работает в своем изолированном адресном пространстве и не знает о других работающих процессах. Обмениваться данными процессы могут через разделяемую память, которая может быть спроецирована на виртуальное адресное пространство нескольких процессов, и таким образом они смогут взаимодействовать друг с другом. Другими словами, главная задача диспетчера виртуальной памяти — организация логической памяти, размер которой больше размера физической, установленной на компьютере. Этот трюк достигается благодаря тому, что страницы памяти, к которым долго не было обращений, и которые не имеют атрибута неперемещаемых, сохраняются диспетчером в файле pagefile.sys на жестком диске и удаляются из оперативной памяти, освобождая ее для других приложений. В момент, когда происходит обращение к данным, находящимся в перемещенной на винчестер странице, диспетчер виртуальной памяти незаметно для приложения копирует страницу обратно в оперативную память, и только затем обеспечивает доступ к ней. Этот механизм обеспечивает выделение дополнительной памяти программам, которые нуждаются в ней, и при этом следит за тем, чтобы все работающие в системе программы обладали достаточным объемом физической памяти для того, чтобы продолжать функционирование. *Диспетчер кэша* применяется для кэшированного чтения и записи и позволяет существенно ускорить работу таких устройств, как винчестеры и др. При этом наиболее востребованные файлы дублируются диспетчером кэша в оперативной памяти компьютера, и обращение к ним обслуживается с использованием этой копии, а не оригинала, расположенного на сравнительно медленном долговременном носителе. Кэш в Windows XP является единым для всех логических дисков, вне зависимости от используемой файловой системы. Кроме того, он является динамическим, а это значит, что диспетчер управляет его размерами в зависимости от доступного объема свободной физической памяти в каждый конкретный момент. *Диспетчеры окон и графики* выполняют все функции, связанные с пересылкой системных сообщений и отображением информации на экране.

Процесс функционирования Windows XP можно разделить на три фазы: процесс начальной нагрузки, штатный режим работы и завершение работы.

3.2.2. Процесс начальной загрузки

Для загрузки Windows XP необходим следующий минимальный набор файлов, расположенных:

1) в корневом каталоге загрузочного диска

- *Ntldr*
- *Boot.ini*
- *Bootsect.dos* (необходим только при использовании мультизагрузки)
- *Ntdetect.com*

2) в системном подкаталоге */system*32
- *Ntoskrnl.exe*
- *Hal.dll*
- разделы реестра *SYSTEM*

3) в системном подкаталоге */system*32*/drivers*
- необходимые драйверы устройств

Процесс загрузки компьютера начинается с процедуры начального тестирования оборудования (*POST — Power-On Self Test*). Код, выполняющий *POST*, зашит в базовой системе ввода-вывода (*BIOS*) каждого компьютера, и именно ему передается управление при включении питания. Если в процессе тестирования обнаруживаются какие-либо ошибки, то *BIOS* генерирует коды ошибок (*POST codes*), которые отличаются для *BIOS* разных производителей, и звуковые коды. Если процедура *POST* завершается успешно, то *BIOS* передает управление главной загрузочной записи (*MBR — Master Boot Record*) первичного жесткого диска системы, и можно сказать, этим завершается первая «аппаратная» стадия загрузки компьютера (весь процесс зависит только от аппаратуры компьютера, но не от установленного программного обеспечения).

На второй стадии загрузочная запись, оперируя данными о разбиении жесткого диска на логические тома, передает управление исполняемому коду, расположенному в загрузочном секторе. В ОС Windows XP этим кодом является загрузчик ОС *Ntldr*. Первое, что делает загрузчик, это переходит в защищенный режим и производит необходимые для успешного функционирования в этом режиме манипуляции с памятью. Кроме функций, позволяющих работать с памятью, *Ntldr* имеет также несколько модулей, позволяющих работать с некоторыми другими базовыми ресурсами системы, в первую очередь с файловой системой. Все другие действия выполняются с помощью вызова прерываний *BIOS*.

После первичной инициализации загрузчик предоставляет пользователю возможность выбрать операционную систему, которая будет загружена, из списка систем, установленных на компьютере (то есть, *Ntldr* выводит на экран надпись *OS Loader V*5.0 и приглашение выбрать операционную систему; это сообщение выводится только в том случае, если в файле *boot.ini* зарегистрировано более одной ОС), после чего, если выбрана Windows XP, начинает загрузку файлов ОС.

После выбора операционной системы загрузчик запускает *Ntdetect.com*. Этот компонент считывает из *CMOS*-памяти системную дату и время, после чего производит поиск и распознавание аппаратных средств, подключенных в данный момент к компьютеру. Завершив работу, *Ntdetect* возвращает управление и собранную им информацию обратно в *Ntldr*.

Далее загружается и инициализируется ядро *Ntoskrnl.exe* и уровень абстрагирования от оборудования *Hal.dll*. При своей инициализации ядро производит ряд действий в следующей последовательности:

- инициализация диспетчера памяти;
- инициализация диспетчера объектов;
- установка системы безопасности;
- настройка драйвера файловой системы;
- загрузка и инициализация диспетчера ввода-вывода (обычно — самая длительная фаза);
- последняя стадия — загрузка системных сервисов, которые, собственно, и реализуют взаимодействие с пользователем.

Список основных системных сервисов следующий.

1) *Smss.exe* — диспетчер сеансов. Он управляет другими сервисами и службами Windows, в том числе запускает *Win*32 (*Csrss*) и некоторые системные утилиты, выполняемые на этапе загрузки. Еще две важные функции — это реализация графического пользовательского интерфейса и запуск процессов *Csrss.exe* и *WinLogon.exe*. Этот диспетчер запускается, тем не менее, в самом конце загрузки.

2) *Csrss.exe* — данный модуль предназначен, главным образом, для организации взаимодействия между компьютером и пользователем.

3) *Lsass.exe* — служба, запускаемая *WinLogon.exe* и отвечающая за безопасность системы. Она предоставляет возможность пользователю зарегистрироваться в системе, и только после того, как в системе зарегистрировался хотя бы один пользователь, загрузка считается успешной.

Уже после загрузки операционной системы пользователь, чтобы доказать, что он тот, за кого себя выдает, должен пройти процедуру аутентификации, то есть ввести собственное регистрационное имя (или на жаргоне — логин) и пароль. Заметим, что данные действия при пониженных требованиях к безопасности могут быть настроены по умолчанию. Процедура подключения к системе позволяет определить, кем является пользователь и обладает ли он правом входа и работы с системой. Эту процедуру выполняет служба *WinLogon*. При этом в системе происходят следующие события:

1) процесс *WinLogon* отображает на экране фон рабочего стола (к этому моменту объект рабочего стола уже создан, но еще не отображается), а также приглашение к вводу пользователем логина и пароля; введенные данные передаются подсистеме безопасности;

2) подсистема безопасности обращается к базе данных *SAM* (*Security Accounts Manager*) и проверяет, обладает ли пользователь полномочиями работы с системой;

Если пользователь является авторизированным пользователем системы, то подсистема безопасности формирует для него идентификатор доступа, который вместе с управлением передает обратно процессу *WinLogon*.

Процесс *WinLogon* посредством обращения к подсистеме *Win*32 создает новый процесс для пользователя и прикрепляет ему только что созданный идентификатор доступа. Каждый процесс, в дальнейшем создаваемый пользователем, отмечается принадлежащим этому пользователю идентификатором доступа. Таким образом, любые попытки доступа пользователя к ресурсам системы контролируются и отслеживаются. Более того, всегда точно известно, кто инициировал то или иное действие в системе. Благодаря обязательной процедуре подключения к системе упрощается реализация таких механизмов, как аудит системы и квоты на использование ресурсов. Помимо прочих данных, пользовательский идентификатор доступа содержит идентификатор пользователя, а также идентификаторы всех групп, к которым принадлежит данный пользователь.

Следует отметить, что, если ОС не загружается корректно, то при нажатии в процессе загрузки ОС Windows XP клавиши *F*8 происходит переход в расширенное меню запуска, содержащее пункты:

- *Безопасный режим* — загрузка Windows XP с минимальным требуемым количеством системных файлов и драйверов устройств;
- *Безопасный режим с загрузкой сетевых драйверов* — как и предыдущий, но с поддержкой подключения к сети;
- *Безопасный режим с поддержкой командной строки* — такой же режим, как и *Безопасный режим,* за исключением того, что загружает режим командной строки, а не *GUI* (*Graphical User Interface*);
- *Включить протоколирование загрузки* — позволяет записать этапы загрузки ОС Windows XP в файл *Ntbtlog.txt*;
- *Включить режим VGA* — режим, загружающий драйвер стандартного монитора *VGA* с разрешением 640 на 480 точек на дюйм и 16 цветами;
- *Загрузка последней удачной конфигурации* — режим, восстанавливающий последнюю неиспорченную копию реестра ОС Windows XP.

3.2.3. Файловые системы Windows XP

Все операционные системы, как современные, так и давно уже неиспользуемые, имеют одну общую черту — хранение информации в ОС осуществляется подсистемой, называемой *файловой системой.*

▶ *Файловая система* — это набор спецификаций и соответствующее им программное обеспечение, которое отвечает за создание, удаление, организацию, чтение, запись, модификацию и перемещение файлов информации, а также за управление доступом к файлам и за управление ресурсами, которые используются файлами. Файловая система определяет способ организации данных на диске (или на другом носителе).

Информация на магнитных дисках размещается и передается блоками. Каждый блок называется *сектором* и располагается на концентрических дорожках поверхности диска. Группа дорожек одного радиуса, расположенных на поверхностях магнитных дисков, образуют *цилиндры.* Каждый сектор состоит из *поля данных* и *поля служебной информации,* ограничивающей и идентифицирующей его. Размер сектора (объем поля данных) устанавливается контроллером или драйвером. Физический адрес сектора на диске определяется с помощью трех «координат»:

1) номер цилиндра;

2) номер рабочей поверхности диска;

3) номер сектора на дорожке.

Обмен информацией между ОЗУ и дисками физически осуществляется только секторами.

Диск может быть разбит на несколько *разделов,* которые могут использоваться как одной ОС, так и несколькими. На каждом разделе может быть организована своя файловая система. Для организации хотя бы одной файловой системы должен быть определен, по крайней мере, один раздел.

Разделы могут быть двух типов:

1) *первичный*;

2) *расширенный.*

Максимальное число первичных разделов — четыре, но обязательно должен быть хотя бы один. Если первичных разделов больше одного, то один должен быть *активным,* в нем находится загрузчик ОС.

На одном диске может быть только один расширенный раздел, который в свою очередь может содержать большое количество подразделов — *логических дисков.*

ОС Windows XP поддерживает работу со следующими файловыми системами: *FAT*16, *FAT*32, *NTFS.*

Аббревиатура *FAT* (*File Allocation Table*) означает «таблица размещения файлов». Этот термин относится к линейной табличной структуре со сведениями о файлах — именами файлов, их атрибутами и другими данными, определяющими местоположение файлов или их фрагментов в среде *FAT*. Элемент *FAT* определяет фактическую область диска, в котором хранится начало физического файла.

В файловой системе *FAT* логическое дисковое пространство любого логического диска делится на две области:

1) системную область;

2) область данных.

Системная область создается при форматировании и обновляется при манипулировании файловой структурой. Область данных содержит файлы и каталоги, подчиненные корневому, и доступна через пользовательский интерфейс. Системная область состоит из следующих компонентов:

- загрузочной записи;
- зарезервированных секторов;
- таблицы размещения файлов (*FAT*);
- корневого каталога.

Таблица размещения файлов представляет собой карту (образ) области данных, в которой описывается состояние каждого участка области данных. Область данных разбивается на кластеры. Один или несколько смежных секторов в логическом дисковом адресном пространстве (только в области данных) объединяются в единый дисковый блок — *кластер*.

▶ *Кластер* — минимальная адресуемая единица дисковой памяти, выделяемая файлу или некорневому каталогу. Например, в *FAT16* размер кластера составляет 32 кбайт. Файл или каталог занимает целое число кластеров. Последний кластер при этом может быть задействован не полностью, что приведет к заметной потере дискового пространства при большом размере кластера.

В таблице *FAT* кластеры, принадлежащие одному файлу (некорневому каталогу), связываются в цепочки. Для указания номера кластера в системе управления файлами *FAT*16 используется 16-битовое слово, следовательно, можно хранить информацию максимум о 65536 кластерах. Так как *FAT* используется при доступе к диску очень интенсивно, она загружается в оперативную память и находится там максимально долго.

Корневой каталог отличается от обычного каталога тем, что он размещается в фиксированном месте логического диска и имеет фиксированное число элементов. Структура системы файлов является иерархической. Файлам присваиваются первые доступные адреса кластеров в

томе. Номер начального кластера файла представляет собой адрес первого кластера, занятого файлом, в таблице размещения файлов. Каждый кластер содержит указатель на следующий кластер, использованный файлом, или индикатор (*0xFFFF*), указывающий, что данный кластер является последним кластером файла.

Файлы на дисках имеют 4 атрибута, которые могут сбрасываться и устанавливаться пользователем: *Archive* (архивный), *System* (системный), *Hidden* (скрытый) и *Read-only* (только чтение).

32-разрядная файловая система *FAT*32 обеспечивает оптимальный доступ к жестким дискам, *CD-ROM* и сетевым ресурсам, повышая скорость и производительность всех операций ввода/вывода. *FAT*32 представляет собой усовершенствованную версию *FAT*, предназначенную для использования на томах, объем которых превышает 2 Гбайта. Размер кластера в *FAT*32 равен 4 кбайт. *FAT*32 является полностью независимой 32-разрядной файловой системой и содержит многочисленные усовершенствования и дополнения по сравнению с *FAT*16. Принципиальное отличие *FAT*32 заключается в более эффективном использовании дискового пространства: *FAT*32 использует кластеры меньшего размера, что приводит к экономии дискового пространства. *FAT*32 может перемещать корневой каталог и использовать резервную копию *FAT* вместо стандартной. Расширенная загрузочная запись *FAT*32 позволяет создавать копии критических структур данных, что повышает устойчивость дисков к нарушениям структуры *FAT* по сравнению с предыдущими версиями. Корневой каталог представляет собой обычную цепочку кластеров, поэтому может находиться в произвольном месте диска, что снимает ограничение на размер корневого каталога.

Файловая система *NTFS* (*New Technology File System*) содержит ряд значительных усовершенствований и изменений, существенно отличающих ее от других файловых систем. *NTFS* обладает характеристиками защищенности, поддерживая контроль доступа к данным и привилегии владельца, играющие исключительно важную роль в обеспечении целостности жизненно важных конфиденциальных данных. Папки и файлы *NTFS* могут иметь назначенные им права доступа вне зависимости от того, являются они общими или нет. Если файл будет скопирован из раздела или тома *NTFS* в раздел или на том *FAT*, все права доступа и другие уникальные атрибуты, присущие *NTFS*, будут утрачены.

Форматирование тома для *NTFS* приводит к созданию нескольких системных файлов и главной таблицы файлов (*Master File Table, MFT*). *MFT* содержит информацию обо всех файлах и папках, имеющихся на томе *NTFS*. *NTFS* — это объектно-ориентированная файловая система,

которая обрабатывает все файлы как объекты с атрибутами. Практически все объекты, существующие на томе, представляют собой файлы, а все что имеется в файле, представляет собой атрибуты, включая атрибуты данных, атрибуты системы безопасности, атрибуты имени файла. Каждый занятый сектор на томе *NTFS* принадлежит какому-нибудь файлу. Частью файла являются даже метаданные файловой системы (информация, которая представляет собой описание самой файловой системы).

3.3. ОС семейства UNIX

Распространенность ОС семейства *Unix* возрастает по нескольким причинам. Во-первых, пользователи все чаще обращают внимание на стоимость и лицензированность применяемого программного обеспечения. В этом отношении указанное семейство ОС обладает несомненным преимуществом. Во-вторых, ведущие предприятия и транспортные компании начинают осознавать важность знания принципов функционирования программного обеспечения, подтверждаемого наличием исходных кодов. В третьих, для преподавателей информатики отнюдь не лишней будет возможность международного обмена методическими разработками с зарубежными коллегами, успешно применяющими ОС *Unix* в курсах дисциплин «*Computer science*».

3.3.1. Общие сведения

UNIX — это ядро многопользовательской операционной системы с разделением времени. Она дает пользователям возможность запускать свои программы, управляет периферийными устройствами и обеспечивает работу файловой системы.

Работу ОС UNIX можно представить в виде функционирования множества взаимосвязанных процессов. При загрузке системы сначала запускается ядро (процесс 0), которое в свою очередь запускает командный интерпретатор *shell* (процесс 1).

Взаимодействие пользователя с системой UNIX происходит в интерактивном режиме посредством командного языка. Оболочка операционной системы — *shell* — интерпретирует вводимые команды, запускает соответствующие программы (процессы), формирует и выводит ответные сообщения.

Важной составной частью UNIX является файловая система. Она имеет иерархическую структуру, образующую дерево каталогов и фай-

лов. Корневой каталог обозначается символом «/», путь по дереву каталогов состоит из имен каталогов, разделенных символом «/», например: */usr/include/sys*. В каждый момент времени с любым пользователем связан текущий каталог, то есть местоположение пользователя в иерархической файловой системе.

Каждый файл ОС UNIX может быть однозначно определен некоторой структурой данных, называемой описателем файла (дескриптором). Он содержит всю информацию о файле: тип файла, режим доступа, идентификатор владельца, размер, адрес файла, даты последнего доступа и последней модификации, дату создания и пр.

Обращение к файлу происходит по имени. Путь к файлу от корневого каталога называется полным именем файла. Если обращение к файлу начинается с символа «/», то считается, что указано полное имя файла и его поиск начинается с корневого каталога, в любом другом случае поиск файла начинается с текущего каталога.

У любого файла может быть несколько имен. Фактически имя файла является ссылкой на файл, специфицированный номером описателя.

Работа пользователя в системе начинается с того, что активизируется сервер терминального доступа *getty* (программа, организующая диалог работы с пользователем в многопользовательской ОС), который запускает программу *login*, запрашивающую у пользователя имя и пароль.

Далее происходит проверка аутентичности пользователя (подлинности регистрации пользователя в системе) в соответствии с той информацией, которая хранится в файле */etc/passwd*. В этом файле хранятся записи, содержащие

- регистрационное имя пользователя;
- зашифрованный пароль;
- идентификатор пользователя;
- идентификатор группы;
- информация о минимальном сроке действия пароля;
- общая информация о пользователе;
- начальный каталог пользователя;
- регистрационный *shell* пользователя.

Если пользователь зарегистрирован в системе и ввел правильный пароль, *login* запускает программу, указанную в */etc/passwd* — регистрационный *shell* пользователя.

3.3.2. Файловая система Unix

Пользователь системы — это объект, обладающий определенными правами, определяющими возможность запуска программ на выполнение, а также владение файлами. Единственный пользователь системы, обладающий неограниченными правами — это *суперпользователь* или *администратор системы*.

Система идентифицирует пользователей по *идентификатору пользователя* (*UID — User Identifier*). Каждый пользователь является членом одной или нескольких *групп* — списка пользователей, имеющих сходные задачи. Каждая группа имеет свой уникальный *идентификатор группы* (*GID — Group Identifier*) Принадлежность группе определяет совокупность прав, которыми обладают члены данной группы.

Права пользователя UNIX — это, прежде всего, права на работу с файлами. Файлы имеют двух владельцев — пользователя (*user owner*) и группу (*group owner*). Соответственно, атрибуты защиты файлов определяют права пользователя-владельца файла (*u*), права члена группы-владельца (*g*) и права всех остальных (*o*). В каждый момент времени с любым пользователем связан текущий каталог, то есть местоположение пользователя в иерархической файловой системе.

Всякий файл ОС UNIX в соответствии с его типом может быть отнесен к одной из следующих групп: *обычные файлы, каталоги, специальные файлы* и *каналы*.

Обычный файл представляет собой последовательность байтов. Никаких ограничений на файл системой не накладывается, и никакого смысла не приписывается его содержимому: смысл байтов зависит исключительно от программ, обрабатывающих файл.

Каталог — это файл особого типа, отличающийся от обычного файла наличием структуры и ограничением по записи: осуществить запись в каталог может только ядро ОС UNIX. Каталог устанавливает соответствие между файлами (точнее, номерами описателей) и их локальными именами.

Специальный файл — это файл, поставленный в соответствие некоторому внешнему устройству и имеющий специальную структуру. Его нельзя использовать для хранения данных как обычный файл или каталог, но над ним можно производить те же операции, что и над любым другим. При этом ввод/вывод информации в этот файл будет соответствовать вводу с внешнего устройства или выводу на него.

Канал — это программное средство, связывающее процессы ОС UNIX буфером ввода/вывода. Например, запуск процессов в виде *$ процесс_1 | процесс_2* означает, что стандартный вывод процесса_1

будет замкнут на стандартный ввод процесса_2. При этом сначала создается канал, а потом на выполнение одновременно запускаются оба процесса, и общее время их выполнения определяется более медленным процессом.

3.3.3. Командный язык системы UNIX

Набор имени команды производится с клавиатуры после появления приглашения ввода команд в *shell*-интерпретаторе, обычно, — $.

Для облегчения работы с системой UNIX имеется возможность использовать шаблоны имен файлов (или *метасимволов*):

? — один любой символ;

* — произвольное количество любых символов.

Например:

*.c — задает все файлы с расширением «c»;

pr???.* — задает файлы, имена которых начинаются с «pr», содержат пять символов и имеют любое расширение.

3.3.3.1. Справочные команды

date — получение даты и времени.

who — получение списка пользователей, работающих в системе в данный момент. Выводится имя пользователя, номер терминала, дата и время начала работы этого пользователя. Команда *who am i* выведет информацию о самом пользователе.

man — получение справочной информации:

$ man [*имя команды*]

Альтернативой команды **man** в некоторых клонах UNIX является команда **use**:

use [*имя команды*]

3.3.3.2. Команды работы с каталогами

pwd — печать имени текущего каталога. Например:

$ pwd

ls — вывод на экран содержимого каталога:

$ ls [*-ключи*] [*имя каталога*]

Если имя каталога не указано, выводится содержимое текущего каталога. Ключи определяют формат выдачи, например:

-l — вывод полной информации о каждом файле;

-a — вывод полного списка файлов, включая «.» и «..»;
-t — сортировка списка по времени создания;
-C — вывод списка в несколько колонок по алфавиту и т.п.

Пример:

$ ls –l

cd — смена директории (каталога):

$ cd [*полное_имя_каталога*]

При этом указанный каталог станет текущим. Команда **cd** без аргументов восстановит в качестве текущего каталога начальный каталог пользователя.

mkdir — создание нового каталога:

$ **mkdir** [*-ключи*] *имя_нового_каталога*

Для создания нового каталога пользователь должен иметь право записи в родительский каталог текущего каталога.

rmdir — удаление директории:

$ rmdir *список_каталогов*

Система не позволит удалить каталог, если он не пуст или если у пользователя нет прав записи в него. Текущий каталог не должен принадлежать поддереву удаляемых каталогов.

3.3.3.3. Команды работы с файлами

rm — удаление файлов (ссылок на файл):

$ rm [*-ключи*] *список_файлов*

Эта команда удаляет ссылки на файлы (то есть локальные имена файлов), если у пользователя есть право записи в каталог, содержащий эти имена. Если удаляемый файл защищен от записи, команда запрашивает подтверждение на удаление файла.

Ключи:

-i — вводит необходимость подтверждения для каждого удаляемого файла;

-f — отменяет необходимость подтверждения для любого удаляемого файла;

-r — задает режим рекурсивного удаления всех файлов и подкаталогов данного каталога, а затем и самого каталога.

chmod — изменение атрибутов защиты файла:

$ chmod *атрибуты список_файлов*

Атрибуты файла могут быть заданы разными способами:

1) буквенной кодировкой.

Атрибуты защиты обозначаются так:

r — доступ по чтению;
w — доступ по записи;
x — доступ по исполнению.

Категории пользователей задаются следующим образом:

u — атрибуты для владельца файла;
g — атрибуты для группы владельца файла;
o — атрибуты для прочих пользователей;
a — атрибуты для всех категорий пользователей.

Производимая операция кодируется с помощью таких символов:

= — установить значения всех атрибутов для данной категории пользователей;

+ — добавить атрибут для данной категории пользователей;

- — исключить атрибут для данной категории пользователей.

Пример. Разрешить доступ по чтению и записи к файлам с именем *student* владельцу и группе-владельцу:

$ chmod ug+rw *student*.*

2) в виде восьмеричного числа. Числовые значения атрибутов защиты кодируются трехразрядным восьмеричным числом, где существование соответствующего атрибута соответствует наличию единицы в двоичном эквиваленте восьмеричной цифры этого числа, а отсутствие атрибута — нулю. Например:

символьное представление	rwx	r-x	r--
двоичное представление	111	101	100
восьмеричное представление	7	5	4

Пример. Установить атрибуты чтения и записи для владельца и группы-владельца и только чтения для остальных пользователей:

$ chmod 0664 *ab??.doc*

cat — слияние и вывод файлов на стандартное устройство вывода:

$ cat [-*ключи*] [*входной_файл1*[*входной_файл2*...]]

Команда по очереди читает указанные входные файлы, если их несколько, объединяет и выводит считанные данные в стандартный поток вывода (на экран). С помощью перенаправления потоков (программных каналов) команда cat может быть использована для выполнения разнообразных операций.

Примеры:

1. $ cat > *file1*

– в файл *file1* помещается текст, набираемый на клавиатуре.

Если до этого файл *file*1 не существовал, он будет создан; если существовал, его первоначальное содержимое будет утрачено. Окончание ввода текста происходит при нажатии комбинации клавиш Ctrl+D.

2. $ cat *file*1 > *file*2

– содержимое файла *file*1 копируется в файл *file*2. Файл *file*1 при этом остается без изменений.

3. $ cat *file*1 *file*2 > *result*

– содержимое *file*2 будет добавлено к содержимому *file*1 и помещено в файл *result*.

4. $ cat *file*1 >> *file*2

– содержимое файла *file*1 добавляется в конец файла *file*2.

cp — копирование файлов:

$ cp *вх_файл_1* [*вх_файл_2* [...*вх_файл_n*]] *вых_файл*

Эта команда имеет два режима использования:

1. Если выходной файл есть обычный файл, то входной файл может быть только один; его содержимое копируется в выходной файл. Если выходной файл существовал, то его старое содержимое утрачивается, а атрибуты защиты остаются; если выходной файл не существовал, то он будет создан и унаследует атрибуты входного файла.

2. Если выходной файл есть каталог, то в него скопируются все указанные входные файлы, но каталог, естественно, должен быть создан заранее.

Пример. Скопировать два файла из текущего каталога в указанный с теми же именами:

$ cp *f1.txt f2.txt ../usr/text*

$ ls *../usr/text*

f1.txt

f2.txt

mv — пересылка файлов:

$ mv *вх_файл_1* [*вх_файл_2 [...вх_файл_n*]] *вых_файл*

Отличие команды пересылки от команды копирования состоит только в том, что входные файлы после выполнения команды уничтожаются.

Пример. Перенести файлы с расширением «*c*» из указанного каталога в текущий:

$ mv *text/*.c* .

ln — создание новых ссылок на файл:

$ ln *вх_файл_1* [*вх_файл_2* [...*вх_файл_n*]] *вых_файл*

Эта команда имеет два режима использования:

1. Если выходной файл является обычным файлом, то входной файл может быть только один; в этом случае на него создается ссылка с именем вых_файл и к нему можно обращаться и по имени вх_файл_1, и по имени вых_файл. Количество ссылок на файл в описателе увеличивается на 1.

2. Если выходной файл есть каталог, то в нем создаются элементы, включающие имена всех перечисленных входных файлов и ссылки на них.

Пример:

$ ls

*file*1

$ ln *file*1 *file*2

$ ls

*file*1

*file*2.

3.3.3.4. Команды работы с текстовыми файлами

grep — поиск шаблона (подстроки) в файлах:

$ grep [–*ключи*] *подстрока* *список_файлов*

Найденные строки выводятся на стандартный вывод в формате, определяемом ключами. Если файлов несколько, то перед каждой строкой выводится имя соответствующего файла. Ключи:

-c — вывод имен всех файлов с указанием количества строк, содержащих шаблон;

-i — игнорирование регистра (различия строчных и заглавных латинских букв);

-n — вывод перед строкой ее относительного номера в файле;

-v — вывод строк, не содержащих шаблона (инверсия вывода);

-l — вывод только имен файлов, содержащих шаблон.

wc — подсчет количества строк, слов и символов в файлах:

$ wc [-*ключи*] [*список_файлов*]

Подсчет строк — ключ –l, слов — ключ –w и символов — ключ –c (по умолчанию –lwc). Если список файлов пуст, то подсчет ведется в стандартном потоке ввода.

sort — сортировка файлов:

$ sort [-*ключи*] *список_файлов*

Эта команда сортирует входные файлы по строкам в соответствии с увеличением кодов символов. Ключи:

-r — обратный порядок сортировки;

-f — не учитывать различие строчных и прописных латинских букв

-n — числовой порядок сортировки и т.д.

cmp — вывод места первого расхождения:

$ cmp *файл_1 файл_2*

Выводит номер символа и номер строки (в текстовых файлах), в которой впервые встречается расхождение во входных файлах. Работает с любыми файлами.

diff — вывод всех расхождений в файлах:

$ diff *файл_1 файл_2*

Выводит все строки, в которых встречаются расхождения между входными файлами. Работает только с текстовыми файлами.

find — поиск файлов в поддереве каталогов:

find список_каталогов условия_поиска

Команда последовательно просматривает все поддеревья, начинающиеся с одного из каталогов, указанных в списке каталогов, анализирует их атрибуты, и если они удовлетворяют условиям поиска: выполняет действия, заданные в *условиях_поиска.*

В команде может быть задано множество условий поиска, необходимые комбинации которых объединяются в булевское выражение с помощью логических операций:

! *условие*	отрицание условия;
пробел	соответствует операции «И»;
-o	операция «ИЛИ»;
\(*выражение* \)	булевское выражение в скобках

3.3.3.5. Команды работы с процессами

& — запуск процесса как фонового (параллельного):

$ *имя_процесса* [*-ключи*] [*параметры*] &

При выполнении этой команды следующее приглашение ОС появится сразу же после запуска процесса (не дожидаясь его завершения). Фоновый процесс не допускает ввода с клавиатуры и выводит сообщения на экран, нарушая целостность ввода и вывода привилегированного процесса.

ps — получить список всех процессов:

$ ps [*-ключи*]

При отсутствии ключей будет выведен список процессов самого пользователя (идентификатор процесса, номер терминала и время процессора, затраченное на процесс). Ключи:

-e — вывод информации обо всех процессах в системе;

-a — вывод информации о процессах, связанных с данным терминалом;

-l — вывод информации в длинном формате.

ГЛАВА 4

ОФИСНЫЕ ПРИЛОЖЕНИЯ ОПЕРАЦИОННЫХ СИСТЕМ

4.1. Введение в MICROSOFT OFFICE

Microsoft Office для Windows — популярнейший (начиная с Microsoft Office 95) комплекс прикладных программ, который включает:

■ **Microsoft Word** — редактор текстов. Word позволяет объединить в одном документе собственно текст, диаграммы, рисунки и таблицы. Наличие гиперссылок позволяет использовать его для создания электронных документов. В Word включена поддержка файловых форматов многих распространенных текстовых редакторов.

■ **Microsoft Excel** — редактор электронных таблиц. Документ Excel, иначе называемый *листом,* состоит из организованных в столбцы и строки ячеек. Его основное назначение — расчет интересующих пользователя параметров и анализ полученных результатов за счет установки зависимостей (подчас весьма сложных) между данными, хранящимися в различных ячейках. Весьма существенно, что наряду с возможностью создания собственных формул, пользователь имеет возможность задействовать встроенный мастер функций, которые для удобства распределены по категориям.

■ **Microsoft Access** — система управления реляционными базами данных, предназначенная для хранения и обработки больших объемов информации.

■ **PowerPoint** — программа презентационной графики, ориентированная на создание слайдов и материалов для выдачи слушателям. Предлагает пользователю набор стандартных слайдов с профессионально выполненным дизайном. Предоставляет весьма обширный набор анимационных и звуковых эффектов для повышения внимания аудитории к ключевым моментам доклада (мультимедиа-демонстрации).

■ **Microsoft Outlook** — содержит подсистему для работы с электронной почтой; планировщики расписаний на день, на неделю и на месяц; базу данных деловых контактов, в которой можно вести списки имен, адресов и телефонов; список задач с возможностью назначения приоритетов и управления временем; подсистему управления документами.

Во всех приложениях Office используются стандартные команды, окна диалога и основные операции, что существенно облегчает изучение этих приложений. Приложения Office изначально проектировались для совместной работы, что существенно облегчает обмен данными между ними.

Панель Office

С помощью панели можно запускать приложения Office, открывать заранее созданные шаблоны Office, настраивать программы Office или запускать любые другие приложения и утилиты системы (например, Проводник Windows). Панель Office (рис. 4.1) представляет собой набор панелей инструментов с настраиваемыми кнопками, расположенных вдоль верхней, нижней или боковой стороны экрана. Кроме того, панель Office может находиться на рабочем столе в виде «плавающего» окна. Панель Office во многом аналогична панелям инструментов в отдельных приложениях Office — чтобы запустить программу или открыть нужную папку, достаточно нажать соответствующую кнопку на панели.

Рис. 4.1

Работа с приложениями Microsoft Office

Запустить приложения Microsoft Office можно различными способами:

- нажать кнопку **Пуск** и выбрать имя программы в папке **Программы**;
- щелкнуть значок программы на панели Office;
- открыть шаблон документа;
- сделать двойной щелчок на имени файла, содержащего документ Office в **Проводнике** Windows или Microsoft Outlook.

Создание файлов на базе шаблонов

Если необходимо создать конкретный деловой документ, но при этом неясно, какое именно приложение Office следует для этого использовать, можно выполнить в меню **Пуск** команду **Создать документ** Microsoft Office (New Office Document) или же нажать кнопку **Создать документ** (Start A New Document) на панели Office. Это позволит просмотреть

разнообразные типы разработанных заранее документов (шаблонов) и открыть именно тот, который нужен. Шаблоны позволяют сосредоточить внимание на информации, которую необходимо представить в документе, и избавиться от длительного процесса разработки внешнего вида и форматирования. Каждое приложение Office содержит несколько шаблонов документов, которые можно открыть непосредственно из окна диалога **Создание документа** (New).

Запуск приложения Office с использованием шаблона документа можно производить одним из следующих способов.

1. Нажать кнопку **Пуск** и выполнить команду **Создать документ Microsoft Office** (или нажать на панели Office кнопку **Создать документ**). Открывается окно диалога с несколькими вкладками, каждая из которых соответствует определенному типу шаблонов. В окне диалога также имеется область предварительного просмотра, где приводится уменьшенное изображение текущего выделенного шаблона.

2. Выбрать вкладку, соответствующую типу создаваемого документа. Например, для просмотра списка существующих шаблонов факсов следует выбрать вкладку **Письма и факсы** (рис. 4.2). Если нажать в окне диалога кнопку **Таблица** (Detalls), можно узнать размер файла шаблона, тип файла (включая связанное с ним приложение) и дату внесения последних изменений.

3. Сделать двойной щелчок на шаблоне, который нужно открыть. Windows запускает связанное с ним приложение и загружает шаблон. Когда выбранный шаблон откроется в виде нового документа без имени, необходимо занести в документ информацию и сохранить файл под другим именем, чтобы не испортить исходный шаблон (позднее он может снова понадобиться). Большинство шаблонов содержат базовые инструкции, облегчающие создание документа.

Использование Мастеров — специальных шаблонов

▶ *Мастером* называется специальный автоматизированный документ, способный руководить процессом пошагового создания делового документа.

Использование мастеров во многих случаях предпочтительнее простых шаблонов, поскольку мастера позволяют изменять стиль или формат документа во время его создания (шаблоны по опреде-лению всегда содержат один и тот же формат). Например, **Мастер факсов** позволяет указать в факсе настраиваемые поля для имени, адреса и номера телефона, а также украшает документ декоративными элементами. Полезные мастера содержатся в многочисленных вкладках окна диалога **Создание документа**.

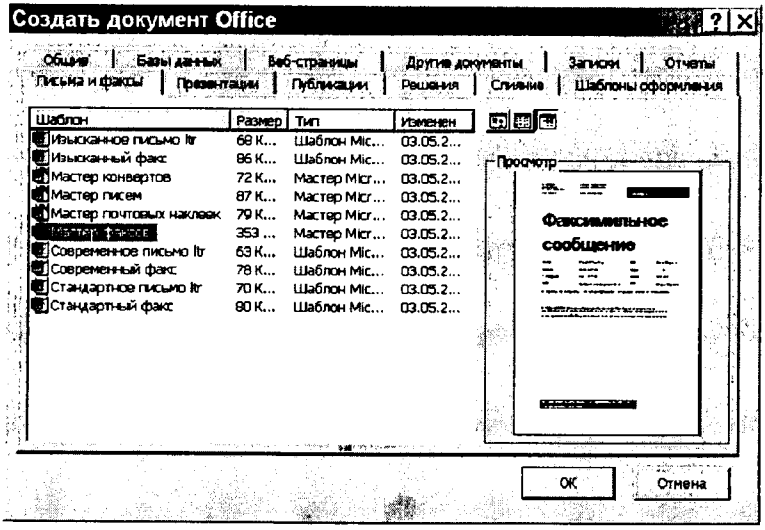


Рис. 4.2

Работа с панелью задач

Чтобы переключиться из активного приложения Windows на программу, выполняющуюся в фоновом режиме, необходимо нажать кнопку на панели задач, которая связана с нужной программой. Окно выделенного приложения выходит на передний план и (обычно) закрывает собой окна всех остальных приложений. Данное приложение становится новым активным приложением и воспринимает ввод информации пользователем. Если под управлением Windows работает более трех-четырех программ, можно расширить панель задач, чтобы отвести под кнопки больше места. Для этого необходимо переместить указатель мыши к верхнему краю панели задач. Когда он примет вид указателя изменения вертикальных размеров, следует перетащить верхний край так, чтобы высота панели задач удвоилась.

Выход из приложений Office

Закончив работу с приложением Office, пользователь должен закрыть программу перед выключением компьютера, поскольку могут быть утрачены изменения, внесенные с момента последнего выполнения команды **Сохранить**.

Способы закрытия приложений Office:

- выполнить команду **Выход** (Exit) в меню **Файл** (File) приложения;
- нажать кнопку **Закрыть** на заголовке окна приложения  в правом верхнем углу);
- нажать Alt+F4 (здесь и далее подобная запись обозначает одновременное нажатие соответствующих клавиш);
- щелкнуть на кнопке приложения на панели задач правой кнопкой мыши и выбрать в контекстном меню команду **Закрыть**.

Если в документе имеются несохраненные изменения, появляется окно диалога с предложением сохранить их. Если нажать кнопку **Да**, то изменения будут сохранены в файле с текущим именем (если имени файла еще не существует, пользователь сможет ввести его). Если же нажать кнопку **Нет**, то все изменения, сделанные с момента последнего сохранения, пропадут. Наконец, нажатие кнопки **Отмена** (Cancel) закрывает окно диалога и возвращает пользователя к работе с приложением.

4.2. Основы работы с Office: окна и панели инструментов

Во всех программах комплекса Microsoft Office применен один и тот же пользовательский интерфейс. Отображаемая в каждом приложении информация представлена в одном или нескольких стандартных окнах, которые можно прокручивать, масшта-бировать, а в некоторых случаях разделять на два вида одного документа. Кроме того, каждое приложение Office содержит стандартные команды для вывода на печать. Можно настроить принтер, осуществить предварительный просмотр документа перед печатью и создать окончательную распечатку.

Работа с окнами приложений

При запуске приложения Office окно программы может находиться в одном из трех состояний — свернутом (в виде кнопки на панели задач), развернутом (до размеров целого экрана) или нормальном (свободно перемещаемом по экрану).

При работе с нормальными окнами пользователь может изменять размер окна и его пропорции, а также перемещать его в пределах рабочего стола. Для изменения размеров окна необходимо установить указатель мыши над краем окна, которое планируется растянуть или сжать,

и перетащить этот край в новое положение (стандартный указатель мыши при этом заменяется указателем изменения размеров). Для перемещения окна перетащить его заголовок из одного положения в другое. Невозможно изменить размеры или положение окон, свернутых в кнопки на панели задач, а также окон, развернутых до размеров экрана.

Понятие рабочей области

Элементы пользовательского интерфейса, которые появляются при запуске приложения Office, — меню, панели инструментов, строки состояния и окна — образуют так называемую *рабочую область* программы. На рис. 4.3 изображена рабочая область Excel — типичного приложения Office.

Вдоль верхнего края окна приложения проходит *строка заголовка* (или просто *заголовок)* — прямоугольник с названием программы и управляющими кнопками для изменения размеров и закрытия окна. Если окно документа развернуто, в строку заголовка также включено название документа. Под заголовком находится *главное меню* с командами, которые выполняют всю основную работу в программе. Еще ниже располагается один или несколько наборов кнопок. Эти наборы называются *панелями инструментов.* Кнопки на панелях инструментов

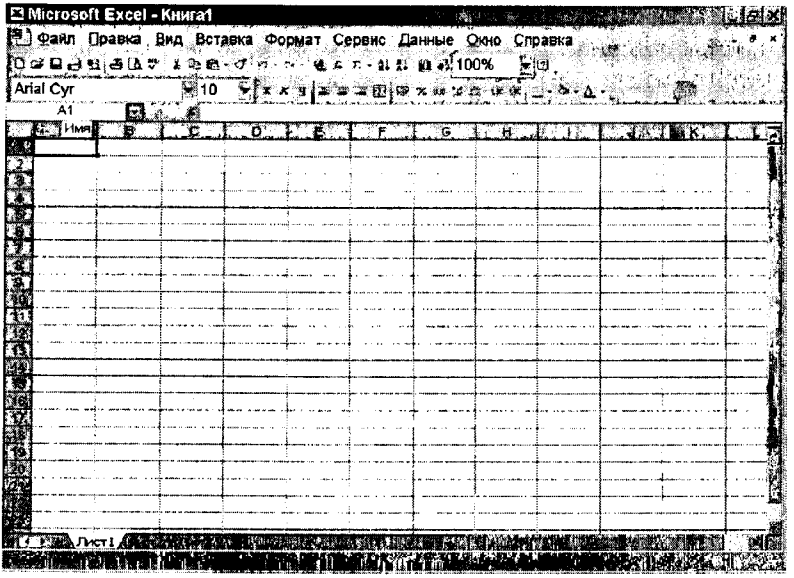


Рис. 4.3

обычно предоставляют ускоренный доступ к командам меню (отсюда распространенный термин «командные кнопки») — для выполнения команды достаточно просто нажать кнопку.

Под панелями инструментов находится область документов, уникальная для каждого приложения Office. В каждом приложении Office используется свой, отличающийся от других тип документа. Наконец, каждое приложение Office включает горизонтальные и вертикальные *полосы прокрутки* для перемещения в окне документа и *строку состояния* (у нижнего края окна) с информацией об используемых клавишах переключения режимов (Num Lock и Insert) и других специфических для данного приложения деталях.

Перечень стандартных элементов рабочей области приведен в табл. 4.1.

Таблица 4.1

Стандартные элементы рабочей области

Элемент рабочей области	Описание
Заголовок	Прямоугольная полоса в верхней части окна приложения, содержащая название приложения Office, имя документа (если окно развёрнуто) и управляющие кнопки
Управляющие кнопки	Кнопки Свернуть (Minimize), Развернуть (Maximize) или Восстановить (Restore) и Закрыть (Close) в окне приложения и в каждом окне документа
Главное меню	Область под заголовком, содержащая названия меню. Каждое меню открывает доступ к определённой группе команд приложений
Панели инструментов	Один или несколько рядов с раскрывающимися списками и командными кнопками, расположенных под главным меню
Полосы прокрутки	Горизонтальная и вертикальная полосы у нижнего и правого края окна, используемые для просмотра частей документа, не отображаемых в окне в настоящий момент
Строка состояния	Индикаторы клавиш переключения режимов Num Lock и Insert, а также любые данные, относящиеся к конкретному приложению (например, номер страницы или содержимое ячейки)

Работа с несколькими окнами документов

Word, Excel и PowerPoint позволяют открывать несколько окон документов одновременно, чтобы пользователь мог сравнить взаим-

освязанные документы, переслать информацию из файла в файл или работать над отчетом или презентацией, создаваемыми на основе нескольких документов. При работе с документами в отдельных окнах каждое открытое окно документа имеет свой собственный заголовок, управляющие кнопки и полосы прокрутки. Подобный вид экрана задается командой **Окно/Упорядочить все**.

Для выбора окна активного документа и управления его внешним видом Word, Excel, PowerPoint и Access содержат меню **Окно**. В нижней части меню перечислены имена открытых в приложении документов (активный документ отмечен маркером). Для перехода к другому открытому документу нужно выбрать его имя или ввести его номер при открытом меню **Окно**.

В верхней части меню **Окно** расположены команды, с помощью которых можно открывать новые окна и упорядочивать существующие. В приложениях Word, Excel и PowerPoint команда **Новое** открывает новое окно и помещает в него активный документ. Таким образом удается

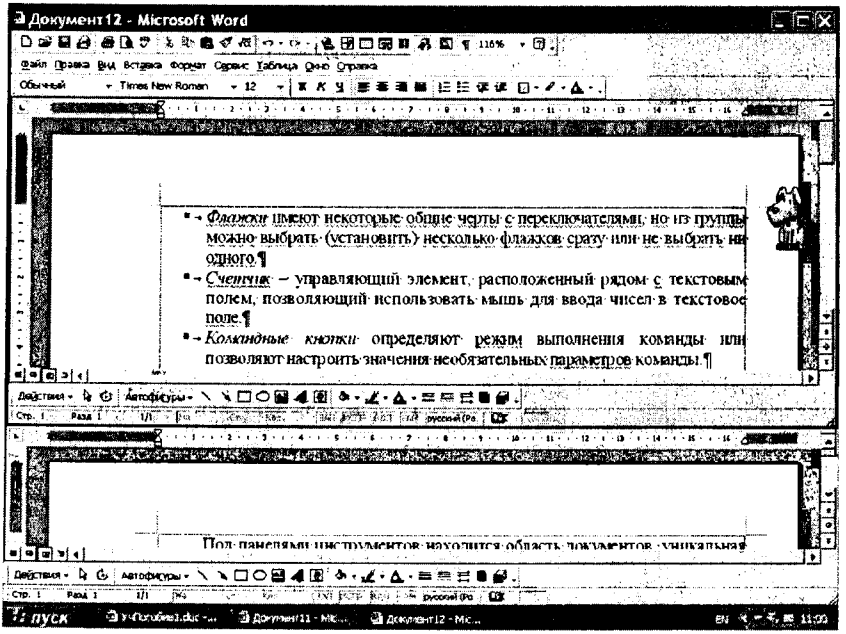


Рис. 4.4

отобразить содержимое одного документа в нескольких окнах. Они различаются между собой по номеру, следующему за именем документа.

В приложении Word под командой **Новое** в меню **Окно** расположена команда **Упорядочить** (Расположить) **все**, которая равномерно распределяет рабочую область между всеми открытыми окнами документов (результат применения команды **Расположить все** показан на рис. 4.4). Команда **Расположить все** оказывается полезной при сравнении всех открытых документов. Если после выполнения команды **Расположить все** часть документа окажется скрытой, ее можно отобразить с помощью полос прокрутки. Упорядочивание документов командой **Расположить все** часто называют *мозаикой,* поскольку окна раскладываются вплотную друг к другу, как мозаичные плитки. В приложении PowerPoint тоже имеется команда **Расположить все**, а в Excel и Access — аналогичные команды, позволяющие в большей степени управлять процессом упорядочивания окон.

Команда **Разделить**, присутствующая только в Word и Excel, просто разделяет экран на две части, показывая пользователю два окна активного документа. При выполнении этой команды в Word на экране появляется серая разделяющая линия, перемещая ее, можно точно указать, где требуется разделить окно.

Команда **Разделить** используется, когда необходимо редак-тировать две разные части документа в одном окне или читать инструкции из одной части документа при работе с другой. Например, бывает полезно читать инструкции, приведенные в верхней части шаблона, во время заполнения пустых мест, находящихся внизу. Важно, что команда **Разделить** следит за редактированием, так что изменения, внесенные в одном окне, автоматически отражаются в другом. Работа в режиме команды **Разделить** завершается командой **Окно/Снять разделение**.

Соглашения меню

Приложения Office содержат меню **Окно** с практически одинаковыми командами. Кроме того, во всех пяти приложениях Office имеются меню **Файл**, **Правка**, **Формат** и **?** (Help) со многими общими командами, поэтому освоение этих меню в одном приложении позволяет пользоваться ими во всех приложениях Office. В табл. 4.2 приведены наиболее важные соглашения, относящиеся к меню (об окнах диалога речь пойдет в следующем разделе).

Таблица 4.2

Наиболее важные соглашения, относящиеся к меню

Элемент меню	Значение	Пример
Серый цвет команды	Команда меню в настоящий момент недоступна	Вставить Shift + Ins
Многоточие	Команда меню вызывает на экран окно диалога	Границы и заливка ...
Маркер рядом с командой	Команда соответствует флажку, который в данный момент установлен. Повторное выполнение помеченной команды убирает маркер и снимает флажок	☑ Форматирование
Меню следующего уровня	Выбор команды вызывает появление следующего меню с новыми командами	Макрос▶
Сочетание клавиш	Дополнительная возможность вызова команды меню с помощью клавиатуры	Заменить Ctrl + H
Подчёркнутая буква	Нажатие на клавиатуре буквы, подчёркнутой в элементе меню, выполняет команду	Колонтитулы
Кнопка панели инструментов	Можно выполнить команду меню с помощью соответствующей кнопки на панели инструментов	Предварительный просмотр

Макросы — средства автоматизированного выполнения задач, разработанные на языке *VBA — Visual Basic for Application.*

Элементы окон диалога

На рис. 4.5 изображено типичное окно диалога приложения Office (оно появляется при выполнении команды **Файл/Печать** в приложении Word). Окна диалога позволяют выполнить некоторые действия с помощью списков, кнопок и других элементов окна. Чтобы заполнить окно диалога, пользователь просто вводит нужную информацию и затем нажимает кнопку OK. Чтобы перейти к следующему элементу окна диалога, достаточно щелкнуть по нему или нажимать клавишу **Tab** до тех пор, пока нужный элемент не окажется выделенным. Во многих окнах диалога не требуется заполнять все пустые поля. Если при виде окна диалога у пользователя вдруг возникают сомнения по поводу данной команды, можно отменить ее кнопкой **Отмена** или кнопкой **Закрыть** на заголовке окна диалога. Кроме того, можно вызвать справку — для

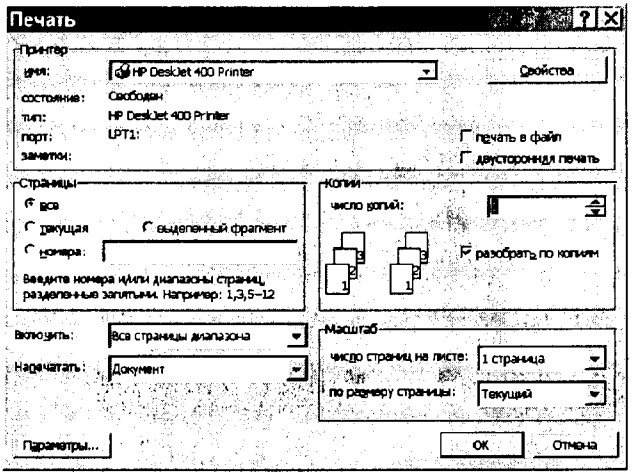


Рис. 4.5

этого следует сначала нажать кнопку **Справка** на заголовке окна диалога, а затем щелкнуть мышью на интересующем элементе.

На рис. 4.5 изображены некоторые элементы окон диалога вместе с краткими описаниями того, как с ними работать.

■ *Раскрывающиеся списки* позволяют выбрать значение из перечня, причем обычно имеется стандартное значение, рекомендуемое по умолчанию.

■ *Текстовые поля* позволяют вводить в окно диалога текст, числа или специальные символы. Формат данных текстовых полей отличается для разных окон диалога.

■ *Переключатели* предназначены для выбора одного значения из группы — в любой момент времени активной может быть лишь одна кнопка.

■ *Флажки* имеют некоторые общие черты с переключателями, но из группы можно выбрать (установить) несколько флажков сразу или не выбрать ни одного.

■ *Счетчик* — управляющий элемент, расположенный рядом с текстовым полем, позволяющий использовать мышь для ввода чисел в текстовое поле.

■ *Командные кнопки* определяют режим выполнения команды или позволяют настроить значения необязательных параметров команды.

Типичные функции правой кнопки мыши в приложениях Office

■ Чтобы изменить размер окна приложения Office или закрыть его, щелкнуть на заголовке окна правой кнопкой мыши, а затем выполнить нужную команду в открывшемся оконном меню.

■ Чтобы отредактировать или отформатировать выделенный текст в приложении, щелкнуть на тексте правой кнопкой мыши, а затем выбрать из меню нужную команду форматирования.

■ Чтобы добавить, убрать или настроить панель инструментов в приложении Office, щелкнуть на панели инструментов правой кнопкой мыши. В открывшемся меню выбрать имя панели инструментов, которую требуется вывести или убрать с экрана, или же выполнить команду **Настройка** (Customize) для изменения свойств панели.

■ Чтобы закрыть приложение прямо из панели задач, щелкнуть правой кнопкой мыши на кнопке приложения и выполнить команду **Закрыть**.

Работа с панелями инструментов

▶ *Панелью инструментов* называется настраиваемый набор кнопок и раскрыва ющихся списков. Они позволяют быстро вызывать наиболее часто используемые команды и операции приложения.

Кнопки на панелях инструментов Office обычно являются эквивалентами команд меню, но их назначение проще запомнить, потому что кнопки содержат *значки* — графические представления своих действий.

На рис. 4.6 изображены панели инструментов, которые по умолчанию выводятся в рабочей области Microsoft Excel. Стандартная панель инструментов, содержащая самые полезные команды Excel, в стандартной конфигурации находится непосредственно под главным меню. Под стандартной панелью инструментов находится панель форматирования — часто используемая панель инструментов с раскрывающимися списками и кнопками, которые предназначены для форматирования ячеек в листе.

Рис. 4.6

Изображенные кнопки присутствуют практически во всех приложениях Office, хотя могут располагаться на панели в несколько ином порядке. Для того чтобы вспомнить назначение той или иной кнопки на панели инструментов, нужно ненадолго задержать над ней указатель мыши — на экране появится название кнопки. Данное справочное средство называется «подсказкой». Кроме того, можно выполнить команду **?/Что это такое?** и затем щелкнуть по интересующей кнопке.

Перемещение панелей инструментов

Если для данных в рабочей области приложения необходимо освободить как можно больше места, одну или несколько панелей инструментов можно перенести в другое место экрана. Большинство панелей инструментов Office может находиться в верхней (стандартное положение) или нижней части окна приложения, а также присутствовать в виде «плавающей палитры» в любом месте экрана. Чтобы перенести панель инструментов, нужно щелкнуть мышью на пустом месте панели (лучше всего — у левого края) и перетащить панель в новое положение. Во время перемещения вместе с указателем мыши по экрану движется прямоугольный контур панели, облегчающий выбор конечного положения. Если панель инструментов окажется расположенной поверх другой панели, то произойдет *стыковка* панелей, при которой большая часть кнопок окажется скрытой (об этом состоянии свиде-тельствуют две маленькие стрелки » в конце панели).

На рис. 4.7 показана рабочая область Word после перемещения панели форматирования в нижнюю часть окна, а стандартной панели — в виде «плавающей» панели в правый верхний угол листа. «Плавающие» панели всегда располагаются поверх документа. После того как панель инструментов сделана «плавающей», ее можно перемещать (перетаскиванием заголовка) или изменять ее размеры с помощью мыши.

Добавление и удаление панелей инструментов

Когда фирма Microsoft разрабатывала приложения Office, она провела ряд тестов, призванных определить, какие команды и операции чаще всего выполняются пользователями Office. По результатам тестов для каждого приложения Office был создан набор панелей инструментов, предоставляющих доступ к командам и операциям, которые, по мнению пользователей, являются самыми полезными при выполнении определенной задачи. Самые необходимые кнопки были размещены на стандартной панели инструментов; кнопки, связанные с процессом

форматирования документов, — на панели форматирования и т. д. Были созданы панели инструментов, которые появляются только в конкретном приложении Office и позволяют оптимальным образом использовать его возможности.

При первом запуске приложения Office в верхней части окна находится одна или две панели инструментов. Например, при первом запуске Excel или Word отображаются две панели инструментов — стандартная и панель форматирования. Чтобы ознакомиться со списком панелей инструментов, поддерживаемых приложением, нужно щелкнуть на любой из панелей правой кнопкой мыши (место щелчка значения не имеет). В результате появляется меню, в котором перечислены все доступные в приложении панели инструментов. Его возможный вид приведен на рис. 4.8.

Активные панели (то есть те, которые отображаются на экране в данный момент) в этом списке помечены маркером. Под названиями панелей имеется специальная команда **Настройка**, позволяющая менять оформление и содержание представленных в списке панелей. Команда **Настройка** также применяется для отображения новых панелей инструментов и для восстановления стандартной конфигурации измененных панелей.

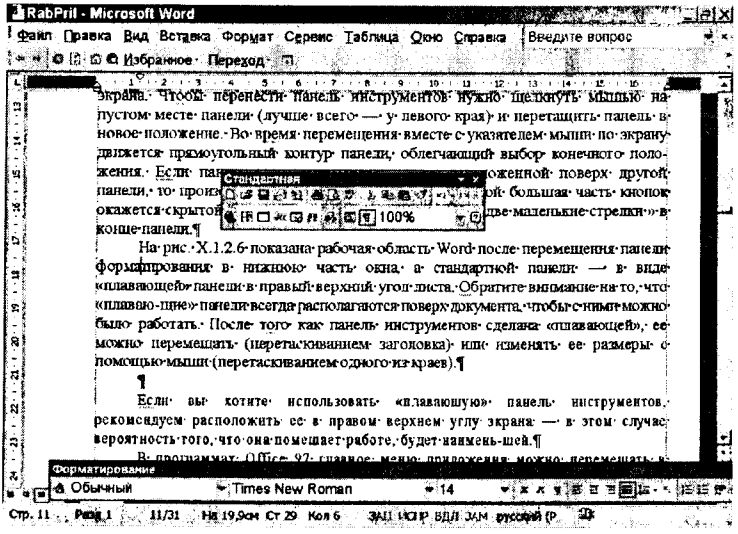


Рис. 4.7

Стандартная
Форматирование
Visual Basic
Web
WordArt
Автотекст
Базы данных
Настройка изображения
Рецензирование
Рисование
Таблицы и границы
Формы
Элементы управления
Панель 1
Настройка

Рис. 4.8

Открытие существующего документа

Имеются четыре способа открыть существующий документ Office. Если связанное с документом приложение Office не запущено в настоящий момент, самый быстрый способ — выполнить команду **Открыть документ** Microsoft Office в меню **Пуск**. Можно также нажать кнопку **Открыть документ** на панели Office. Если же вы работаете с приложением Office, следует выполнить команду **Файл/Открыть** или нажать кнопку **Открыть** на стандартной панели инструментов. На рис. 4.9 изображено окно диалога, которое появляется при выполнении команды **Открыть** в Microsoft Word (аналогичные окна появляются при выполнении команды **Открыть** в любом приложении Office).

В окне **Открытие документа** отображаются файлы и папки, находящиеся в той папке, которая последней использовалась в вашем приложении (ее часто называют *текущей* папкой, поскольку по умолчанию Office помещает созданные файлы именно в нее). Название

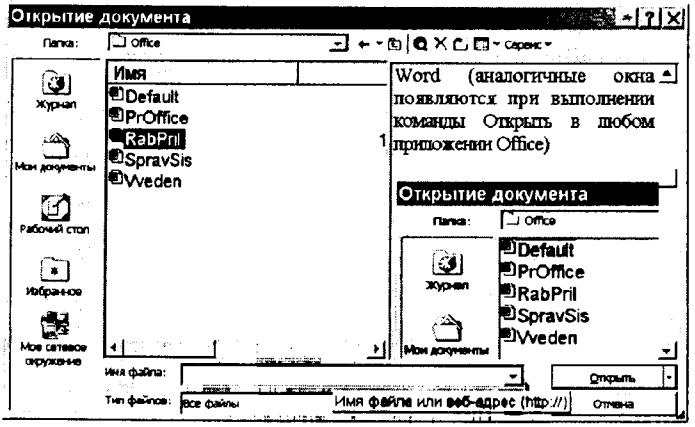


Рис. 4.9

текущей папки приведено в поле **Папка** в верхней части окна диалога, а расположенные в папке файлы и папки перечислены ниже в списке. Чтобы открыть файл документа из списка, нужно сделать двойной щелчок мышью на имени файла, и он появится в приложении Office готовым к работе.

Поскольку жесткий диск типичного компьютера содержит десятки папок и тысячи файлов, окно диалога **Открытие документа** содержит ряд средств для поиска и открытия конкретных документов. Например, с помощью раскрывающегося списка **Тип файла** можно управлять типом файлов, отображаемых в окне диалога. Значение **Все файлы** позволяет просмотреть все содержимое папки.

Помимо команды **Открыть** существует и другой способ открытия нескольких последних файлов — щелчок на их именах, перечисленных в нижней части меню Файл. Этот способ быстрее команды **Открыть**, поскольку не приходится искать файл в окне диалога. На рис. 4.10 показаны имена четырех последних открывавшихся файлов Word из меню Файл. На первом месте стоит последний из документов, который открывался в активном приложении, на втором — предпоследний и т. д. Каждый раз, когда открывается новый файл, вершина списка пополняется новым именем, а последнее имя пропадает. Имя файла включает *путь* к файлу, если только файл не находится в текущей папке. Путь состоит из всех папок, которые необходимо открыть, чтобы добраться до документа. Знание пути к файлу помогает найти файл.

Кроме того, для открытия недавно редактировавшегося файла можно нажать кнопку **Пуск** на панели задач, щелкнуть на строке **Документы** и затем выбрать имя нужного файла. В меню **Документы** приведен список всех файлов, которые недавно использовались в приложениях Microsoft Office, а также многих других программах. Щелчок на имени файла запускает связанное с ним приложение.

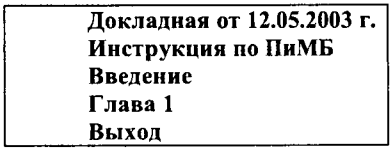


Рис. 4.10

Просмотр папок

Файлы можно находить в папках, просматривая устройства компьютера и их файловую структуру с помощью управляющих элементов окна диалога **Открытие документа**.

После выбора дискового устройства в окне диалога **Открытие документа** список папок и файлов обновляется в соответствии с выбранным диском. С помощью двойного щелчка на папке из списка можно просмотреть файлы и папки следующего уровня, хранящиеся в данной папке. При переходе в новую папку ее содержимое появляется в списке. Найденный файл можно в любой момент открыть двойным щелчком на нем в окне диалога или выделением файла с последующим нажатием кнопки **Открыть**.

Предварительный просмотр файлов в окне диалога Открытие документа

В окне имеется раскрывающееся поле со списком из четырех кнопок, которые позволяют узнать больше о размере файла, его типе и содержимом *перед тем,* как его открывать. На рис. 4.9 показано окно диалога после нажатия кнопок **Вид/Просмотр**. Содержимое документа, выделенного в списке, появляется в дополнительном окне с вертикальной полосой прокрутки.

Автоматизированный поиск файлов

При поиске файла желательно знать одну из следующих его характеристик:

- частичное или полное имя файла;
- тип файла (документ Word, лист Excel или база данных Access);
- одно или несколько слов в тексте файла или в его перечне свойств;
- период времени, прошедший с момента последней модификации файла;
- местоположение папки, в которой следует осуществлять поиск.

Поиск по тексту или свойству

Если забыто имя файла, но известна часть текста файла, то можно попробовать произвести поиск с помощью команд **Сервис/Найти** (рис. 4.11)

Сложный поиск

В окне диалога **Сервис/Найти** также имеется вкладка **Расширенный поиск** (открывающееся окно диалога приведено на рис. 4.12).

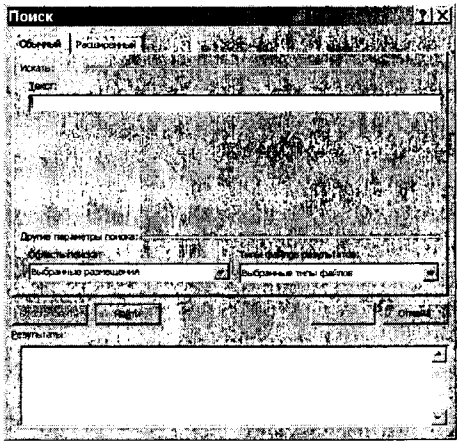


Рис. 4.11

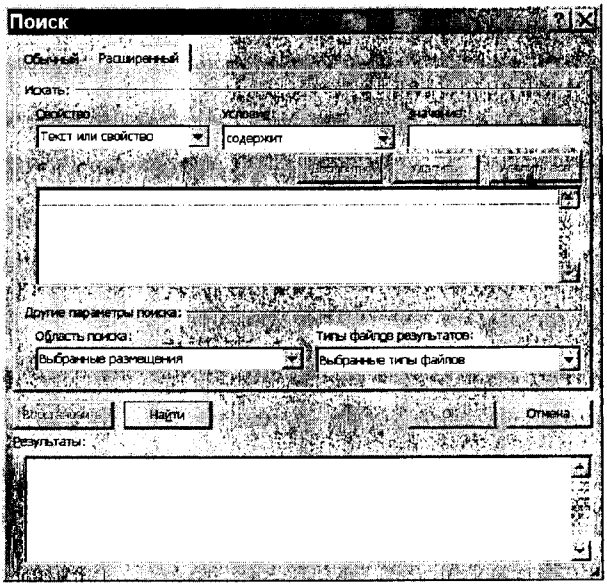


Рис. 4.12

Удаление, переименование и создание ярлыков в окне диалога Открытие документа

Исключительно полезным свойством окна диалога **Открытие документа** является возможность удалять и переименовывать файлы в текущей папке, а также создавать ярлыки документов в папке **Избранное**. Пользователь может производить необходимые файловые операции во время работы с приложениями Office, ему уже не приходится вызывать Проводник Windows каждый раз, когда понадобилось удалить или переименовать файл.

Проще всего происходит удаление файлов. Чтобы удалить файл из текущей папки, достаточно выделить его в списке, нажать клавишу Delete и подтвердить свое решение щелчком кнопки **Да**.

Чтобы переименовать файл, необходимо выделить его из списка (с помощью клавиатуры или мыши) и щелкнуть на нем мышью (один раз; случайный *двойной щелчок* вызовет открытие документа). Office заключает имя файла в прямоугольную рамку и выделяет его, открывая возможность редактирования имени.

Сохранение файла

Сохранение файла — важный шаг, поскольку до переноса внесенных изменений на диск они хранятся лишь в оперативной памяти. Если у файла уже имеется имя, можно просто обновить файл на диске командой **Файл/Сохранить**. Команда **Сохранить** копирует рабочий документ Office из памяти компьютера на диск, сохраняя документ и защищая от возможной потери данных при неожиданном завершении приложения из-за отказа питания или по другой причине.

Команда **Файл/Сохранить как** позволяет задать имя файла, устройство и папку для хранения файла, а также указать ряд параметров, зависящих от приложения, в том числе и формат сохраняемого документа. На рис. 4.13 показано окно диалога при выполнении команды **Сохранить как** в Microsoft Word.

Раскрывающийся список **Папка** предназначен для указания диска и папки, где будет храниться файл. Office сохранит файл в текущей папке (указанной в этом списке), если только не будет задано другое место. Двойной щелчок на значке папки рядом с ее именем открывает папку. Чтобы подняться в иерархии папок на один уровень вверх, нужно нажать кнопку **Переход на один уровень вверх**. Кроме того, можно перейти в папку **Избранное** нажатием кнопки **Открыть папку Избранное**.

Рекомендации при задании имени файла:

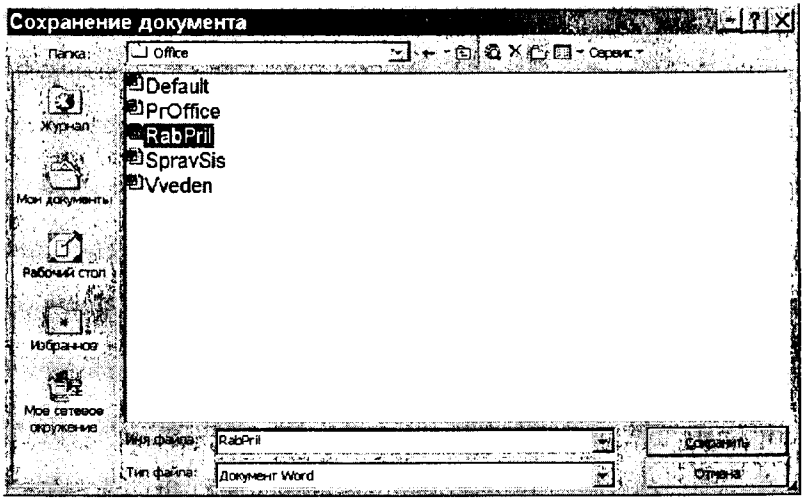


Рис. 4.13

- длина — не более 20 символов (чтобы имя вписывалось в окно диалога);
- не использовать следующие символы /\ < > * « : ;.

Сохранение документа в другом формате

После указания имени и местонахождения файла можно изменить формат, в котором будет сохраняться документ. Каждое приложение Office пользуется своим собственным уникальным форматом для перевода слов и графики на экране в документ. Соответственно переносить документы из одного текстового редактора в другой можно только при условии, что существует общий формат, который понятен обоим приложениям. Форматы, поддерживаемые текущим приложением, будут выведены на экран при входе в меню **Тип файла** окна **Сохранение документа**. Окно диалога **Сохранение** открывается кнопками **Сервис/ Параметры сохранения** в окне **Сохранение документа** (рис. 4.14).

Сохранение и выборка файлов в сети

Один из вариантов сделать документ доступным для членов рабочей группы — поместить документ на совместное сетевое устройство, которое обычно поддерживается администратором сети рабочей группы, который следит за наличием свободного места, обеспечивает

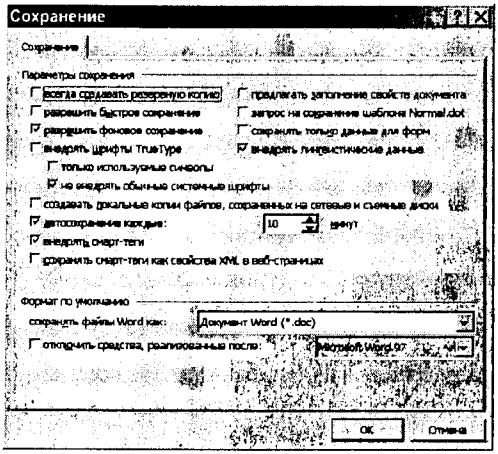


Рис. 4.14

функционирование сетевого оборудования и занимается вопросами технической поддержки и обучения. Полезным средством просмотра сети и перемещения в ней файлов является Проводник Windows, изображенный на рис. 4.15.

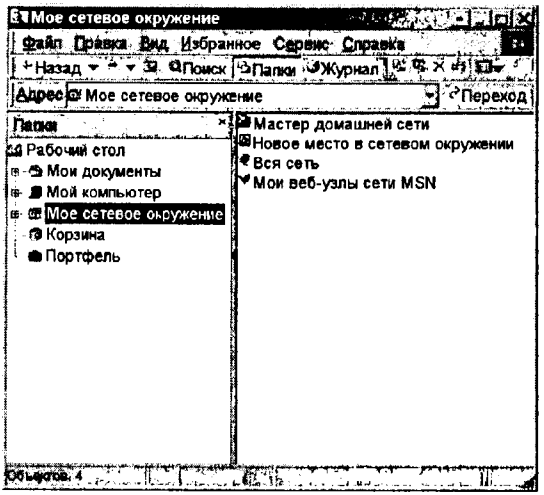


Рис. 4.15

Копирование файла по локальной сети с помощью Проводника Windows выполняется следующим образом.

1. Завершить создание документа в приложении Office (открытые файлы копировать нельзя) и запустить **Проводника** Windows — нажать кнопку **Пуск** и последовательно выбрать: **Программы** и **Проводник**. Просматривая список **Все папки** (All Folders), найти нужный документ Office и щелкнуть на нем правой кнопкой мыши, чтобы открыть контекстное меню.

2. Скопировать файл в буфер обмена командой **Копировать** из контекстного меню.

3. Прокрутить иерархический список папок в левой части окна **Проводника** до самого верха и щелкнуть на сетевом устройстве, которым необходимо воспользоваться. Просмотреть сетевое устройство, пока не будет найдена папка назначения (к ней должен иметь доступ каждый из членов рабочей группы).

4. Выполнить в Проводнике команду **Правка/Вставить**. На сетевом устройстве (в нижней части списка файлов) появляется копия файла, готовая к просмотру.

По электронной почте (или другим способом) известить членов рабочей группы об отправке файла и о том, как им следует с этим файлом работать. Чтобы позднее скопировать документ с внесенными поправками обратно на свой жесткий диск, проделайте эти же действия в обратном порядке (исправленному варианту стоит дать другое имя, чтобы отличить его от исходного).

Рассылка файлов по электронной почте

Наконец, последний способ распространения документов Office по сети — включение их в сообщения электронной почты. Преимущества такого подхода в том, что файлы немедленно доставляются ограниченному кругу пользователей из состава рабочей группы, а также в возможности сопровождать их более длинными и содержательными сообщениями, нежели при использовании команды По маршруту. Для посылки файла по электронной почте следует создать документ и выполнить команду **Файл/Отправить/Сообщение**. Эта команда запускает редактор сообщений электронной почты и вызывает окно диалога, в котором составляются сообщения. Exchange включает в сообщение значок, представляющий отправляемый файл, и при получении сообщения коллеги могут сделать двойной щелчок на этом значке, чтобы просмотреть документ и отредактировать его на своем компьютере.

На рис. 4.16 показан образец сообщения, создаваемого в Microsoft Excel командой **Сообщение**.

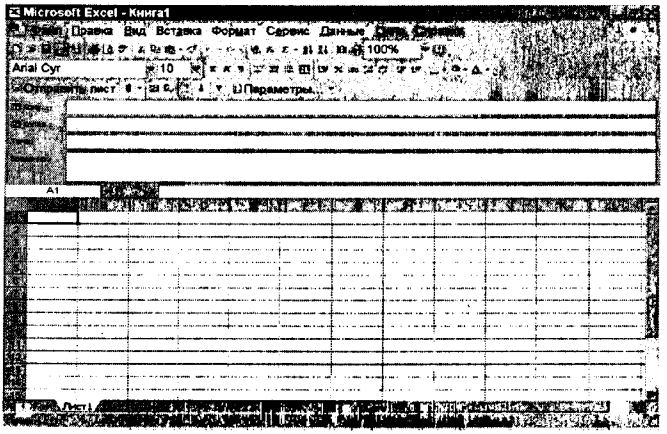


Рис. 4.16

Следует заметить, что команда **Отправить** не проводит документ по маршруту с возвратом его отправителю – она просто доставляет его копию каждому адресату из списка. Такой способ распространения документов в рабочей группе оказывается менее формальным, но требует больше места на дисках.

Работа с Интернетом

Команда **Гиперссылка**, присутствующая в приложениях Word, Excel, PowerPoint и Access, предлагает указать путь к файлу, который может представлять собой документ на жестком диске или допустимый Интернет-адрес. Гиперссылки выглядят как подчеркнутые слова определенного цвета и активируются щелчком мыши. По умолчанию они имеют синий цвет, но при активации он изменяется. При активации гиперссылки Office запускает программу-браузер и устанавливает соединение при помощи модема, факс-модема, линии ISDN или иного коммуникационного устройства. После того, как соединение будет установлено, появляется панель инструментов Web, с использованием которой можно легко переключаться между несколькими открытыми гиперссылками.

4.3. Текстовый редактор MS WORD

Microsoft Word – редактор текстов. Word позволяет, благодаря общему для всех приложений языку VBA, соединить на одной странице разнородные документы (текст, диаграммы, иллюстрации).

4.3.1. Ввод и редактирование текста в MS Word

MS Word предоставляет пользователю возможность выбора режима работы с документом (табл. 4.3).

Таблица 4.3

Режимы работы с документом

Режим	Описание
Обычный (Normal)	Режим общего назначения для быстрого редактирования и форматирования. Не показывает поля и колонтитулы
Режим разметки (Page Layout)	Символы и текст показаны так, как они будут выглядеть на печатной странице, включая поля, колонтитулы. Работают все команды редактирования и форматирования, однако несколько медленнее, чем в обычном режиме
Режим электронного документа (Online Layout)	Формат удобен для работы на экране. Текст не содержит разрывов страниц, мелкий шрифт увеличивается, а окно можно украсить фоновым изображением
Предварительный просмотр (Print Preview)	Показывается одна или несколько печатных страниц; есть возможность настройки параметров страницы
Режим структуры (Outline)	Служит для просмотра общей структуры документа. Позволяет видеть документ с разной степенью детализации и быстро перемещать фрагменты текста
Режим главного документа (Master Document)	Включает средства работы с вложенными документами. Главным называется документ, созданный вставкой других (вложенных) документов в режиме, похожем на режим структуры. Удобен при работе с большими документами

Чтобы переключиться в любой режим (за исключением режима предварительного просмотра), необходимо выполнить соответствующую команду в меню **Вид** (рис. 4.17).

Вид	
	Обычный
	Веб-документ
	Электронный документ (схема документа)
	Разметка страницы
	Структура

Рис. 4.17

Рис. 4.18

Для переключения в режим предварительного просмотра выполнить команды **Файл/Предварительный просмотр** или нажать кнопку **Предварительный просмотр** на стандартной панели инструментов. Кроме того, в режимы **Обычный, Веб-документ, Разметки** и **Структуры** можно переключиться кнопками слева от горизонтальной полосы прокрутки (рис. 4.18).

Изменить размеры текста и графики на экране можно командами **Вид/Масштаб** или с помощью списка **Масштаб** на стандартной панели инструментов.

Вставка специальных символов

Команды **Вставка/Символ** позволяют вставлять разнообразные символы и буквы, отсутствующие на клавиатуре (например, знак авторского права ©).

Можно вставить в текст документа текущее время и дату – команды **Вставка/Дата и время**.

Перемещение курсора по документу

Самый простой способ перевести курсор в нужное место документа, в данный момент видимое на экране, – щелкнуть в этом месте левой кнопкой мыши. В табл. 4.4 представлены сочетания клавиш для перемещения курсора в любое место документа.

Таблица 4.4

Сочетания клавиш для перемещения курсора

Клавиша или Сочетание клавиш	Перемещение
←	К предыдущему символу
→	К следующему символу
↑	На одну строку вверх
↓	На одну строку вниз
Ctrl + ←	На одно слово назад
Ctrl + →	На одно слово вперёд
Ctrl + ↑	На один абзац назад
Ctrl + ↓	На один абзац вперёд
Home	К началу строки
End	К концу строки
Ctrl +Home	К началу документа
Ctrl +End	К концу документа
Page Up	На одно окно вверх
Page Down	На одно окно вниз

Редактирование текста

В табл. 4.5 представлены клавиши для удаления символов.

Поскольку любые изменения параметров текста (тип и размер шрифта, параметры абзаца и пр.) распространяются либо на текст, который будет набран с текущей позиции курсора и далее, либо на выделенный фрагмент существующего текста, знание способов выделения фрагментов текста при работе с текстовым редактором весьма актуально.

Таблица 4.5

Сочетание клавиш для удаления символов

Клавиша или сочетание клавиш	Удаляет
Delete (или команды **Правка/Удалить** (Edit/Clear))	Символ после (справа от) курсора
Ctrl + Delete	Все символы до конца слова, в котором находится курсор (или слова, следующего за курсором)
Backspace	Символ перед курсором (то есть слева от курсора)
Ctrl + Backspace	Все символы от начала слова, в котором находится курсор

Выделение текста с помощью клавиатуры

1. Установить курсор в начало (конец) фрагмента текста.
2. Нажать и удерживать клавишу Shift.
3. Перевести курсор в конец (начало) фрагмента текста, используя клавиши (сочетания клавиш) управления курсором (табл. 4.4).
4. Отпустить клавишу Shift.

Весь документ может быть выделен сочетанием клавиш Ctrl + A.

Выделение текста мышью

Наиболее распространенный способ:

1) щелкнуть мышью на начале (конце) фрагмента текста;
2) нажать левую кнопку мыши и, удерживая ее, перетащить указатель мыши в конец (начало) фрагмента текста;
3) отпустить левую кнопку мыши.

В табл. 4.6 представлены наиболее употребимые способы выделения текста мышью.

Таблица 4.6

Способы выделения текста мышью

Действие	Выделяемый фрагмент
Двойной щелчок на слове	Слово
Щелчок на предложении при нажатой клавише Ctrl	Предложение
Щелчок в полосе выделения рядом со строкой	Строка
Перетаскивание вверх или вниз в полосе выделения	Несколько строк
Двойной щелчок в полосе выделения рядом с абзацем или тройной щелчок в абзаце	Абзац
Тройной щелчок в полосе выделения	Весь документ

Наконец весь документ можно выделить командами **Правка/Выделить все** (Select All).

Перемещение и копирование текста с помощью буфера обмена (clipboard)

1. Выделить текст.
2. Выполнить:
   - или команды **Правка/Вырезать** (для копирования — **Правка/Копировать**);

- или нажать на панели инструментов кнопку  (для копирования –кнопку );
- или нажать Ctrl + X (для копирования – Ctrl +C);
- или щелкнуть на выделенном фрагменте правой кнопкой мыши и выполнить команду **Вырезать** или **Копировать** из контекстного меню.

3. Установить курсор в позицию вставки текста.
4. Выполнить
   - или команды **Правка/Вставить**;
   - или нажать на панели инструментов кнопку  ;
   - или нажать Ctrl + V;
   - или щелкнуть на нужной позиции текста правой кнопкой мыши и выполнить команду **Вставить** из контекстного меню.

Преимущества использования буфера обмена:

- после вставки текст не исчезает из него (в буфере сохраняется копия до того момента, пока в буфер не будет помещена другая информация командами **Вырезать** или **Копировать**);
- буфер обмена может использоваться для перемещения или копирования информации между различными Windows-приложениями.

Перемещение и копирование текста с помощью мыши

1. Выделить фрагмент текста.
2. Переместить указатель мыши на выделенный фрагмент (вид указателя изменится с I на стрелку) и нажать левую кнопку мыши.
3. Для перемещения текста нужно просто перетащить его в новое место. Чтобы скопировать текст, во время перетаскивания нужно держать нажатой клавишу Ctrl.

Данный способ работает только при установленном флажке **Использовать перетаскивание при правке**. Флажок находится на вкладке **Правка** окна диалога, открываемого командами **Сервис/Параметры**.

Поиск и замена текста

Основная процедура для поиска текста в Word

1. Выполнить команды **Правка/Найти** или нажать Ctrl + F, чтобы открыть вкладку **Найти** окна диалога **Найти и заменить** (рис. 4.19).
2. Для проведения быстрого поиска с использованием параметров, заданных при последнем выполнении команды **Найти**, перейти к шагу 3. При необходимости задания параметров поиска нажать кнопку **Больше** и выбрать нужные параметры (рис. 4.20).

3. Если нужно найти определенные слова или предложения, то следует ввести текст в поле **Найти** (нажатие кнопки ? позволит выбрать текст, который искался ранее).

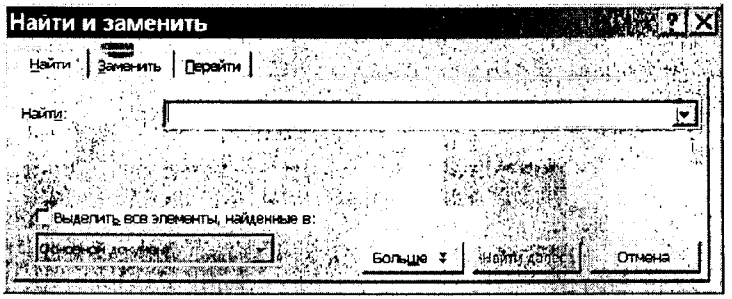


Рис. 4.19

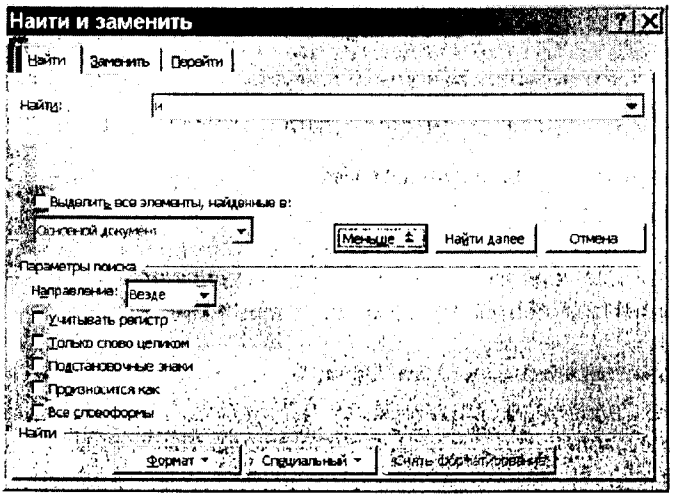


Рис. 4.20

4. Если ищется определенный формат или сочетание форматов, то необходимо нажать кнопку **Формат**, выбрать тип форматирования и указать его атрибуты в появившемся окне диалога (рис. 4.21).

5. Нажать кнопку **Найти далее** для поиска следующего вхождения искомого текста или форматирования.

6. Закрыть окно диалога кнопкой **Отмена**.

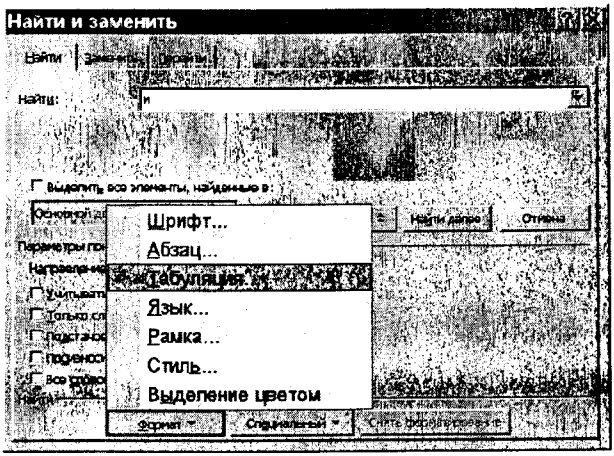


Рис. 4.21

4.3.2. Форматирование документа Word

4.3.2.1. Непосредственное форматирование символов

Использование окна диалога Шрифт

Это окно позволяет накладывать на символы любые атрибуты формата. Для вызова окна диалога нужно выполнить команды **Формат/ Шрифт** (рис. 4.22) или щелкнуть на выделенном тексте правой кнопкой мыши и выбрать из контекстного меню команду **Шрифт**.

Использование окна диалога Абзац

В общем случае абзацы следует форматировать, накладывая на них подходящие стили из списка на панели инструментов форматирования. Для непосредственного же форматирования абзаца необходимо:

1) установить курсор в пределах абзаца или же выделить абзац целиком или частично; для форматирования нескольких соседних абзацев выделить весь фрагмент, в который входит хотя бы некоторая часть каждого из них;

2) открыть окно диалога **Абзац** (рис. 4.23) и выбрать нужные атрибуты формата.

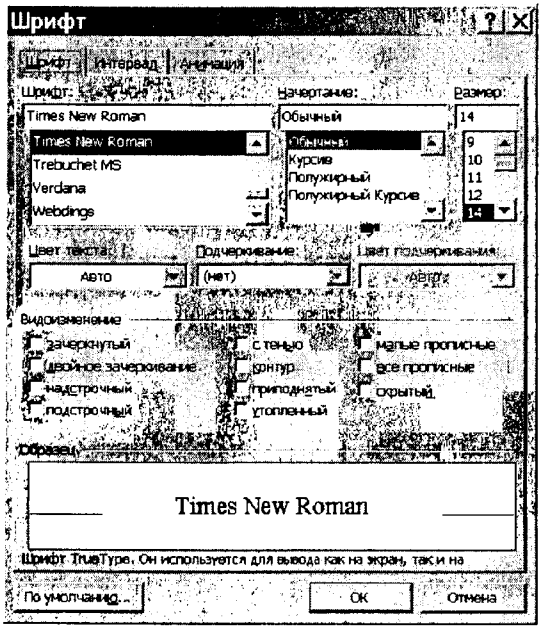


Рис. 4.22

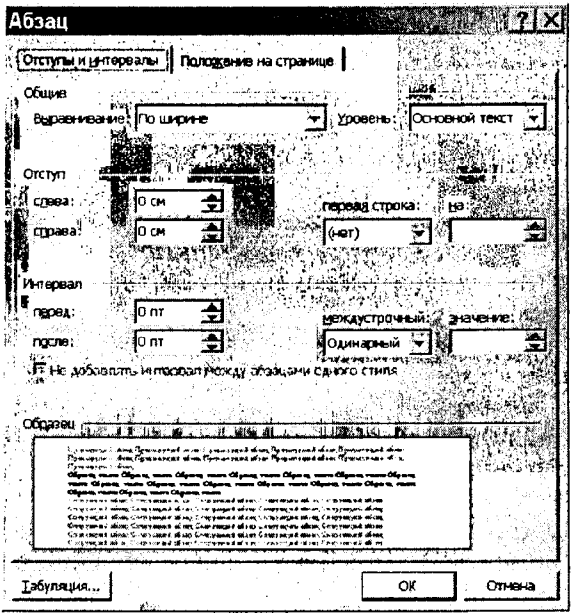


Рис. 4.23

Ввод размера в окна диалога

В некоторых окнах диалога Word приходится вводить размеры, где показывается текущее значение в виде числа, за которым следует единица измерения. В табл. 4.7 приводится таблица соотношений различных единиц измерения, используемых в Office.

Таблица 4.7

Соотношение различных единиц измерения, используемых в Office

Единицы	Сокращение	Пункты	Пики	Сантиметры	Дюймы
Пункты	пт	1	1/12	0,035	1/72
Пики	пк	12	1	0,42	1/6
Сантиметры	см	28,35	2,38	1	0,39
Дюймы	″	72	6	2,54	1

4.3.2.2. Настройка стилей и шаблонов Word

Изменение стилей

Каждый раз, когда открывается новый, пустой документ, он получает копию набора стилей из шаблона, на базе которого он был создан. Конкретный набор стилей, а также атрибуты формата, хранящиеся в них, изменяются в зависимости от выбранного шаблона. Однако в любом случае среди них присутствует базовый набор стилей общего назначения, известных как *встроенные* стили, например, **Обычный**, **Основной текст** и **Заголовок 1 — Заголовок 9**. В некоторых шаблонах предусмотрены дополнительные встроенные стили для особых целей. Например, для форматирования элементов титульного листа в шаблоне **Современный отчет** имеются стили **Название предприятия**, **Заголовок на обложке** и **Подзаголовок на обложке**. Кроме того, в шаблон документа могут включаться стили, определяемые самим пользователем.

Пользователь может изменять любые стили в документе. При изменении стиля весь текст, на который он наложен, получает новые атрибуты формата. В этом состоит важное преимущество стилей перед непосредственным форматированием текста. Поскольку в каждом документе содержится свой собственный набор стилей, изменение стиля влияет только на этот документ; оно не влияет на шаблон или другие документы, созданные на основе этого шаблона.

При изменении или создании стиля следует помнить, что один стиль может быть основан на другом. В шаблоне **Обычный**, поставляемом в

составе Word, стиль **Обычный** является базовым для большинства других стилей абзацев. Например, стиль **Основной текст** определяется как «обычный + интервал после 6 пт». Это определение означает, что **Основной текст** имеет все атрибуты формата стиля **Обычный**, за исключением интервала после абзаца: в стиле **Обычный** интервал после абзаца составляет 0 пунктов, а в стиле **Основной текст** – 6 пунктов. Любые наложенные поверх стиля атрибуты *отменяют* действие атрибутов стиля.

Если изменить такой базовый стиль, как **Обычный**, то мгновенно изменятся все основанные на нем стили. Например, если назначить стилю **Обычный** новые атрибуты – шрифт Courier New и интервал в 10 пунктов после абзаца, то шрифт стиля **Основной текст** также изменится на Courier New. Однако на **Основной текст** не распространяется интервал в 10 пунктов после абзаца, поскольку конкретное значение этого интервала уже указано в нем самом (то есть данный атрибут не наследуется от стиля **Обычный**).

Создание одного стиля на основе другого помогает сделать форматирование более последовательным. Например, если присвоить стилю **Обычный** новый шрифт, то этот шрифт автоматически распространится на все порожденные стили, что позволит избежать наличия разных шрифтов в документе.

Изменение стилей с помощью окна диалога **Стиль**

Возможности:

- переименование стиля;
- выбор стиля, на котором основан изменяемый стиль;
- выбор стиля, который автоматически присваивается абзацу, следующему за тем, которому назначен изменяемый стиль;
- назначение сочетания клавиш для быстрого наложения стиля;
- копирование изменяемого стиля в шаблон документа;
- удаление стилей, созданных пользователем.

Вызов окна диалога (рис. 4.24) производится последовательностью действий **Формат/Стили и форматирование**/щелчок правой кнопкой мыши по имени стиля в диалоговом окне/выбор во всплывшем меню команды **Изменить**. В этом же меню имеется команда удаления выбранного стиля.

Изменение и создание шаблонов документов

В шаблоне хранятся разнообразные элементы, составляющие основу документа Word. При создании нового документа некоторые из этих элементов (например, текст и стили) копируются в него из выбранного

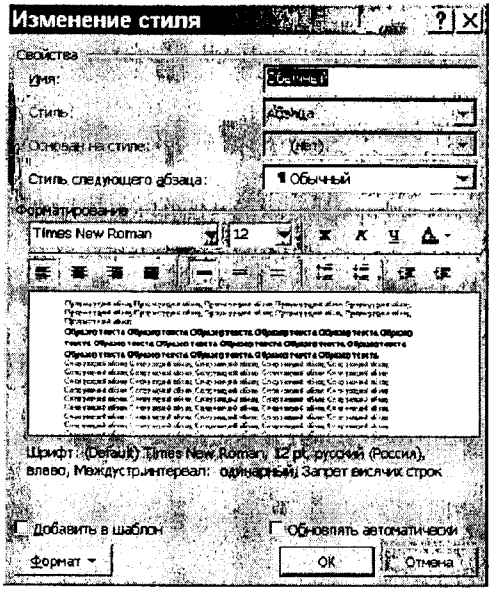


Рис. 4.24

шаблона. Другие элементы (такие, как автотекст и макросы) хранятся в шаблоне. Тем не менее шаблон остается присоединенным к документу, и последний может получить доступ к этим элементам. После создания нового документа и шаблон и документ содержат свою личную копию этих элементов. Изменение каких-либо элементов в документе не влияет на шаблон, а изменения в шаблоне не влияют на документ.

Изменение шаблонов

Шаблоны Word изменяются в результате следующих действий:

• создание элементов автотекста, макросов, панелей инструментов, структур меню или сочетаний клавиш. При создании любого из этих элементов можно выбрать между сохранением его в шаблоне **Обычный** или в шаблоне документа, если они отличаются;

• нажатие кнопки **По умолчанию** в окне диалога **Шрифт/Язык** или **Параметры страницы** и подтверждение запроса. При этом сохраняются атрибуты символов, языка или параметры страницы в шаблоне документа. Атрибуты символов и языка хранятся в стиле **Обычный**.

Можно также сознательно изменить содержание выбранных шаблонов с помощью команд **Сервис/Шаблоны и надстройки**/кнопка **Организатор** (рис. 4.25). В открывшемся окне диалога затем выбрать вкладку, соответствующую типу элемента шаблона, и далее следовать инструкциям диалогового окна.

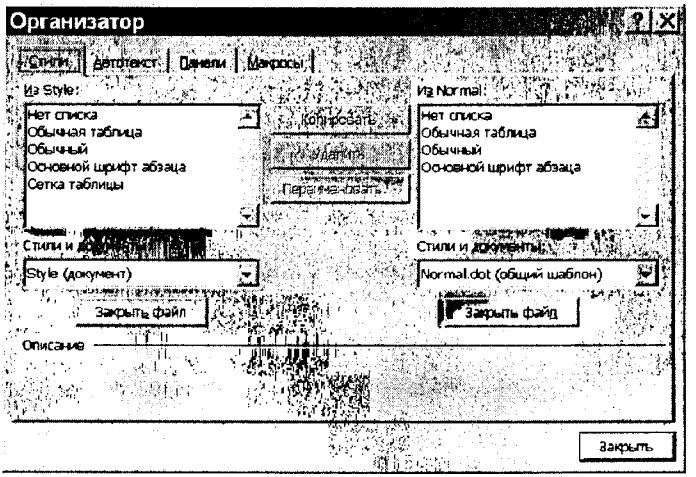


Рис. 4.25

Еще один способ изменения шаблона – открыть файл шаблона и отредактировать его приемами редактирования обычных документов.

1. Выполнить команды **Файл/Открыть** или нажать кнопку **Открыть** на стандартной панели инструментов.

2. В окне диалога **Открытие документа** выбрать из списка **Типы файлов** значение **Шаблоны документов**, а затем выбрать имя изменяемого шаблона. Большинство шаблонов хранятся во вложенной папке **Шаблоны** той папки, в которой был установлен Office (обычно это папка Program Files/Microsoft Office).

3. Отредактировать и отформатировать шаблон приемами работы с обычными документами.

4. Выполнить команду **Файл/Сохранить**.

Создание новых шаблонов

Происходит почти так же, как и процесс создания документа.

1. Выполнить команды **Файл/Создать**.

2. В окне диалога **Создание документа** выбрать раздел **Создание с помощью** шаблона (рис. 4.26) и подраздел **Общие шаблоны** (рис. 4.27).

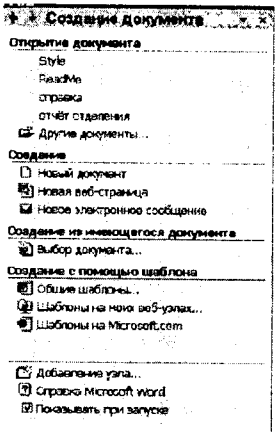


Рис. 4.26

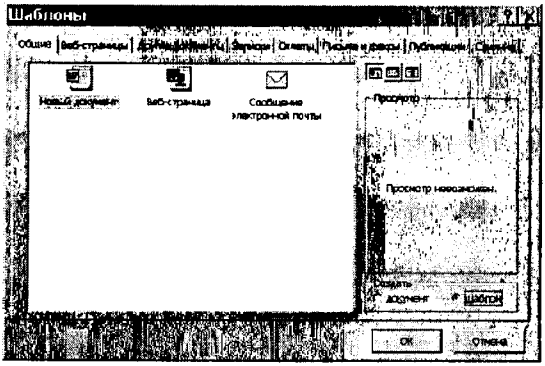


Рис. 4.27

3. Ввести в новый шаблон текст и графику, отредактировать и отформатировать шаблон приемами работы с обычными документами.
4. Выполнить команду **Файл/Сохранить**.

4.3.3. Проверочные средства Word

Автоматическая проверка орфографии при вводе

Необходимо выполнить команды **Сервис/Параметры/**вкладка **Правописание** (рис. 4.28) и установить флажок **Автоматически проверять орфографию**.

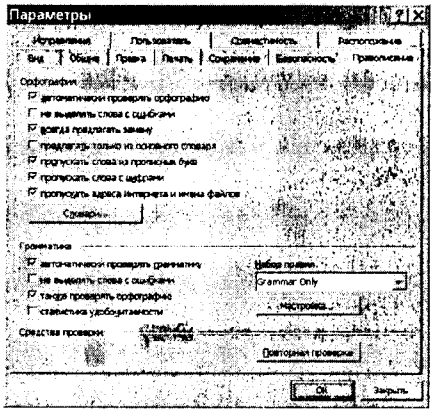


Рис. 4.28

Word сначала проверяет орфографию уже набранного текста, после чего начинает проверять каждое новое слово сразу же после его набора. При обнаружении ошибочного слова (отсутствующего в словаре) оно подчеркивается красной волнистой линией. Можно его пропустить, исправить вручную или щелкнуть на нем правой кнопкой мыши для открытия контекстного меню (рис. 4.29), например, при вводе слова «опчатки».

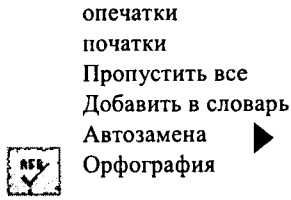


Рис. 4.29

Из контекстного меню следует выбрать:

- один из предложенных вариантов написания слова для его исправления;
- команду **Пропустить все** — для прекращения выделения данного слова на время текущего сеанса (или до нажатия кнопки **Повторная проверка** в окне диалога **Параметры**);
- команду **Добавить в словарь** — слово заносится во вспомогательный словарь, и прекращается пометка этого слова как ошибочного;
- команду **Автозамена** с последующим выбором одного из вариантов написания слова из открывшегося подменю;
- команду **Орфография** для доступа к дополнительным параметрам проверки орфографии.

Ручная проверка орфографии

1. Для проверки во всем документе установить курсор в любом его месте. Для проверки фрагмента документа – выделить его.
2. Выполнить команды **Сервис/Правописание** либо нажать клавишу F7, либо щелкнуть мышью по кнопке **Правописание** на стандартной панели инструментов.
3. При обнаружении ошибок руководствоваться командами диалогового окна **Правописание** (рис. 4.30).

Проверка грамматики

Указывает на возможные ошибки и недостатки – такие, как несогласованность подлежащего и сказуемого или неверные предложные сочетания. Выявляет выражения, свидетельствующие о бедности стиля (например, штампы или неуместные слова).

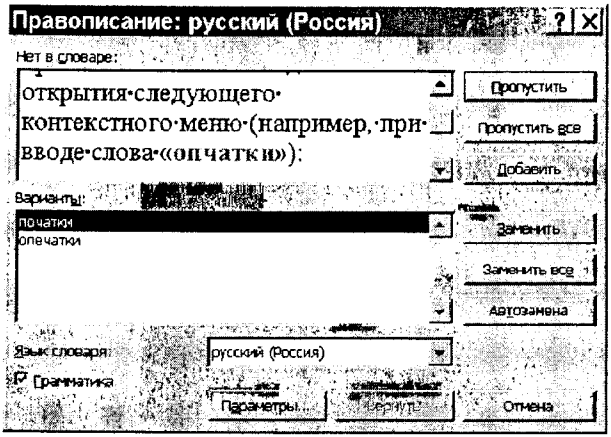


Рис. 4.30

Автоматическая проверка грамматики

Необходимо выполнить команды **Сервис/Параметры/**вкладка **Правописание** и установить флажок **Автоматически проверять грамматику** (рис. 4.31). При нарушении грамматических правил предложение подчеркивается зеленой волнистой линией. При этом можно внести исправления вручную, а можно щелкнуть правой кнопкой мыши на подчеркнутом фрагменте и вызвать контекстное меню, используя которое можно:

- внести исправления, выбрав один из предлагаемых вариантов;
- выполнить команду **Пропустить предложение**;
- выполнить команду **Грамматика** для открытия окна диалога **Грамматика** (рис. 4.31).

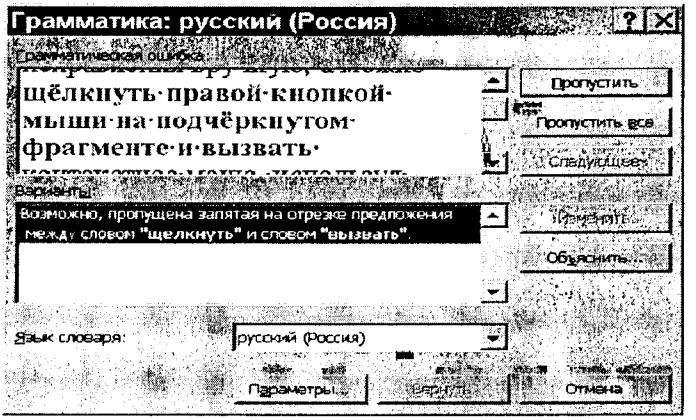


Рис. 4.31

Настройка программы проверки грамматики

Необходимо в окне диалога **Правописание** (рис. 4.30) нажать кнопку **Настройка** и, выбрав характер проверяемого текста, уточнить (изменить) правила, соответствующие выбранному стилю письма (рис. 4.32).

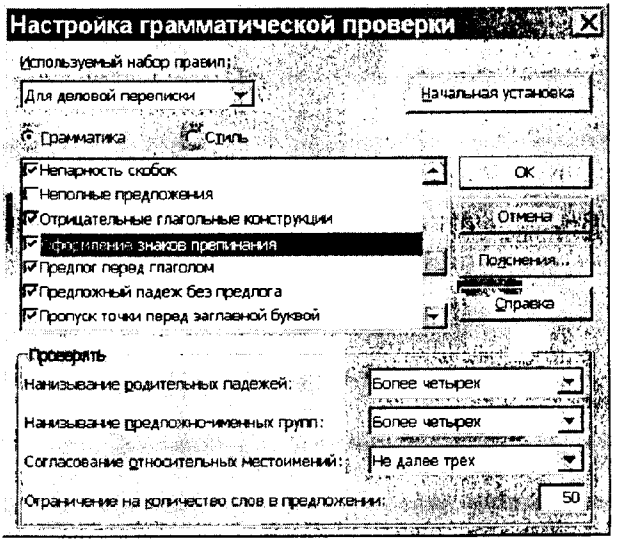


Рис. 4.32

Поиск синонимов в тезаурусе

Тезаурус Word используется для поиска синонимов или антонимов слов и выражений. Для его включения необходимо выполнить ряд действий.

1. Выделить слово или выражение. Для поиска синонимов одного слова достаточно установить в нем курсор.

2. Нажать Shift+F7 или выполнить команды **Сервис/Язык/Тезаурус**. В результате появляется окно диалога (рис. 4.33).

3. В списке **Значения** выбрать нужное значение слова. В списке **Замена синонимом** появится список синонимов для выбранного значения. Если информация о выделенном слове отсутствует в тезаурусе, приводится алфавитный список слов с похожим написанием.

4. Выделить самый удачный синоним (антоним, связанное слово) в списке **Замена** и нажать кнопку **Заменить**. На рис. 4.33 показано, как выглядит диалоговое окно для слова «необходимо» с выбранным значением «нужно» и синонимом «потребно».

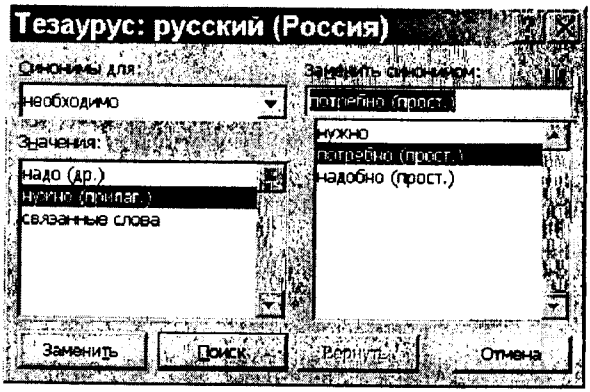


Рис. 4.33

Расстановка переносов

Способы расстановки переносов:

1) автоматически;

2) Word расставляет переносы, но необходимо подтвердить положение каждого из них;

3) ручная расстановка символов переноса нескольких типов.

Автоматическая расстановка переносов

Для этого выполнить команды **Сервис/Язык/Расстановка переносов**. Открывается диалоговое окно (рис. 4.34).

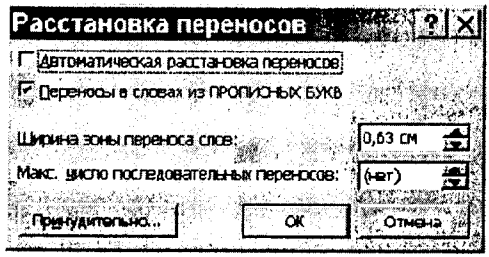


Рис. 4.34

Зона переноса управляет числом переносов, расставляемых Word. Если обнаруживается слово, выходящее за пределы правого отступа, и в конце строки остается интервал, меньший ширины зоны переноса, то Word переносит на следующую строку слово целиком. В противном случае вставляется перенос. Широкая зона переносов уменьшает количество расставляемых переносов, но приводит к более «рваному» правому краю.

Расстановка переносов с подтверждением

Для этого необходимо вызвать диалоговое окно **Расстановка переносов** (рис. 4.34) и нажать кнопку **Принудительно**. Word переключается в режим разметки и остается в нем до окончания расстановки переносов (рис. 4.35).

При этом на выбор предоставляется несколько возможностей:

- для установки переноса в предлагаемом месте нажать кнопку **Да**;
- для установки разрыва в другое место установить курсор в нужную позицию и нажать кнопку **Да**;
- для отказа от переноса слова нажать кнопку **Нет**.

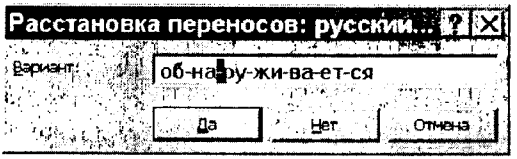


Рис. 4.35

Ручная расстановка переносов

Способы ручной расстановки переносов представлены в табл. 4.8

Таблица 4.8

Способы ручной расстановки переносов

Специальный символ	Сочетание клавиш для вставки	Свойства
Мягкий перенос	Ctrl + Дефис (из верхнего ряда клавиатуры, не на цифровой клавиатуре справа)	Если мягкий перенос приходится на конец строки, он выводится на печать, а содержащее его слово разрывается. Мягкий перенос в строке на печать не выводится
Обычный дефис	Дефис « - »	Всегда выводится на печать. Слово может быть разорвано на месте обычного дефиса
Неразрывный дефис	Ctrl + Shift + Дефис (из верхнего ряда клавиатуры, не на цифровой клавиатуре справа)	Всегда выводится на печать. Слово никогда не разрывается на месте неразрывного дефиса. Может применяться чтобы написанное через дефис слово или выражение всегда находилось на одной строке
Неразрывный пробел	Ctrl + Shift + Пробел	Для вывода нескольких слов (например, названия) на одной строке

4.4. Редактор таблиц MS EXCEL

Microsoft Excel — это программа управления электронными таблицами общего назначения, которая используется для вычислений, организации и анализа деловых данных. С Excel можно решать множество задач — от подготовки простейших счетов или планирования семейного бюджета до создания сложных трехмерных диаграмм или ведения бухгалтерского учета в фирмах средней величины.

4.4.1. Создание и форматирование книги и листа

Процесс создания листа включает запуск Excel, ввод информации, редактирование и перестановку значений в ячейках, добавление формул, сохранение данных и печать. При желании можно даже включить в ячейки листа гиперссылки для обращения к другим файлам, расположенным на жестком диске или в Интернете. При первом запуске Excel в рабочей области приложения открывается новый файл с расширением .xls, который называется рабочей книгой. На рис. 4.36 изображен вид типичного исходного экрана Excel со стандартным главным меню, панелями инструментов, строкой формул, строкой состояния и пустой книгой.

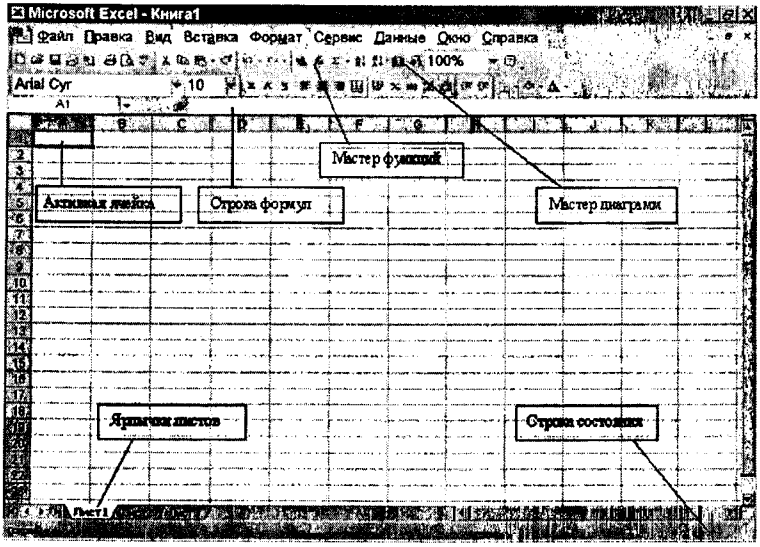


Рис. 4.36

Рабочая книга Excel состоит из рабочих листов. Лист делится на строки и столбцы (листы Excel могут содержать до 65 536 строк и 256 столбцов). Столбцы обозначаются буквами латинского алфавита, а строки – арабскими цифрами. Пересечения строк со столбцами образуют ячейки листа. Например, ячейка на пересечении столбца A и строки 1 – это ячейка A1. Имена ячеек часто называются адресами ячеек. В нижней части окна рабочей книги находятся ярлычки, с помощью которых обращаются к другим листам книги (щелчком левой кнопки мыши на ярлычке нужного листа).

Excel позволяет:

- назначать листам имена (двойным щелчком на ярлычке или заданием нового имени (длина – до 31 символа) в окне диалога, открываемом командами **Правка/Переименовать лист**);
- добавлять к книге новые листы (команды **Вставка/Лист**);
- удалять ненужные листы (команды **Правка/Удалить лист**).

В окне книги имеются полосы прокрутки, с помощью которых можно переходить от одного листа к другому или из одного места активного листа в другое.

Перемещение по листу

Чтобы сделать активной другую ячейку, можно нажать клавиши со стрелками или щелкнуть на нужной ячейке кнопкой мыши. Это называется выделением или подсветкой ячейки (имя выделенной ячейки отображается в поле имен). В табл. 4.9 представлены сочетания клавиш для перемещения по листу.

Таблица 4.9

Сочетания клавиш для перемещения по листу

Клавиши	Перемещение
↑,↓,←, →	К следующей ячейке в выбранном направлении
Ctrl+↑, Ctrl+ ↓ Ctrl+←, Ctrl+→	К следующей ближайшей в выбранном направлении ячейке, содержащей данные (непустой)
Enter	На одну ячейку вниз
Tab	На одну ячейку вправо
Shift+Enter	На одну ячейку вверх
Shift+Tab	На одну ячейку влево
Home	К столбцу A текущей строки
Page Up	На один экран вверх
Page Down	На один экран вниз
Alt+Page Up	На один экран влево

Окончание табл. 4.9

Клавиши	Перемещение
Alt+Page Down	На один экран вправо
Ctrl+Home	К ячейке A1
Ctrl+End	К ячейке последней строки и последнего столбца, в которых содержатся данные
Ctrl+Backspase	Возврат к активной ячейке (или выделенному диапазону), скрытой при прокрутке листа

Некоторые команды Excel работают как с отдельными ячейками, так и с их наборами (диапазонами ячеек). Диапазоны обозначаются двоеточием между адресами крайних ячеек диапазона. Например, обозначение A1:E1 соответствует ряду из пяти ячеек, расположенных вдоль верхнего края листа. На рис. 4.37 выделен диапазон B2:F10, состоящий из 45 ячеек.

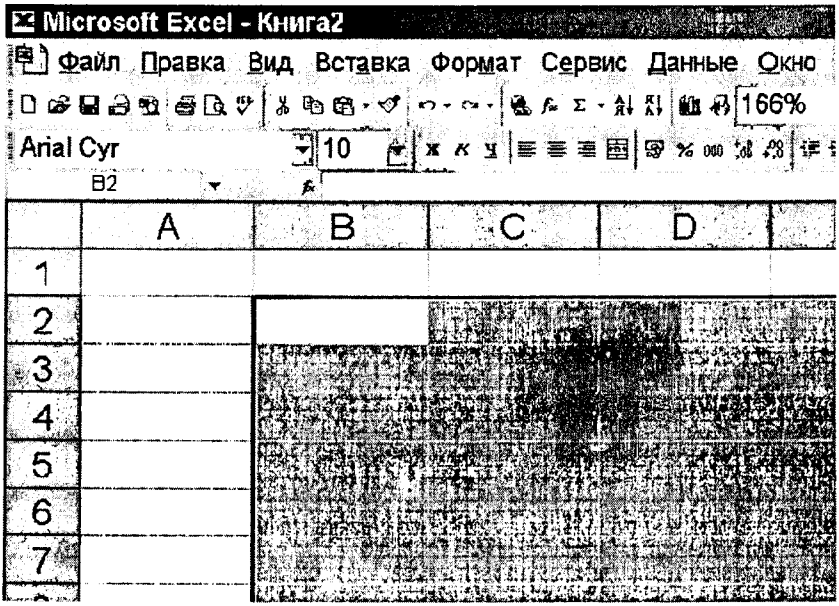


Рис. 4.37

Выделение диапазона ячеек мышью

1. Установить указатель над первой выделяемой ячейкой.
2. Удерживая нажатой кнопку мыши, перетащить указатель по остальным ячейкам диапазона. Отпустить кнопку мыши.
3. Если нужно выделить дополнительные, несмежные диапазоны, то шаги 1 и 2 нужно повторять, удерживая нажатой клавишу Ctrl.

Для выделения целого столбца следует щелкнуть на имени (номере) столбца, для выделения строки – щелкнуть на номере строки. Выделить несколько столбцов или строк можно, выделив один столбец или строку и перетаскивая указатель в нужном направлении. Для выделения всего листа достаточно нажать кнопку, расположенную в левом верхнем углу листа.

Выделение диапазона ячеек с помощью клавиатуры

1. С помощью клавиш перемещения перейти на первую ячейку из числа выделяемых.
2. Удерживая нажатой клавишу Shift, выделить с помощью клавиш перемещения оставшиеся ячейки диапазона. Отпустить клавишу Shift.
3. Для выделения дополнительных, несмежных диапазонов ячеек, нажать Shift+F8. В строке состояния появляется индикатор ДОБ (ADD), означающий, что имеется возможность добавить к выделенному диапазону новые ячейки. Для этого нужно повторить шаги 1 и 2. Для отмены режима добавления повторно нажать Shift+F8.

Ввод информации

Excel разрешает вводить в ячейки следующие виды информации:
- числовые значения (например, числа 15,000, $29,95 и 33%);
- текстовые значения (например, слова Итого, 1-й квартал);
- даты и время суток (например, Янв-96, 11/12/63 или 1:00 PM);
- формулы (например, =B5*1,081 или =СУММ(B3:B7));
- гиперссылки на адреса Интернета и другие документы.

Каждый тип информации имеет свои собственные характеристики формата – это означает, что Excel хранит и выводит на экран элементы каждого типа по-разному.

Информацию о формате, который имеет текущая ячейка, можно получить, выполнив команды **Формат/Ячейки** (рис. 4.38).

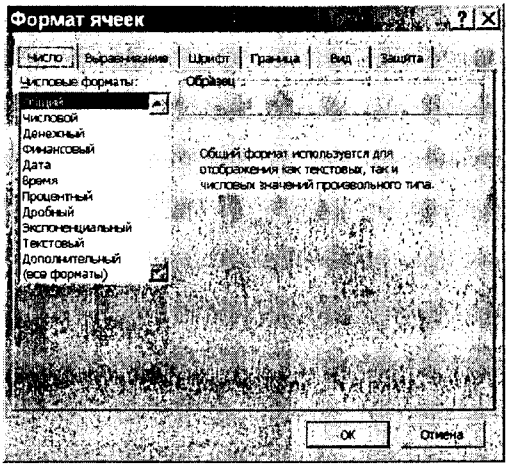


Рис. 4.38

Ввод числовых значений

Чтобы ввести в ячейку число, нужно выделить ее с помощью мыши или клавиатуры, набрать число и нажать Enter. При вводе число постепенно появляется в активной ячейке и в строке формул над листом. Строка формул служит для редактирования содержимого ячеек; если во время набора длинного числа была допущена ошибка, можно щелкнуть мышью на строке формул, подвести курсор к нужной позиции и исправить ошибку. Кроме того, для редактирования содержимого активной ячейки можно сделать на ней двойной щелчок и затем переместить курсор внутри ячейки к месту исправления. Слева от строки формул находится кнопка **Отмена** (Cancel) ✕, нажатие которой приводит к отказу от внесенных в строке формул изменений (если введенное значение еще не было подтверждено нажатием клавиши Enter), и кнопка **Ввод** (Enter), щелчок которой принимает внесенные в ячейку исправления.

Числовое значение может быть целым (32), десятичной дробью (499,95), обыкновенной дробью (10 3/4) или экспоненциальной научной нотацией (4.09E+13). В него могут включаться некоторые символы, среди которых знаки плюс (+) и минус (-), процент (%), дробная черта (/) и экспонента (E), а также знак доллара ($). Если набрать слишком длинное число, которое не помещается в ячейке, Excel автоматически

расширяет ячейку или изменяет формат числа и выводит его в научной нотации, требующей меньшего количества десятичных цифр. Если Excel выводит число в научной нотации или заполняет ячейку символами ########, придется вручную расширить столбец, чтобы увидеть число полностью. Однако при этом Excel продолжает хранить введенное число независимо от его представления в ячейке, поэтому *фактическое значение* данной ячейки всегда выводится в строке формул, когда ячейка становится активной. По умолчанию числовые значения выравниваются по правому краю ячейки.

Ввод текстовых значений

Чтобы ввести в ячейку текстовое значение, нужно выделить ячейку, набрать текст и нажать Enter. Текстовое значение, или *метка,* представляет собой любую комбинацию алфавитно-цифровых знаков верхнего и нижнего регистра, включая цифры и специальные символы. Excel автоматически распознает текстовые значения и выравнивает их по левому краю ячейки. Если соседние ячейки заполнены, то Excel позволяет вводить более длинный текст, перекрывающий расположенные справа ячейки (если в них присутствует информация, текстовое значение обрезается). Тем не менее, Excel продолжает хранить текст полностью, в чем можно убедиться, посмотрев в строку формул, когда ячейка становится активной.

Ввод даты и времени

Для этого нужно воспользоваться одним из заранее определенных форматов даты и времени, чтобы потом соответствующее значение могло быть отформатировано командой **Ячейки** (Cells) или использовано в формуле. Excel хранит дату и время в виде *серийных чисел,* значение которых основано на подсчете количества прошедших дней и начинается с величины 1, соответствующей полуночи 1/1/1900. Такие числа могут быть преобразованы в другие форматы даты и времени, а также использованы в хронологических вычислениях (например, количество дней между двумя праздниками можно определить простым вычитанием первой даты из второй). Серийные числа не появляются в ячейках листа, но они освобождают пользователя от привязки к определенному формату даты или времени после ввода значения. В табл. 4.10 приведены некоторые форматы даты и времени, поддерживаемые в Excel.

Таблица 4.10

Форматы даты и времени, поддерживаемые в Excel

Формат	Шаблон	Пример
Дата	m/d/yy	10/1/99
Дата	d-mmm-yy	1-Окт-97
Дата	d-mmm	12-Окт
Дата	mmm-yy	Окт-02
Время	h: mm AM/PM	10:15 PM
Время	h: mm :ss AM/PM	10:15:30 PM
Время	h: mm	22:15
Время	h: mm :ss	19:30:27
Время	mm :ss.0	15:30.3
Комбинированный	m/d/yy h:mm	10/1/98 22:47

Очистка и удаление ячеек

Для удаления содержимого группы ячеек нужно их выделить и нажать клавишу Delete. Excel удаляет содержимое, но сохраняет форматирование ячеек, чтобы новые значения вводились в том же формате.

Чтобы увидеть полный перечень параметров очистки, следует выполнить команды **Правка/Очистка**.

Команда **Удалить** применяется для удаления из листа диапазонов ячеек, целых строк и столбцов

1. Установить курсор в ячейке, строке или столбце, подлежащих удалению. Для удаления диапазона ячеек его нужно выделить.

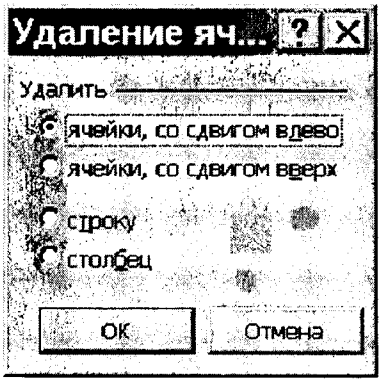


Рис. 4.39

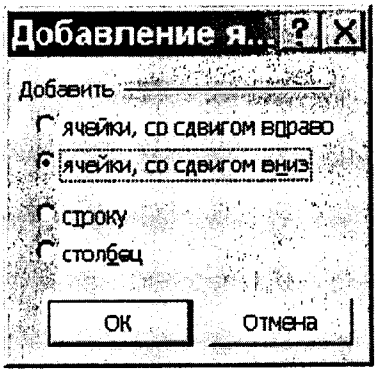


Рис. 4.40

2. Выполнить команды **Правка/Удалить**. Появляется окно диалога (рис. 4.39).

3. Установить нужный способ перемещения ячеек после удаления.

4. Нажать ОК для удаления выделенных ячеек и заполнения пробела другими ячейками.

Командами **Вставка/Ячейки** отдельные ячейки могут быть вставлены в строки или столбцы листа (рис. 4.40). Перед этим следует выделить ячейки снизу или справа от вставляемых ячеек.

Добавление строк к листу

1. Выделить строку под тем местом, где требуется вставить новую строку (строка выделяется щелчком на ее номере).

2. Выполнить команды **Вставка/Строки**.

Добавление столбцов к листу

1. Выделить столбец справа от места вставки нового столбца (столбец выделяется щелчком на его букве).

2. Выполнить команды **Вставка/Столбцы**.

Заполнение рядов текстовых величин, чисел и дат

Excel упрощает задачу ввода данных, позволяя заполнить диапазон ячеек повторяющимся значением или последовательностями значений, называемых рядами. Для ввода ряда значений в диапазон ячеек можно использовать команды **Правка/Заполнить** или специальным приемом работы, называемым автозаполнением.

Автозаполнение включается при перетаскивании по ячейкам маленького черного квадратика, называемого маркером заполнения. Он находится в правом нижнем углу активной ячейки или выделенного диапазона (рис. 4.41). При установке указателя ячейки над маркером заполнения вид указателя меняется на «плюс», означающий разрешение автозаполнения. Набор правил автозаполнения приведен в табл. 4.11 При перетаскивании маркера заполнения вниз или вправо автозаполнение создает значения, характер возрастания которых зависит от последовательности из выделенного диапазона. При перетаскивании указателя вверх или влево автозаполнение создает значения, убывающие соответствующим образом. Если автозаполнение не распознает в значениях выделенных ячеек закономерности, то ячейка просто дублируется.

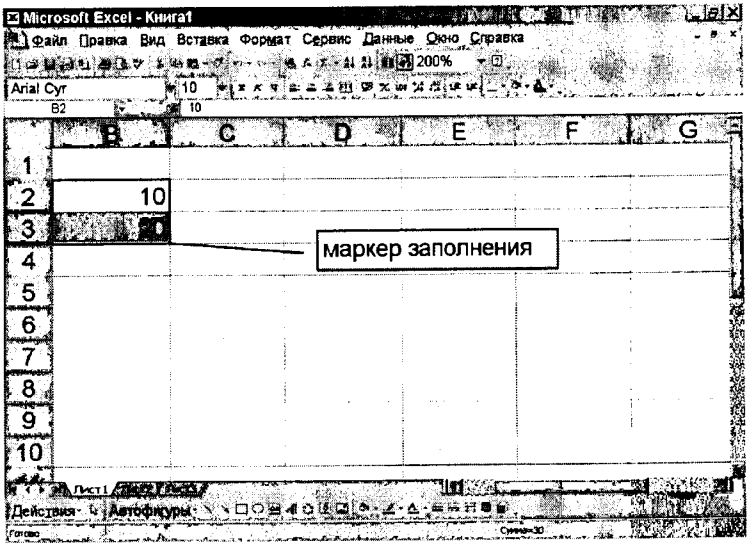


Рис. 4.41

Таблица 4.11

Правила автозаполнения

Тип последовательности	Ряд	Пример
Текст	Изменения отсутствуют, текст дублируется	Блок, Блок, Блок
Числа	Возрастание в зависимости от правил изменения чисел	0, 5, 10, 15, 20, 25 …
Текст с числами	Ряд создаётся изменением чисел в зависимости от правила	Устройство 1, Устройство 2, Устройство 3
Дни недели	Ряд создаётся в соответствии с форматом дней недели	Понедельник, Вторник, Среда
Месяцы	- месяцев	Янв, Фев, Мар
Годы	- года	1999, 2000, 2001, 2002
Время	Ряд создаётся с использованием временных интервалов	1:30 PM, 2:00 PM, 2:30 PM

Создание рядов значений автозаполнением

1. Ввести первую ячейку ряда. Если ряд основан на нескольких значениях, нужно заполнить хотя бы две ячейки для создания закономерности.
2. Выделить все ячейки со значениями, входящие в ряд.
3. Установить указатель над маркером заполнения. Он принимает вид значка «плюс».
4. Перетащить маркер заполнения по ячейкам листа, которые должны быть заполнены значениями ряда, и отпустить кнопку мыши (во время протаскивания всплывает окно с подсказкой).

Окно диалога Прогрессия

Если требуется создать нестандартный ряд (например, геометрическую прогрессию с дробным показателем или ограниченную предельным значением), нужно выделить диапазон заполнения и выполнить команды **Правка/Заполнить /Прогрессия**. Открывается окно диалога (рис. 4.42), в котором нужно задать тип значений и дат, шаг и предельное значение.

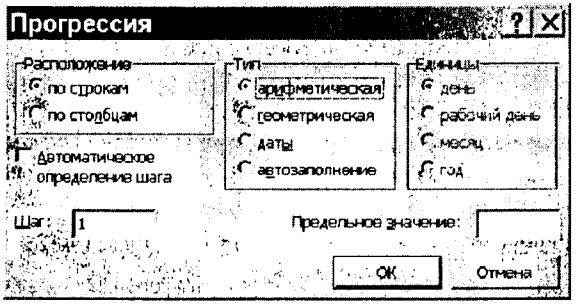


Рис. 4.42

Создание собственных списков автозаполнения

1. Из меню **Сервис** выбрать пункт **Параметры**.
2. Щелкнуть на ярлыке **Списки**.
3. В появившемся диалоговом окне (рис. 4.43) установить указатель списков в строку **Новый список**, в окне **Элементы списка** ввести его составляющие через клавишу **Enter** (вместо этого можно воспользоваться кнопкой **Импорт** списка из ячеек, указав нужный диапазон с заранее введенными элементами создаваемого списка).
4. Щелкнуть на кнопке ОК.

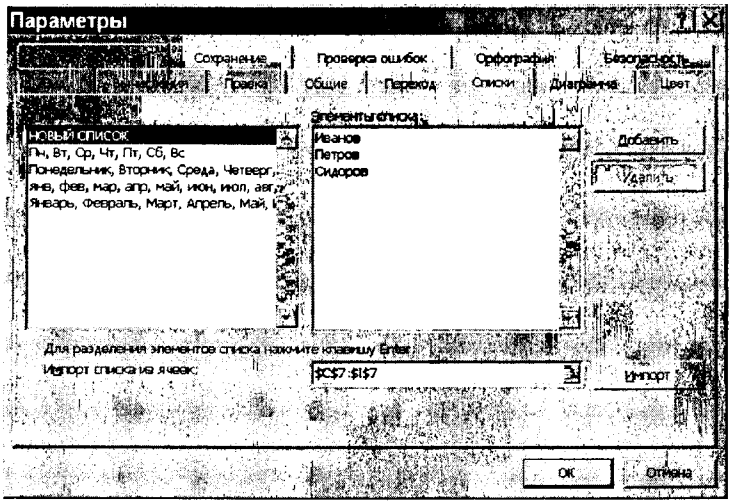


Рис. 4.43

Закрепление шапки таблицы

При работе с достаточно большими таблицами при вертикальной прокрутке окна исчезает шапка таблицы, которую необходимо видеть во время ввода или считывания данных. В Excel для решения этой проблемы предусмотрена возможность фиксации части окна на одном месте, в то время как оставшаяся часть окна будет свободно прокручиваться.

1. Выделить строку, расположенную ниже той, которую предполагается зафиксировать.
2. В меню **Окно** выбрать команду **Закрепить области**.

Для фиксации столбца следует выделить столбец справа от фиксируемого. Для отмены закрепления в меню **Окно** выбрать команду **Снять закрепление областей**.

Разделение окна

Иногда возникает необходимость независимой прокрутки в каждом из двух или более фрагментов. Для этого нужно протащить до нужной позиции указатель разделения (вертикальный или горизонтальный), как показано на рис. 4.44. Затем можно выполнить команды **Окно/Закрепить области**, и указатели разделения примут вид тонких линий.

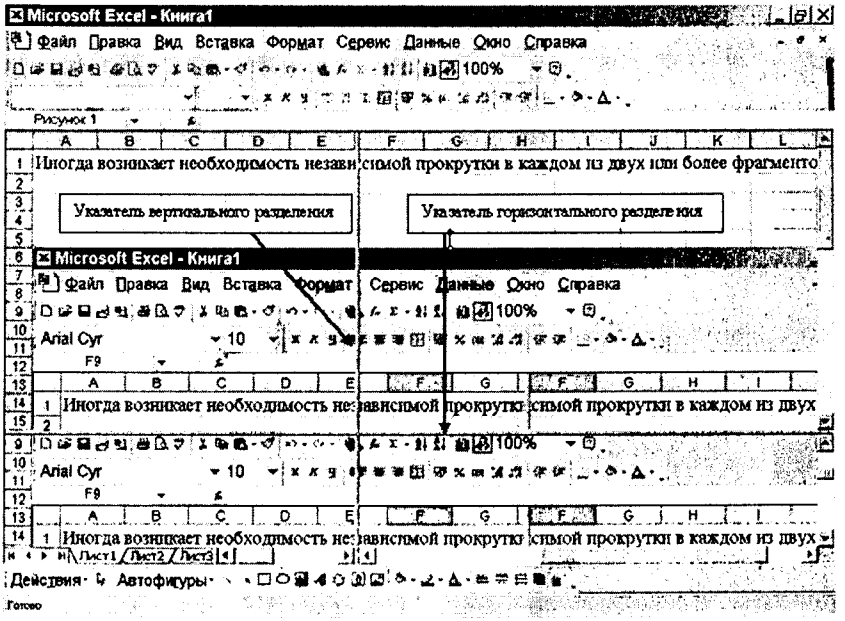


Рис. 4.44

Особенности настройки параметров печати листа Excel

В отличие от Word, следует указать диапазон ячеек, который необходимо распечатать (по умолчанию был бы распечатан диапазон, поместившийся на страницу, размеры и ориентация которой заданы по умолчанию).

Если лист Microsoft Excel не умещается на требуемом числе печатных страниц, допускается изменение масштаба, чтобы печатная копия документа занимала больше или меньше страниц, чем при нормальном размере. Еще одна особенность – возможность напрямую указать принтеру максимально допустимое число страниц в ширину и в высоту при заранее установленных размерах и ориентации этих страниц (рис. 4.45). При этом Excel проведет автоматическое масштабирование выбранного диапазона ячеек для выполнения полученных от пользователя директив.

При использовании параметра **Разместить не более чем на** принудительные разрывы страниц не учитываются. При изменении значения в полях **Разместить не более чем на** Microsoft Excel по необходимости уменьшает образ печати или увеличивает его вплоть до 100 процентов

Рис. 4.45

от натуральной величины. Для того чтобы посмотреть, как новые значения скажутся на размере изображения, щелкните кнопку **ОК**, затем в меню **Файл** выберите команду **Параметры страницы**. Поле **Установить** на вкладке **Страница** показывает, какой процент от натуральной величины будет установлен для образа печати.

4.4.2. Формулы и функции в MS Excel

Любая формула в Excel начинается со знака равенства (=), указывающего Excel на то, что следующий текст является формулой, которую необходимо вычислить и результат вывести в ячейке (если пропустить знак равенства, Excel воспринимает формулу как обычный текст и не вычисляет результат). Элементы, следующие за знаком равенства, являются *операндами*, разделяемыми операторами вычислений. Формула вычисляется слева направо, в соответствии с определенным порядком для каждого оператора в формуле. Предпочтительнее при создании формул оперировать не непосредственно с числами, а с адресами, по которым они находятся. Например, формула в ячейке (например, A3) = 2*3 менее удачна, чем формула =A1*A2 в этой же ячейке, поскольку в случае изменения исходных данных во втором случае не придется редактировать саму формулу. В табл. 4.12 приведен пример использования формул для построения трех функций.

Таблица 4.12

Пример использования формул для построения трех функций

A	B	C	D	E	F	G	H	I	J	K
X	1	2	3	4	5	6	7	8	9	10
$Y_1=x^2$	1	4	9	16	25	36	49	64	81	100
$Y_2=x^3$	1	8	27	64	125	216	343	512	729	1000
$Y_3=2^x$	2	4	8	16	32	64	128	256	512	1024

=B1*B1

=B2*B1

=2^B1

Операторами обозначаются операции, которые следует выполнить над операндами формулы. В Microsoft Excel включено четыре вида операторов: арифметические, текстовые, операторы сравнения и операторы ссылок.

Для описания ссылок на диапазоны ячеек используются следующие операторы (табл. 4.13):

Таблица 4.13

Операторы для описания ссылок на диапазоны ячеек

Оператор ссылки	Значение (пример)
: (двоеточие)	Ставится между ссылками на первую и последнюю ячейки диапазона. Такое сочетание является ссылкой на диапазон (B5:B15)
; (точка с запятой)	Оператор объединения. Объединяет несколько ссылок в одну ссылку (СУММ(B5:B15;D5:D15))
(пробел)	Оператор пересечения множеств, служит для ссылки на общие ячейки двух диапазонов (B7:D7 C6:C8)

Для выполнения более сложных операций по обработке числовой и текстовой информации Excel позволяет включать в текст формул стандартные операции, называемые *функциями*. Функции – заранее определенные формулы, которые выполняют вычисления по заданным величинам, называемым аргументами, и в указанном порядке. В состав Excel входит свыше 250 функций, разбитых для удобства по категориям. Вызов встроенной функции удобнее всего производить с помощью **Масте-**

ра функций, вызываемого щелчком на кнопке **Вставка функции** f_x (рис. 4.46). При вводе функции в формулу диалоговое окно **Мастер функций** отображает имя функции, все ее аргументы, описание функции и каждого аргумента, текущий результат функции и всей формулы.

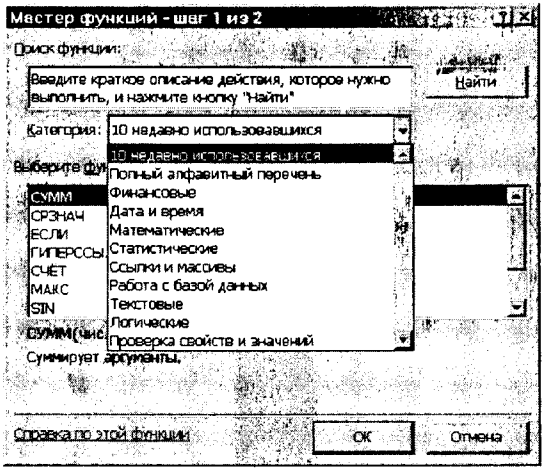


Рис. 4.46

Структура функции начинается со знака равенства (=), за ним следует имя функции, открывающая скобка, список аргументов, разделенных запятыми, закрывающая скобка.

В некоторых случаях может потребоваться использование функции как одного из аргументов другой функции. В формулах можно использовать до семи уровней вложения функций. Например, в следующей формуле функция СРЗНАЧ вложена в функцию ЕСЛИ для сравнения среднего значения нескольких значений с числом 50:

=ЕСЛИ(СРЗНАЧ(F2:F5)>50;СУММ(G2:G5);0).

О ссылках в формулах

Ссылка указывает на ячейку или диапазон ячеек листа и передает в Microsoft Excel сведения о расположении значений или данных, которые требуется использовать в формуле. При помощи ссылок можно использовать в одной формуле данные, находящиеся в разных частях листа, а также использовать в нескольких формулах значение одной ячейки. Кроме того, можно задавать ссылки на ячейки других листов той же книги и на другие книги. Ссылки на ячейки других книг называются *связями*.

Стиль ссылок A1

По умолчанию Microsoft Excel использует стиль ссылок A1, определяющий столбцы буквами (от A до IV, всего не более 256 столбцов), а строки номерами (от 1 до 65536). Например, ссылка B2 указывает на ячейку, расположенную на пересечении столбца B и строки 2 (табл. 4.14).

Таблица 4.14

Ссылка на ячейку или диапазон ячеек

Ячейка или диапазон	Использование
Ячейка в столбце A и строке 10	A10
Диапазон ячеек: столбец A, строки 10–20	A10:A20
Диапазон ячеек: строка 15, столбцы B–E	B15:E15
Все ячейки в строке 5	5:5
Все ячейки в строках с 5 по 10	5:10
Все ячейки в столбце H	H:H
Все ячейки в столбцах с H по J	H:J
Диапазон ячеек: столбцы A–E, строки 10–20	A10:E20

Относительные ссылки

Относительная ссылка в формуле, например A1, основана на относительной позиции ячейки, содержащей формулу, и ячейки, на которую указывает ссылка. При изменении позиции ячейки, содержащей формулу, изменяется и ссылка. При копировании формулы вдоль строк и вдоль столбцов ссылка автоматически корректируется. По умолчанию в новых формулах используются относительные ссылки. Например, при копировании относительной ссылки из ячейки B2 в ячейку B3, она автоматически изменяется с =A1 на =A2.

Абсолютные ссылки

Абсолютная ссылка ячейки в формуле, например, A1, всегда ссылается на ячейку, расположенную в определенном месте. При изменении позиции ячейки, содержащей формулу, абсолютная ссылка не изменяется. При копировании формулы вдоль строк и вдоль столбцов абсолютная ссылка не корректируется. По умолчанию в новых формулах используются относительные ссылки, и для использования абсолютных ссылок надо выбрать соответствующий параметр. Например, при копировании абсолютной ссылки из ячейки B2 в ячейку B3 она остается прежней =A1.

Смешанные ссылки

Смешанная ссылка содержит либо абсолютный столбец и относительную строку, либо абсолютную строку и относительный столбец. Абсолютная ссылка столбцов приобретает вид $A1, $B1 и т. д. Абсолютная ссылка строки приобретает вид A$1, B$1 и т. д. При изменении позиции ячейки, содержащей формулу, относительная ссылка изменяется, а абсолютная ссылка не изменяется. При копировании формулы вдоль строк и вдоль столбцов относительная ссылка автоматически корректируется, а абсолютная ссылка не корректируется. Например, при копировании смешанной ссылки из ячейки A2 в ячейку B3 она изменяется с =A$1 на =B$1.

Коды ошибок в функциях

Если в ячейке появился код ошибки (табл. 4.15), необходимо выделить ее и исправить функцию в строке формул.

Таблица 4.15

Коды ошибок и их описание

Код ошибки	Описание
#DIV/0!	Происходит деление на нуль. Возможно, присутствуют ссылки на пустые ячейки
#NA	Значение отсутствует. Возможно, пропущен аргумент функции
#NAME?	В формуле используется имя диапазона, отсутствующее в книге
#NULL!	Попытка использовать пересечение двух непересекающихся областей. Возможно, в одном из диапазонов-аргументов присутствует лишний пробел
#NUM!	Один из аргументов функции выходит за пределы допустимых значений или недопустим по другим причинам, или же использованная итерационная функция не смогла найти решение
#REF!	Формула включает ссылки на удалённые ячейки
#VALUE!	В качестве аргумента используется текст
#####	Результат вычислений слишком широк, чтобы поместиться в ячейке. Необходимо увеличить ширину ячейки

4.4.3. Создание диаграмм Excel

Диаграммы Excel создаются на основе данных из существующих листов.

Диаграмма строится с помощью **Мастера диаграмм**, который вызывается командами **Вставка / Диаграмма** либо щелчком на кнопке **Мастер диаграмм** на панели инструментов. В результате на экране возникает диалоговое окно (рис. 4.47), отвечая на вопросы которого пользователь указывает тип строящейся диаграммы, диапазон ячеек, в котором хранятся необходимые данные (в случае, если диапазон не был выделен до вызова **Мастера диаграмм**), указывает, какими должны быть заголовки, легенда (пояснение значения того или иного цвета или узора на диаграмме) и подписи данных. Готовую диаграмму можно затем перетащить в нужное место листа и изменить ее размеры обычными приемами работы с рисунками.

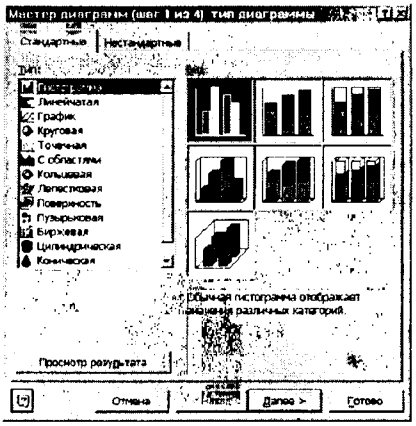


Рис. 4.47

Форматирование диаграммы

Способы решения:

- использование меню **Диаграмма** (рис. 4.48), которое появляется на панели инструментов **Стандартная** вместо меню **Данные** в ходе создания диаграммы);
- использование панели инструментов диаграмм (рис. 4.49), отображается командами **Вид / Панели инструментов**;

Тип диаграммы
Исходные данные
Параметры диаграммы
Размещение
Добавить данные
Добавить линию тренда
Объёмный вид

Рис. 4.48

Рис. 4.49

• вызов контекстного меню щелчком правой кнопкой мыши по интересующей области диаграммы. На рис. 4.50 (диаграмма построена на основе данных табл. 4.12) представлены варианты контекстных меню в зависимости от места щелчка правой кнопкой мыши.

Способы сортировки списков

1. С помощью клавиш  на стандартной панели инструментов (прежде нужно выделить заголовок или ячейку в нужном столбце). Используется для простой сортировки по одному из столбцов. При выделении нескольких столбцов вводится приоритет сортировки слева направо.

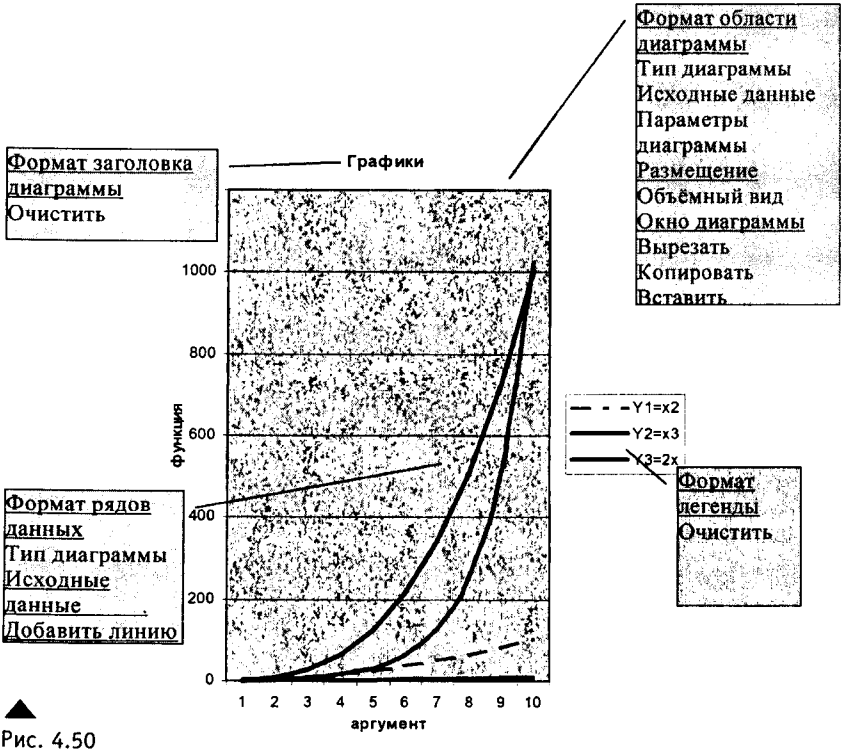


Рис. 4.50

2. Командами **Данные/Сортировка** (диалоговое окно **Сортировка диапазона** представлено на рис. 4.51). Обладает большими возможностями. Для нормальной работы активной должна быть любая ячейка списка.

В списке **Сортировать по** (рис. 4.51) необходимо выбрать нужный столбец из раскрывающегося списка. Установки **по возрастанию** и **по убыванию** определяют порядок сортировки. В полях **Затем по** и **В последнюю очередь по** выбирается при необходимости установка для вторичных сортировок.

Например, чтобы определить, кто из клиентов последним сделал самый дорогой заказ, необходимо отсортировать список по убыванию для столбца «Цена», установив вторичную сортировку по убывающей дате заказа.

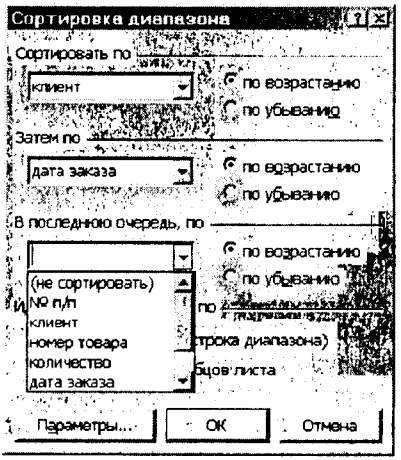
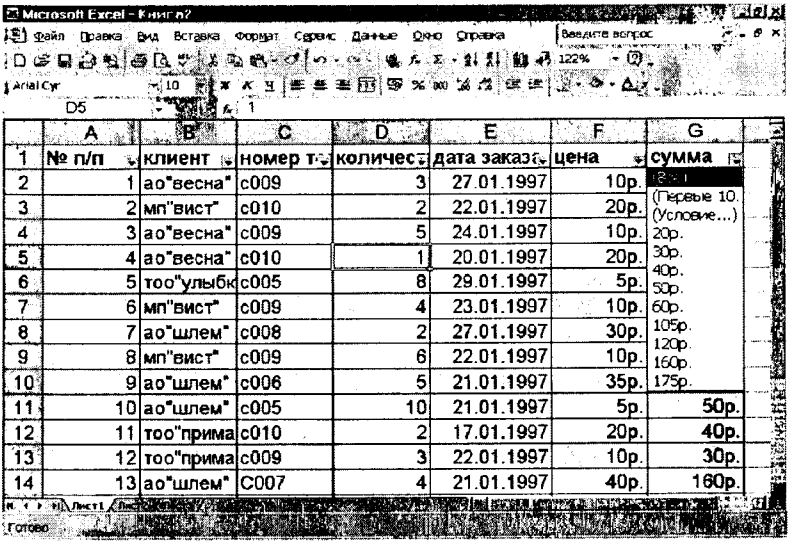


Рис. 4.51

Выбор элементов списка с помощью автофильтра

Выбрать из меню **Данные** команды **Фильтр/Автофильтр**. После этого в заглавной строке каждого столбца появляются маленькие кнопки-стрелки. Щелчок на стрелке приводит к появлению списка, содержащего несколько команд, а также перечень всех элементов этого столбца (рис. 4.52).

Для восстановления всего списка необходимо из меню **Данные** выбрать команды **Фильтр/Отобразить все** или же в раскрывающемся списке автофильтра выбрать опцию «Все».



	A	B	C	D	E	F	G
1	№ п/п	клиент	номер т	количес	дата заказ	цена	сумма
2	1	ао"весна"	c009	3	27.01.1997	10р.	(Первые 10.
3	2	мп"вист"	c010	2	22.01.1997	20р.	(Условие...)
4	3	ао"весна"	c009	5	24.01.1997	10р.	20р.
5	4	ао"весна"	c010	1	20.01.1997	20р.	30р.
6	5	тоо"улыбк	c005	8	29.01.1997	5р.	40р.
7	6	мп"вист"	c009	4	23.01.1997	10р.	50р.
8	7	ао"шлем"	c008	2	27.01.1997	30р.	60р.
9	8	мп"вист"	c009	6	22.01.1997	10р.	105р.
10	9	ао"шлем"	c006	5	21.01.1997	35р.	120р.
11	10	ао"шлем"	c005	10	21.01.1997	5р.	160р.
12	11	тоо"прима	c010	2	17.01.1997	20р.	175р.
13	12	тоо"прима	c009	3	22.01.1997	10р.	**50р.**
14	13	ао"шлем"	C007	4	21.01.1997	40р.	**40р.**
							30р.
							160р.

Рис. 4.52

Настройка автофильтра

Воспользовавшись пунктом «Условие», в списке автофильтра можно сузить список, раскрывающийся при щелчке на кнопке-стрелке автофильтра. Например, можно включить в список только тех клиентов, которые сделали заказы период с 16 января по 21 января.

1. Щелкнуть на стрелке автофильтра в столбце «Дата заказа» и выбрать в появившемся списке пункт Условие.

2. На экране появится диалоговое окно Пользовательский автофильтр. В этом окне следует щелкнуть на верхней левой стрелке в рамке Дата заказа Появится список операторов сравнения (рис. 4.53), из которого выбрать нужный.

Щелкнуть на стрелке верхнего правого поля. Появится список всех элементов, имеющихся в этом столбце (рис. 4.54). Выбрать нужное значение. Отметить оператор **И**, а затем щелкнуть на стрелке в нижнем левом поле и выбрать второе граничное значение.

3. Завершив установки, щелкнуть на кнопке ОК в диалоговом окне **Пользовательский автофильтр**. В результате в списке останутся видимыми только те записи, которые удовлетворяют заданным условиям.

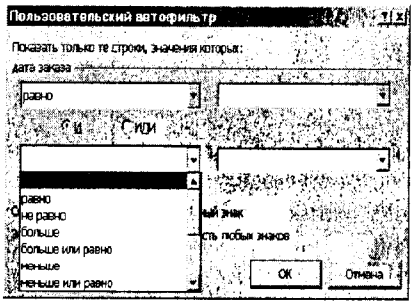


Рис. 4.53

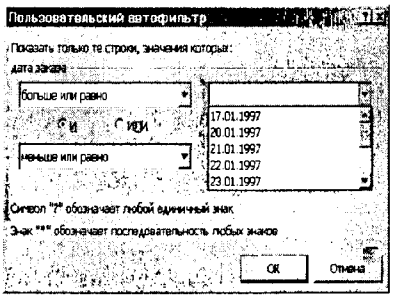


Рис. 4.54

4.5.Система управления базами данных MS ACCESS

Данные представляют собой набор фактов; они превращаются в полезную информацию лишь после того, как будут упорядочены каким-то разумным способом. Из всех приложений Office именно система управления базами данных (СУБД) Access представляет собой инструмент для подобной организации данных.

4.5.1. Общие сведения. Создание таблиц

Состав базы данных (БД)

• *Таблицы*. Структура таблицы определяется в режиме конструирования. Ввод и редактирование данных обычно осуществляется в режиме таблицы. В Access каждая строка соответствует одной записи, а каждый столбец — одному полю.

• *Формы*. В первую очередь предназначены для работы с данными на экране. Обычно выводятся данные из одной записи, либо предоставляется возможность внести данные в одну запись. Одно из преимуществ форм перед режимом таблицы – возможность отображать в одной форме данные из нескольких таблиц.

• *Отчеты*. Предназначены для вывода на печать данных из БД. Конструирование отчета не только позволяет вывести информацию в удобном и привлекательном виде, но и (по аналогии с формами) комбинировать данные нескольких таблиц, а также упорядочивать записи таблицы на основании данных других таблиц. Конструирование отчета происходит в режиме конструктора отчетов. При его создании расставля-

ются специальные элементы управления как для вывода информации, так и для указания ее оформления и положения в отчете. При выводе отчета на печать данные из таблиц оказываются аккуратно упорядоченными и обработанными.

• *Запросы*. С их помощью можно выбрать из БД определенную информацию и упорядочить ее для использования в отчете или для просмотра на экране в форме или таблице. Запросы создаются в режиме конструктора запросов и, по сути, являются вопросами о содержимом БД. Ответы отображаются в режиме таблицы, который очень похож на аналогичный режим просмотра данных. Принципиальное отличие между режимом таблицы для просмотра данных и вывода результатов запроса заключается в том, что в последнем может содержаться информация из нескольких таблиц, выбранная на основании связанных полей.

Процесс создания БД состоит из следующих этапов:

- проектирование и создание таблиц для хранения данных;
- ввод данных;
- разработка других элементов БД, предназначенных для просмотра, редактирования и вывода информации.

Одна таблица может содержать данные о студентах, другая – об учебных дисциплинах. Эти отдельные таблицы необходимо связать воедино. Комбинация всех таблиц и их взаимных связей составляет «фундамент» БД. Приложения, работающие с БД, ориентируются на один из двух основных их видов: плоские таблицы (flat files) или реляционные базы. В плоской таблице вся взаимосвязанная информация должна находиться в одной таблице. Это означает, что любые данные, повторяющиеся в нескольких записях, должны присутствовать в каждой из этих записей. По мере развития и усложнения БД стало ясно, что такой способ неэффективен при хранении больших объемов информации – в результате возникла идея реляционных БД. В реляционной базе используется несколько разных таблиц, между которыми устанавливаются связи (*relations*). Они позволяют ввести информацию в одной таблице и связать ее с записями другой через специальный идентификатор. Хранение данных в связанных таблицах обладает рядом преимуществ:

- экономия времени, так как одни и те же данные не нужно вводить в нескольких таблицах;
- существенное сокращение ошибок, так как повторяющиеся данные вводятся (изменяются) один раз.

Обновление данных (например, ввод сведений о замене преподавателя) является одношаговым процессом, если имена преподавателей

были сохранены в отдельной таблице. Если бы вместо этого имена хранились в таблице ***Студенты и занятия***, то пришлось бы вручную исправлять имена преподавателя для каждой записи.

Все объекты в базе упорядочиваются по своему типу и отображаются на различных вкладках окна БД (рис. 4.55):

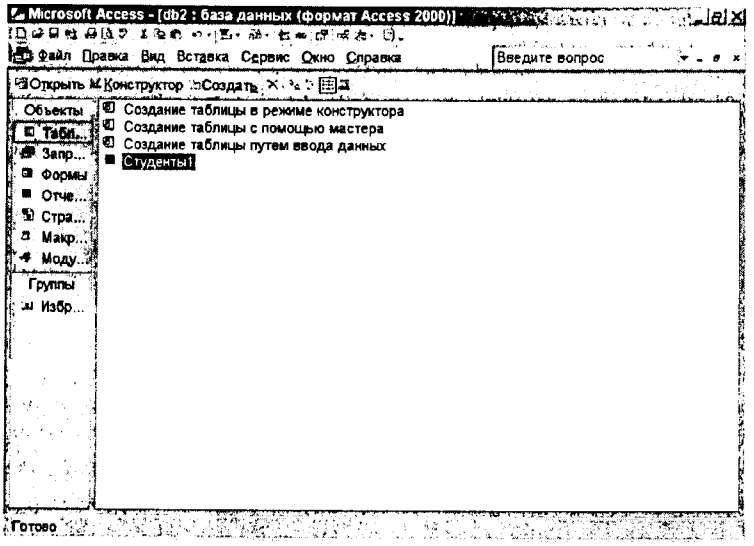


Рис. 4.55

Работа с объектами БД Access происходит в двух различных режимах:

- режим конструирования (переход из режима таблицы – нажатие на кнопку на стандартной панели инструментов);
- рабочий режим или режим таблицы (переход из режима конструирования – нажатие на кнопку на стандартной панели инструментов).

В отличие от открытия пустых файлов в Microsoft Word или пустых листов в Microsoft Excel, все объекты БД в Access (в том числе и таблицы для хранения данных) должны быть сконструированы. В это понятие входит процесс создания объекта и его последующие изменения в соответствии с требованиями пользователя. В большинстве случаев это происходит в специальном режиме конструктора, в котором имеются средства для подобной работы.

После фазы конструирования наступает рабочая фаза, во время которой различные объекты БД используются для ввода и редактирования информации, извлечения информации из базы при помощи запросов и распечатки отчетов, данные которых представлены в нужном виде. Во время рабочей фазы режим конструктора заменяется режимом таблицы или формы, где осуществляются рассмотренные ниже операции по вводу, редактированию и обработке данных.

Этапы проектирования БД

1. Определение входящих в БД таблиц и содержимого их полей.
2. Разработка структуру БД – каким образом поля распределяются по таблицам и в каком порядке.
3. Установка связей между таблицами (если информация в некоторых полях повторяется в ряде записей, следует подумать о помещении этих полей в отдельную таблицу и установке связей). Процесс организации полей и распределения их в одной или нескольких таблицах, а также создания связей называется *нормализацией.*
4. Назначение первичных ключей и индексов. Первичным ключом называется одно или несколько полей, которые однозначно определяют каждую запись в таблице. Наличие индекса помогает Access быстрее находить и сортировать записи. Поля, используемые в качестве первичного ключа, индексируются автоматически, но возможно составить отдельный индекс и для других полей, которые будут использоваться для поиска или сортировки.

Создание таблиц с помощью Мастера

1. Запустить программу Microsoft Access командами **Пуск/Программы/ Microsoft Access**.
2. В окне **Создание** Microsoft Access включить переключатель **Новая база данных** и щелкнуть кнопку ОК.
3. В окне **Файл новой базы данных** выбрать папку своей учебной группы (если ее нет – создать) и дать файлу имя «Студенты». Убедиться, что в качестве типа файла выбрано ***Базы данных Microsoft Access***, и щелкнуть кнопку **Создать**. Откроется окно новой базы – ***Студенты: база данных*** (рис. 4.56).
4. Построить таблицу «Студенты1» новой БД с помощью **Мастера**: выбрать категорию «Деловые», образец таблицы «Студенты» (рис. 4.57).
5. В качестве образцов полей использовать поля ***Код Студента, Фамилия, Имя, Отчество, Адрес*** (при необходимости любое выбранное поле может быть переименовано — см. диалоговое окно).

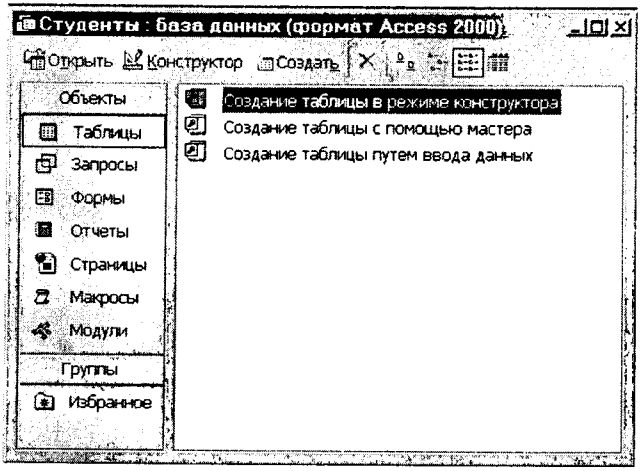


Рис. 4.56

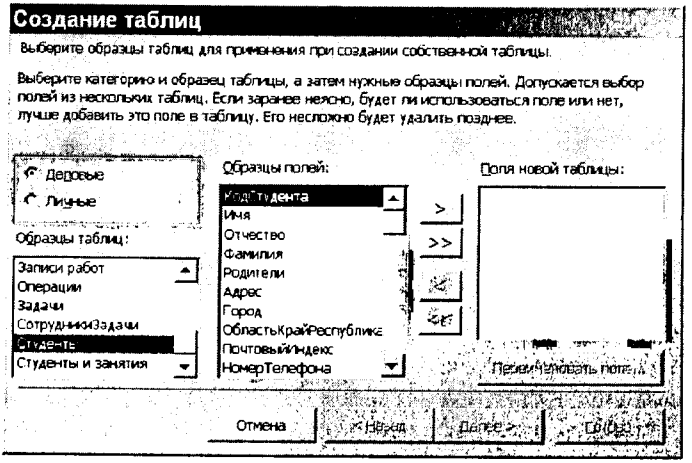


Рис. 4.57

2. После нажатия на кнопку **Далее** ввести имя таблицы «Студенты 1» и нажать кнопку **Готово**.

3. Убедиться в наличии и правильном порядке следования полей созданной таблицы (при необходимости перемещения поля на новое место необходимо выделить нужное поле однократным щелчком мыши и

затем перетащить название этого поля левой кнопкой мыши на новое место).

7. Заполнить записи созданной таблицы данными.

Построение таблицы путем ввода данных (на примере таблицы «Студенты 2»)

1. Дважды щелкнуть на значке **Создание таблицы путем ввода данных** — откроется бланк создания таблицы (рис. 4.58).

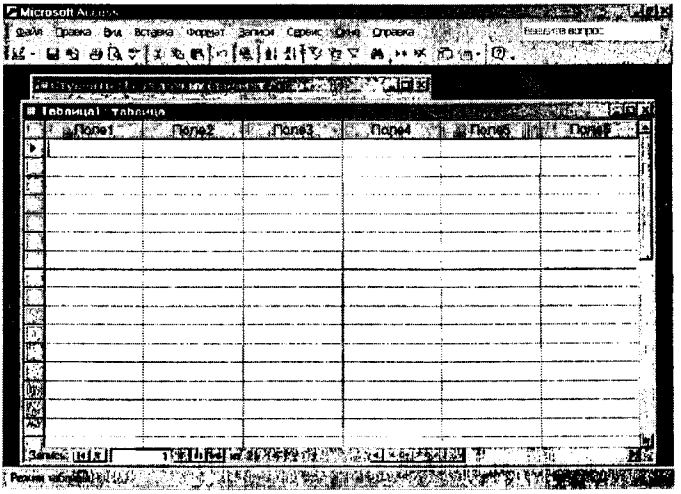


Рис. 4.58

2. Ввести поля, соответствующие таблице, построенной в соответствии с предыдущим заданием.

3. По окончании ввода имен полей и данных при попытке сохранения новой таблицы либо при попытке ее закрытия MS Access будет задан вопрос об имени созданной таблицы («Студенты 2»). После ввода имени на экране появится сообщение (рис. 4.59).

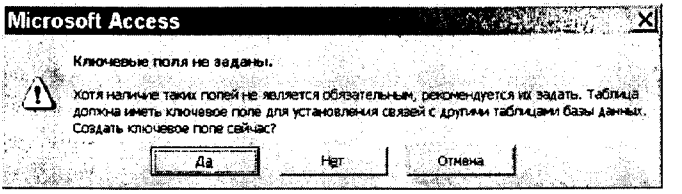


Рис. 4.59

Если ключевое поле не было задано, то следует согласиться с поступившим предложением – этим полем станет поле «Код Студента» – как и в таблице, построенной с помощью **Мастера**.

Построение таблицы в режиме конструктора (на примере таблицы «Студенты 3»)

1. Дважды щелкнуть на значке **Создание таблицы в режиме конструктора** – откроется бланк создания структуры таблицы (рис. 4.60). Ввести поля, соответствующие таблице, построенной с помощью **Мастера**.

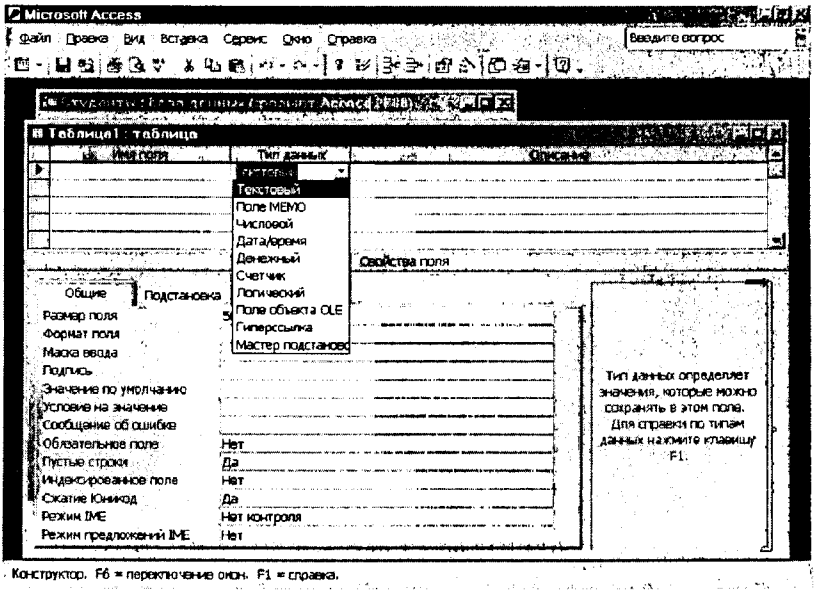


Рис. 4.60

2. Для выбора типа поля щелкнуть кнопку со стрелкой вниз – раскроется список полей (в случае сомнения в выбираемых параметрах одновременно открыть на экране таблицу БД «Студенты 1» в режиме конструктора командами **Окно/Сверху вниз**).

Добавление поля (в режиме конструктора)

Щелкнуть на первой пустой строке столбца **Имя поля** и ввести нужное имя (до 64 произвольных символов, включая пробелы). Затем при-

ступить к заданию свойств, значения которых должны отличаться от принятых по умолчанию. Для вставки поля между существующими полями выполнить одно из следующих действий:

- выделить нижнее поле из пары и выполнить команды **Вставка/Строки**;
- выделить нижнее поле и нажать кнопку  **Добавить строки** на панели инструментов;
- выделить нижнее поле и щелкнуть правой кнопкой мыши – открывается контекстное меню, выполнить команду **Добавить строки**;
- выделить всю нижнюю строку, щелкнув на ячейке в крайнем левом столбце, и нажать клавишу Insert.

Удаление поля

Щелкнуть мышью в любом месте удаляемой строки и выполнить одно из действий:

- команды **Правка/Удалить строки**;
- нажать кнопку  **Удалить строки** на панели инструментов;
- щелкнуть правой кнопкой мыши и выбрать из контекстного меню команду **Удалить строки**;
- выделить всю строку и нажать клавишу Delete.

Перемещение и копирование поля

Выделить всю строку щелчком на ячейке в крайнем левом столбце и выполнить одно из действий:

- перетащить мышью ячейку вверх или вниз;
- выполнить команды **Правка/Вырезать**, затем выделить строку, перед которой должно быть вставлено поле, и выполнить команды **Правка/Вставить**;
- щелчком правой кнопки мыши вызвать контекстное меню и выполнить команду **Вырезать**. Выделить строку, перед которой должно быть вставлено поле, щелкнуть правой кнопкой мыши и выполнить команду **Вставить**;
- вырезать поле сочетанием клавиш Ctrl+X. Выделить строку, перед которой должно быть вставлено поле и нажать Ctrl+V .

Переименование поля

Щелкнуть в ячейке с именем поля и ввести новое имя. Это никак не повлияет на существующие связи и не изменит содержимого БД.

Установка свойств полей

Основные свойства поля определяются его типом (рис. 4.60). Остальные свойства поля задаются на вкладке **Общие** и **Подстановка** в нижней половине окна конструктора (рис. 4.61):

- выбрать нужное поле в верхней части окна;
- перейти на вкладку **Общие** (или на вкладку **Подстановка** для изменения свойств подстановочных полей);
- щелкнуть на изменяемом свойстве и ввести новое значение, в окне справа отображается краткая информация о текущем свойстве, а клавиша F1 вызывает подробную справку.

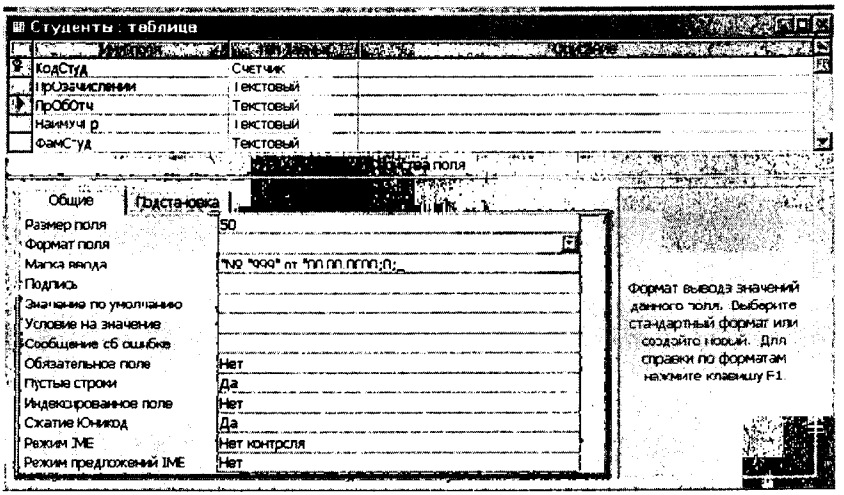


Рис. 4.61

4.5.2.Проектирование структуры БД в MS Access

Ввод и редактирование данных

В качестве примера рассмотрим создание БД, позволяющей хранить следующую информацию о студентах учебного заведения и их успеваемости:

- фамилия, имя, отчество студента;
- адрес (почтовый индекс, адрес) и номер телефона студента;
- номера и даты приказов по учебному заведению о зачислении и об отчислении;

- сведения о специализации;
- наименование дисциплин, изучаемых студентом в каждом семестре;
- количество часов, отводимое учебным планом на каждую дисциплину;
- виды дополнительной отчетности по каждой дисциплине (домашнее задание, расчетно-графическая работа, контрольная работа, курсовая работа);
- виды отчетности по каждой дисциплине (зачет, зачет с оценкой, экзамен);
- результаты сдачи зачетов и экзаменов по семестрам;
- сведения о преподавателях по дисциплинам: фамилия, имя, отчество, наименование кафедры, должность, ученая степень и звание.

Создаваемая БД должна удовлетворять следующим требованиям:

- обеспечивать оперативность доступа к необходимой информации;
- обеспечивать полноту и достоверность хранимой информации;
- иметь минимальный объем;
- не иметь повторяющихся данных большого объема в полях;
- иметь простую логическую структуру;
- допускать возможность расширения числа полей данных в каждой таблице без существенного изменения общей структуры БД;
- обеспечивать удобство анализа хранимой информации;
- иметь возможность создания типовых документов, используемых в учебном процессе вуза (списки учебных групп, зачетные и экзаменационные ведомости, сводные ведомости успеваемости учебной группы и каждого студента за семестр и весь период обучения, приложение к диплому для каждого выпускника и пр.).

Дополнительные требования к создаваемой БД (должны быть учтены при необходимости в ходе доработок БД):

- в случаях заполнения полей кодами данных необходимо обеспечить пользователя БД контекстной помощью по истинным значениям информационных полей, связанных с указанными кодовыми обозначениями;
- при вводе данных, имеющих конечное число значений, иметь возможность не только прямого ввода данных в ячейку, но и выбора указанных значений из всплывающего контекстного меню (поля со списком);
- при необходимости принять меры к исключению ввода данных, не вошедших в перечень, разрешенный для ввода;

- в необходимых случаях обеспечивать контроль правильности вводимых данных путем использования типовых масок ввода и их модификаций (например, для обеспечения правильности ввода номера и даты приказа по вузу о зачислении студента).

Исходя из предъявленных выше требований к создаваемой БД, предлагается следующая структура БД (названия, общее назначение таблиц создаваемой БД и названия полей таблиц БД, взаимосвязи полей таблиц представлены Схемой Данных на рис. 4.62). Жирным шрифтом выделены поля главных таблиц, а обычным шрифтом – поля связанных таблиц.

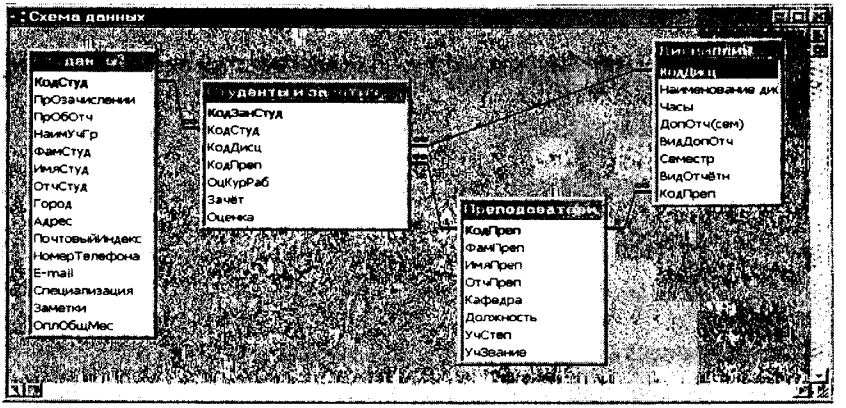


Рис. 4.62

Создание связей

1. Открыть файл БД (все таблицы в нем должны быть закрыты).
2. Открыть окно **Схема данных** (открывается кнопкой либо командами **Сервис / Схема данных**) и проверить наличие всех необходимых таблиц.
3. Для добавления в окно **Схема данных** таблицы выполнить команды **Связи / Добавить таблицу** или нажать кнопку **Добавить таблицу** на панели инструментов. Затем перейти на вкладку **Таблицы**.
4. Выделить нужную таблицу и нажать кнопку **Добавить**. После добавления всех нужных таблиц нажать кнопку **Закрыть**. Окно диалога **Добавление таблицы** открывается автоматически по команде **Схема данных**, если в его окне не содержится ни одной таблицы. Чтобы убрать таблицу из этого окна, щелкнуть на ней в произвольном месте и нажать клавишу Delete. При этом никакие связи данной таблицы не удаляются, просто таблица со связями пропадает из виду.

5. Для создания связи перетащить одно связываемое поле на другое. При этом открывается окно диалога **Изменение связей** (рис. 4.63). В окне перечислены имена полей, участвующих в создании связи. Расположенное внизу поле **Тип отношения** задает тип создаваемой связи (символ «1» обозначает базовую таблицу; символ «?» обозначает подчиненную таблицу; при снятом флажке **Обеспечение целостности данных** между связанными таблицами существует линия без символов).

6. При установке флажка **Обеспечение целостности данных** Access следит за тем, чтобы при вводе или изменении данных не нарушалась связь между таблицами (не позволяет создать подчиненную связь при отсутствии соответствующей записи в базовой таблице; не разрешает изменить значение в связанном поле базовой таблицы при существовании связанных с ней записей в подчиненной таблице; не разрешает удалить запись первичной таблицы при существовании связанных с ней записей подчиненной таблицы). При установке флажка **Каскадное обновление связей** изменение значения в связанном поле базовой таблицы приводит к аналогичным изменениям в соответствующих записях подчиненной таблицы.

7. После задания всех необходимых параметров нажать кнопку **Создать**. Access возвращает пользователя в окно **Схема данных**, в котором появляется линия, обозначающая только что созданную связь.

Изменение связей

Изменить существующую связь можно либо командами **Связи/Изменить связь** при открытой **Схеме данных** либо щелчком правой кнопки мыши по выбранной линии связи в **Схеме данных**. В результате вызыва-

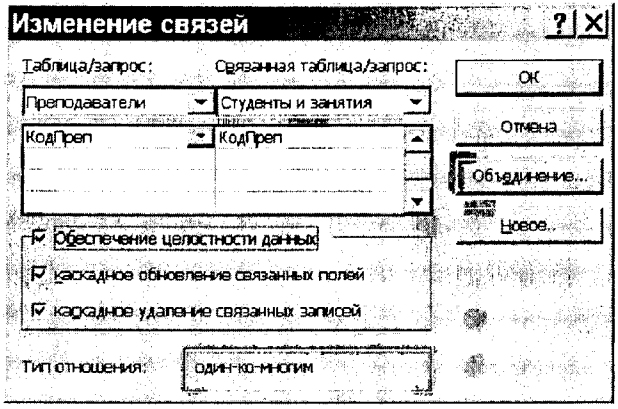


Рис. 4.63

ется диалоговое окно **Изменение связей** (рис. 4.63). Например, для связи по коду преподавателя между таблицами «Преподаватели» и «Студенты и занятия» установить ключи в соответствии с рис. 4.63.

Поиск информации

Самый простой способ поиска нужного фрагмента — команды **Правка / Найти** или сочетание Ctrl+F. В любом случае открывается окно диалога (рис. 4.64). Например, на рис. 4.64 организован поиск студента по заданной фамилии в таблице «Студенты».

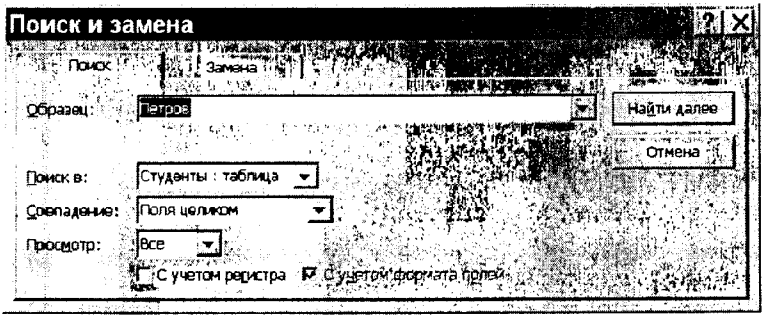


Рис. 4.64

По умолчанию поиск будет организован в поле, где до указанных команд находился курсор (текущее поле). Для поиска по всем полям необходимо указать имя таблицы. Последствия выбора различных значений из списка **Совпадение** представлены в табл. 4.16.

Таблица 4.16

Последствия выбора различных значений из списка Совпадение

Значение из списка совпадение	Критерий совпадения	Пример	Совпадает	Не совпадает
С любой частью поля	Текст может находиться в любом месте поля	ром	Романов кроме паром ром	

Окончание табл. 4.16

Значение из списка совпадение	Критерий совпадения	Пример	Совпадает	Не совпадает
Поля целиком	Введённый текст должен соответствовать всему полю	ром	ром	Романов кроме паром
С начала поля	Текст должен находиться в начале поля, но в конце поля может содержаться дополнительная информация	ром	Романов ром	кроме паром

Воспользовавшись закладкой **Замена**, можно организовать замену, например, одной фамилии на другую.

Фильтрация записей

Самый простой способ фильтрации – использование фильтра по выделенному

1. Найти запись с информацией для отбора.
2. Щелкнуть в любом месте внутри поля или выделить все его содержимое.
3. Выполнить команды **Записи / Фильтр / Фильтр по выделенному** или нажать кнопку  на панели инструментов.

Для фильтрации по выделенному по значениям нескольких полей нужно, чтобы они находились в смежных столбцах (при необходимости переместить столбцы в режиме таблицы). Затем выделить данные, входящие в критерий отбора, и выполнить команду **Фильтр по выделенному.**

Для возврата к просмотру всех записей выполнить команды **Записи / Удалить фильтр** или щелкнуть кнопку  **Удалить фильтр** на панели инструментов.

Использование обычного фильтра

1. Открыть нужную таблицу, например, таблицу «Студенты».
2. Выполнить команды **Записи / Фильтр / Изменить фильтр** или нажать кнопку  **Изменить фильтр** на панели инструментов. Окно с режимом таблицы исчезнет и возникнет окно **Фильтр**.

3. Указать в полях те значения, по которым должна происходить фильтрация (например, по определенному номеру учебной группы). Нажать кнопку **Применение фильтра**.

4. Произвести дополнительную фильтрацию (например, по имени Степан), воспользовавшись командой **Изменить фильтр** и введя соответствующую информацию в поле Имя. Выполнить предыдущее действие, добавив фильтрацию по полю Имя, но перед этим щелкнув на ярлычке **Или** в нижней части окна диалога **Фильтр**.

> **Примечание:** при работе с обычным фильтром не существует возможности поиска значений из заданного диапазона. Например, нельзя найти товар с ценой выше 20 ед, но ниже 40 ед. В таких случаях следует пользоваться **Расширенным фильтром** либо запросами.

Использование Расширенного фильтра

Чтобы избежать двухступенчатого процесса применения фильтра с последующей сортировкой данных и выполнить одновременную фильтрацию и сортировку данных, можно воспользоваться командами **Записи/Фильтр/Расширенный фильтр**. Например, с помощью его помощью можно произвести фильтрацию в таблице «Студенты и занятия» в поле «Оценка» по условию: >3 , но <=5 (рис. 4.65).

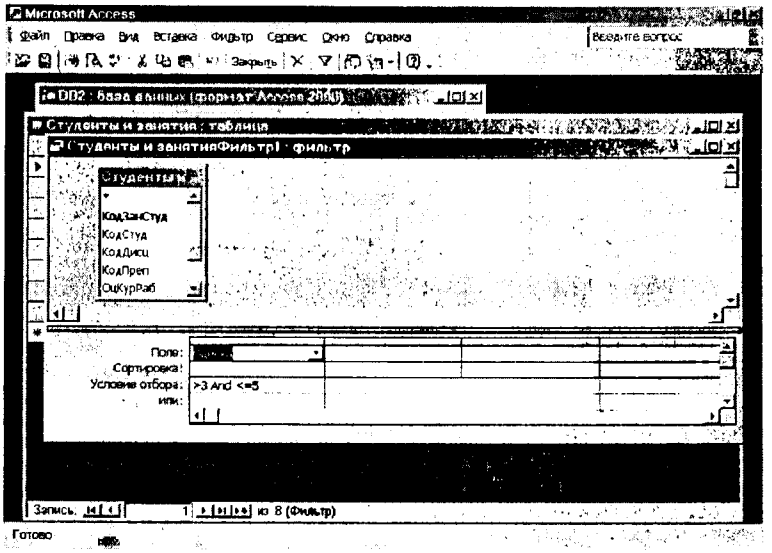


Рис. 4.65

4.5.3. Запросы MS Access

Рассмотрим создание запроса на примере создания сводной ведомости оценок студентов, упорядоченной по фамилиям студентов и номерам учебных групп.

1. Перейти на вкладку **Запросы** в окне БД и выбрать **Создание запроса с помощью мастера** либо выбрать значение **Запрос** из раскрывающегося списка **Новый объект** на панели инструментов.

2. Создать запрос «Оценки» в соответствии с образцом, представленным на рис. 4.66. Результат исполнения этого запроса представлен в табл. 4.17.

3. В режиме конструктора преобразовать запрос таким образом, чтобы записи были отсортированы по возрастанию номера учебной группы, но чтобы поле учебной группы на экране отсутствовало (убрать флажок **Вывод на экран** в поле НаимУчГр на рис. 4.66).

4. Вывести информацию о 25 % наиболее успевающих студентов:

• в режиме **Конструктор** установить сортировку по полю «Оценка» **по убыванию** (рис. 4.67);

• изменить количество отображаемых в запросе записей – в режиме **Конструктор**, не находясь ни в одном из полей, выполнить команды **Вид/Свойства/строка Набор значений** либо нажать кнопку **Набор значений** на панели инструментов (рис. 4.68).

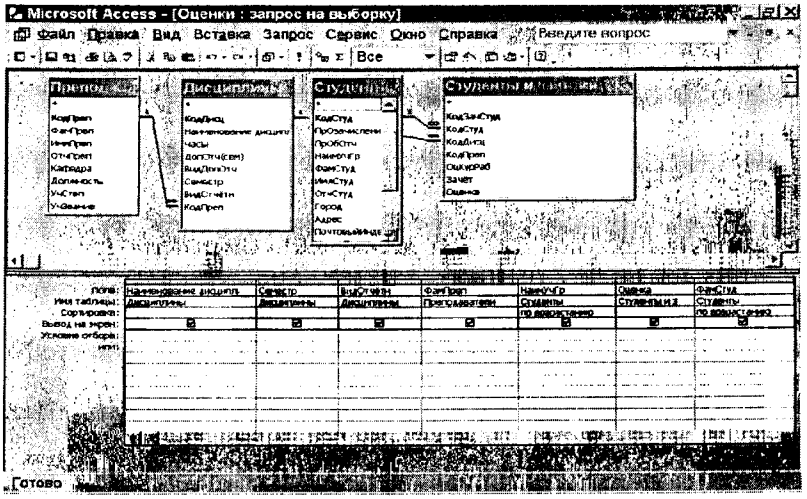


Рис. 4.66

Таблица 4.17

Результат исполнения запроса

Оценки						
ФамСтуд	Наим УчГр	Наименование дисциплины	Се местр	Вид Отчётн	Фам Преп	Оцен ка
Иванов	001	Математическое моделирование	1	экзамен	Крапивин	4
Иванов	001	Информатика	1	зачёт	Ситников	3
Иволгин	001	Математическое моделирование	1	экзамен	Крапивин	5
Иволгин	001	Информатика	1	зачёт	Ситников	5
Петров	001	Математическое моделирование	1	экзамен	Крапивин	4
Петров	001	Информатика	1	зачёт	Ситников	4
Павлов	002	Системный анализ	3	экзамен	Столыпин	4
Сидоров	002	Системный анализ	3	экзамен	Столыпин	3
Иванов	003	Математическое моделирование	1	экзамен	Крапивин	4
Немцов	003	Математическое моделирование	1	экзамен	Крапивин	3
Парамонов	003	Математическое моделирование	1	экзамен	Крапивин	4
Харитонов	003	Математическое моделирование	1	экзамен	Крапивин	5

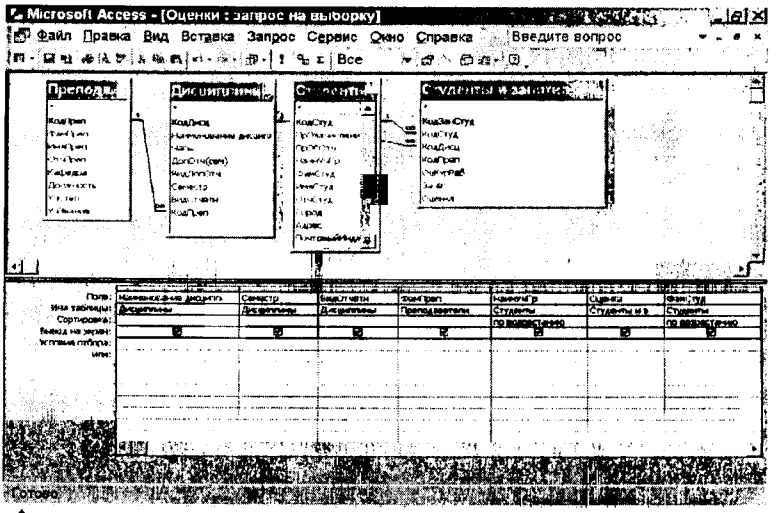


Рис. 4.67

Рис. 4.68

После возвращения в режим таблицы получим результат (табл. 4.18).

Таблица 4.18

Результат запроса

Оценки						
Фамилия студента	Наим уч. Гр.	Наименование дисциплины	Семестр	Вид отчётн.	Фамилия препод.	Оценка
Иволгин	001	Математическое моделирование	1	экзамен	Крапивин	5
Харитонова	003	Математическое моделирование	1	экзамен	Крапивин	5
Иволгин	001	Информатика	1	зачёт	Ситников	5

5. С помощью функции **Групповая операция** вычислить средний балл каждой из групп по рассмотренным дисциплинам.

Примечание: для добавления строки **Групповая операция** в бланк запроса необходимо, находясь в режиме ***Конструктор***, выполнить команды **Вид – Групповые операции** или нажать кнопку **Групповые опе-**

рации на панели инструментов. Пример параметров подобного запроса (режим конструктора) приведен на рис. 4.69;

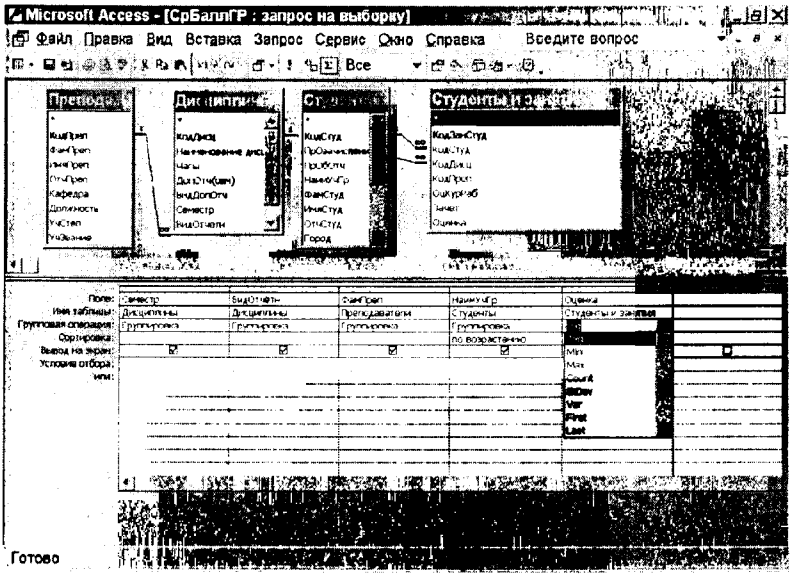


Рис. 4.69

Режим таблицы этого же запроса представлен в табл. 4.19.

Таблица 4.19

Режим таблицы запроса

Средний балл					
Наим. уч. гр	Наимснованис дисциплины	Се-мсстр	Вид отчётн	Фамилия Препод.	Avg Оцснка
001	Информатика	1	зачёт	Ситников	3,66666666
001	Матсматичсскос моделирование	1	экзамсн	Крапивин	4,33333333
002	Системный анализ	3	экзамсн	Столыпин	3,5
003	Матсматичсскос моделирование	1	экзамсн	Крапивин	4

6. Представим себе, что планируется повышение оплаты за пользование общежитием с начала нового учебного года на 15 %. Чтобы выяс-

нить, какова будет величина новой оплаты, ввести вычисляемое поле, а затем подкорректировать его название (рис. 4.70). Для выполнения этого задания необходимо, чтобы таблица «Студенты» включала в себя поле «Оплата общежития ежемесячная».

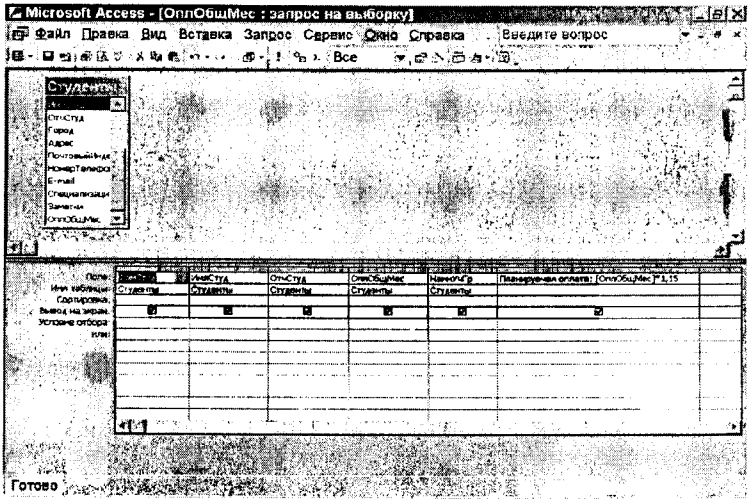


Рис. 4.70

Вид этого же запроса в режиме таблицы представлен в табл. 4.20.

Таблица 4.20

Запрос в режиме таблицы

ОплОбщМес					
Фам. Студ.	Имя студ.	Отч. студ	Наим уч. гр.	Опл. общ. мес	План. оплата
Иванов	Иван	Петрович	001	222,00р.	255,30р.
Иволгин	Пётр	Николаевич	001	312,00р.	358,80р.
Петров	Павел	Сергеевич	001	187,00р.	215,05р.
Сидоров	Алексей	Сергеевич	002	169,00р.	194,35р.
Павлов	Дмитрий	Михайлович	002	254,00р.	292,10р.
Немцов	Борис	Петрович	003	309,00р.	355,35р.
Харитонова	Алла	Ивановна	003	237,00р.	272,55р.
Парамонова	Татьяна	Викторовна	003	194,00р.	223,10р.
Иванов	Сергей	Сергеевич	003	251,00р.	288,65р.
Александров	Игорь	Николаевич	003	187,00р.	215,05р.
				169,00р.	194,35р.

Создание перекрестного запроса с помощью мастера

На рис. 4.71 и в табл. 4.21 приведен пример такого запроса в режиме **Конструктор** и его вид в режиме ***Таблица.***

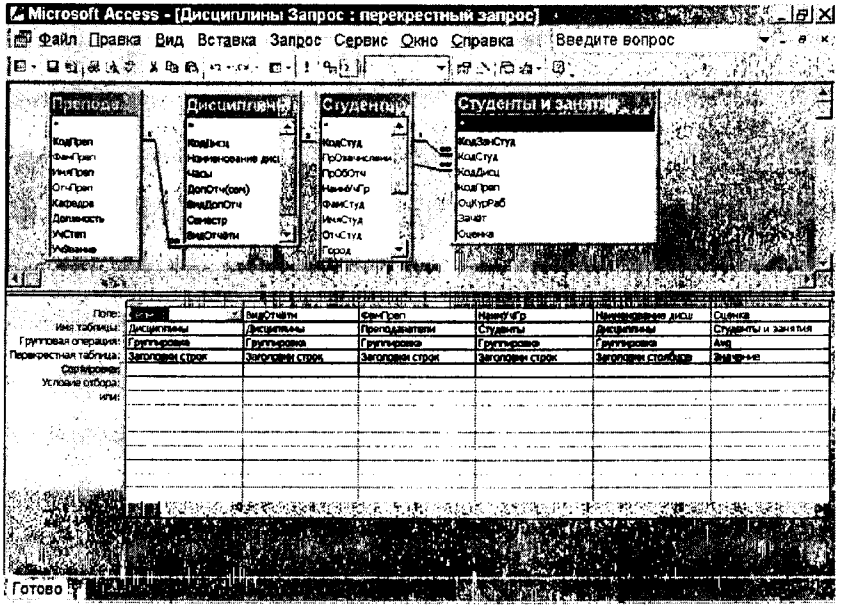


Рис. 4.71

Таблица 4.21

Вид запроса в режиме таблица

Дисциплины Запрос						
Наим уч. гр	Се-местр	Вид отчётн.	Фамилияп реп	Информа-тика	Мат. Модели-рование	Сист. Анна-лиз
001	1	зачёт	Ситников	3,6666		
001	1	экзамен	Крапивин		4,3333	
003	1	экзамен	Крапивин		4	
002	3	экзамен	Столыпин			3,5

Запрос на создание таблицы

Для его организации необходимо (для учебных целей использовать созданный ранее запрос «Оценки»):

- текущий запрос перевести в режим **Конструктора**;
- на панели инструментов **Стандартная** выбрать команду **Запрос** (либо нажать на кнопку **Тип запроса**) и в открывшемся меню выбрать команду **Создание таблицы**. Запрос на создание таблицы полезен при создании резервных копий информации. Таблица создается при каждом запуске запроса (данные существовавшей таблицы замещаются новыми из базового запроса), поэтому запуску предшествует ряд предупреждений:
- запрос на создание таблицы приведет к изменению данных таблицы;
- существующая таблица (в примере – таблица «Оценки1») будет удалена перед выполнением запроса;
- в новую таблицу будет помещено следующее число записей: 12.

Примечание: для возвращения запроса из режима создания таблицы в режим выборки необходимо выбрать нужный запрос в окне БД однократным щелчком левой кнопки мыши, выбрать режим **Конструктор**, затем выполнить команды **Запрос – Выборка**.

Запрос на добавление

Как и запрос на создание таблицы, копирует записи из одной таблицы (или нескольких таблиц) в другое место. Этот вид запроса новых таблиц не создает, а копирует отобранные записи в существующую таблицу. Для изучения этого типа запросов выполнить следующие действия:

- создать 2 копии таблицы «Дисциплины» – «Дис1» и «Дис2»;
- изменить в таблице «Дис1» содержимое полей (прежде всего, названия дисциплин), например, как показано в табл. 4.22;
- с помощью **Мастера** создать запрос на основе таблицы «Дис1» (перенести все поля исходной таблицы, за исключением поля ***Код-Дисц***). Диалоговое окно с выбранными полями показано на рис. 4.72;
- перейти в режим **Конструктора**, выбрать тип – **Запрос на добавление** и выбрать в качестве целевой таблицы (куда будут добавляться записи из таблицы «Дис1») таблицу «Дис2» (рис. 4.73);
- после сохранения запроса в памяти запустить его и убедиться в правильности добавления записей в таблицу «Дис2» (табл. 4.23).

Таблица 4.22

Изменение содержимого полей

Дис1							
Код дисц.	Наименование дисциплины	Часы	Доп. отч (сем)	Вид доп. отч.	Семестр	Вид отчетн.	Код преп.
1	Информатика Дис1	200	1	контр. раб.	1	зачёт	1
2	Системный анализ Дис1	110	1		3	экзамен	2
3	Математическое моделирование Дис1	112			1	экзамен	3
4	Экономическая теория Дис1	150			2	зач. с оц.	2

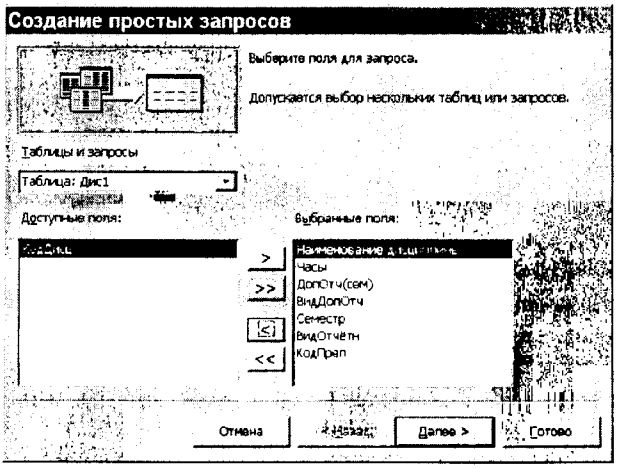


Рис. 4.72

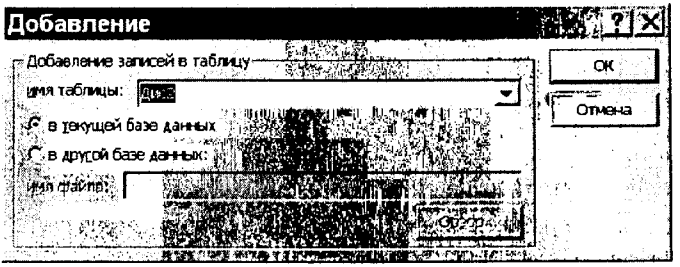


Рис. 4.73

Таблица 4.23

Добавление записей в таблицу

Дис2							
Код дисц	Наименование дисциплины	Часы	Доп. отч. (сем.)	Вид доп. отч.	Семестр	Вид отч.	Код преп.
1	Информатика	200	1	контр. раб.	1	зачёт	1
2	Системный анализ	110	1		3	экзамен	2
3	Математическое моделирование	112			1	экзамен	3
4	Экономическая теория	150			2	зач. с оц.	2
13	Информатика Дис1	200	1	контр. раб.	1	зачёт	1
14	Системный анализ Дис1	110	1		3	экзамен	2
15	Математическое моделирование Дис1	112			1	экзамен	3
16	Экономическая теория Дис1	150			2	зач. с оц.	2

Запрос на обновление

Позволяет изменить значение любого поля БД для записей, удовлетворяющих указанным критериям. При работе с запросом на обновление **Access** добавляет к бланку запроса строку **Обновление**. Она используется для ввода выражений, определяющих способ изменения обновляемого поля.

Для изучения этого типа запросов нужно выполнить следующие действия:

1) создать копию таблицы «Студенты» – «Студ1»;

2) создать новый запрос с помощью **Мастера** на основе таблицы «Студ1», выбрав одно поле – «ОплОбщМес» (рис. 4.74);

3) по окончании создания запроса перейти в режим **Конструктор**, выбрать на панели инструментов тип запроса **Обновление** и в появившейся строке **Обновление** записать выражение **[ОплОбщМес]*1,15** (рис. 4.75);

4) после сохранения изменений в структуре созданного запроса, запустить его на выполнение;

5) проверить правильность обновления записей в поле «ОплОбщ-Мес» таблицы «Студ1» (табл. 4.24).

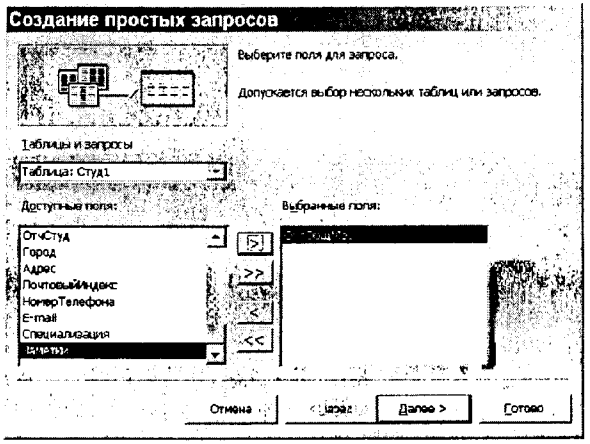


Рис. 4.74

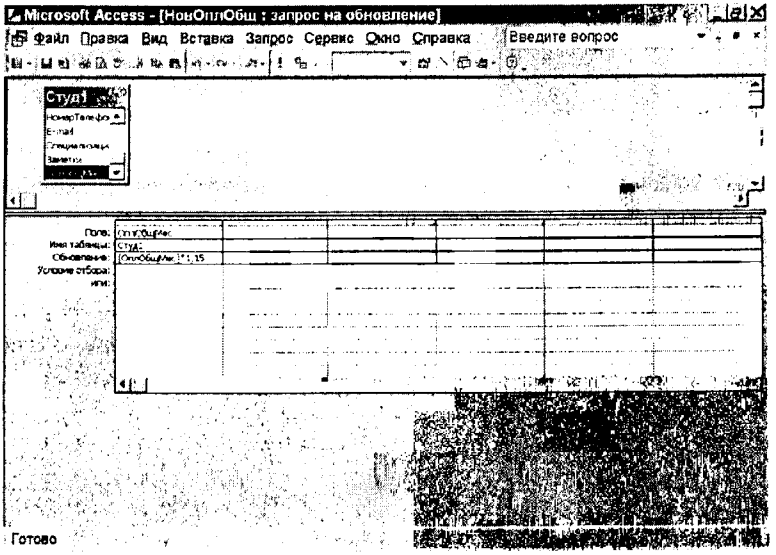


Рис. 4.75

Таблица 4.24

Обновление записей

Код Студ	Наим УчГр	ФамСтуд	ИмяСтуд	ОтчСтуд	Адрес	Тел.	ОплОбщ Мес
1	001	Иванов	Иван	Петрович			255,30р.
2	001	Иволгин	Пётр	Николаевич			358,80р.
3	001	Петров	Павел	Сергеевич			215,05р.
4	002	Сидоров	Александр	Сергеевич			194,35р.
5	002	Павлов	Дмитрий	Михайлович			292,10р.
6	003	Немцов	Борис	Петрович			355,35р.
7	003	Харитонова	Алла	Ивановна			272,55р.
8	003	Парамонова	Татьяна	Викторовна			223,10р.
9	003	Иванов	Сергей	Сергеевич			288,65р.
10	003	Александров	Игорь	Николаевич			215,05р.

Запрос на удаление

Наиболее опасный тип запроса. Уничтожает в заданной таблице (таблицах) все записи, отвечающие заданным критериям. В целях безопасности перед выполнением запроса на удаление следует сначала провести запрос на выборку, в котором условия используются только для отбора записей. Это позволит просмотреть список удаляемых записей и убедиться в том, что из БД выбран правильный набор данных. Только после этого следует переходить к запросу на удаление и уничтожать записи.

4.5.4. Отчеты MS Access

Отчет — это средство отображения данных при выводе на печать. Структуры **Экранных форм** и **Отчетов** похожи, т.е. все, что говорилось о формах, справедливо и для отчетов.

Основная работа происходит в режиме **Конструктора**, то есть **Мастером** создается «заготовка» отчета (чаще всего она не подходит полностью для окончательного печатного документа), а затем в **Конструкторе** этот отчет корректируется/дополняется под нужные требования пользователя. Когда отчет создан, то его выводят на печатающее устройство. Также его можно просматривать и на экране, но необходимо помнить, что отчет создается для печати, т.е. он обычно ограничен размерами стандартного печатного листа.

СУБД Access позволяет создавать отчеты автоматически – **Автоотчеты**. Существуют следующие виды:

- в один столбец;
- ленточный.

Структура этих **Автоотчетов** такая же, как и для **Экранных форм**.

Следующие действия необходимо сделать для создания **Автоотчета**:

1) в окне базы данных выбрать закладку **«Отчеты»**;
2) нажать кнопку ***«Создать»***;
3) выбрать режим создания отчета, например, **Автоотчет в столбец**;
4) выбрать таблицу, на основе которой будет создаваться отчет, например, таблицу **«Список предприятий»**;
5) нажать кнопку **«Ок»**.

Тогда запускается **Мастер автоотчета** и создает автоотчет.

Замечания

1. Отчеты формируются постранично, то есть количество полей большое, то не все поля одной записи могут поместиться на одной странице. Если видна не вся страница, то с помощью линии прокрутки можно посмотреть остальную ее часть. Для просмотра всей страницы (макета страницы) можно нажать кнопку **«Масштаб»**. Имеется также кнопка **«Две страницы»**, которая позволяет увидеть макет двух страниц.
2. Использование **Ленточной** формы автоотчета (все поля в одну строку) может дать очень широкий отчет для большого количества полей.

Использование Конструктора отчетов

Конструктор отчетов используется обычно для модификации структуры отчетов, созданных с помощью **Мастера**.

Структура отчета в **Конструкторе** похожа на структуру **Экранной формы**, только добавлены еще две части:

- *верхний колонтитул* – эта часть располагается в верхней части каждой страницы отчета;
- *нижний колонтитул* – эта часть располагается в нижней части каждой страницы отчета.

Замечание: Мастер при создании отчета вставляет в нижний колонтитул стандартную функцию **Now()**, а также выражение **= «Страница» & [Page] & «из» & [Pages]**, тогда в результате внизу каждой страницы будет выводиться текущая дата и номера страниц в виде:

Пятница 28 мая 2000 Страница 2 из 6.

Такие выражения могут создаваться пользователем с помощью **Построителя выражений**. Для запуска **Построителя выражений** необходимо:

1) вызвать контекстное меню на нужном месте;
2) выбрать пункт ***Свойства***;
3) выбрать закладку ***Данные***;
4) нажать кнопку ***Данные***.

Мастер отчетов

Используется для создания более сложных отчетов, чем **Автоотчеты**.

Запуск **Мастера:**

1) в окне базы данных нажать кнопку ***Создать***;
2) выбрать режим ***Мастер отчетов***;
3) выбрать таблицу, на основе которой будет создан отчет;
4) нажать клавишу ***Ок***, тогда запускается **Мастер.**

Шаг 1: выбрать поля, необходимые в отчете.

Шаг 2: можно определить параметры группировки, например, в **Списке товаров** сгруппировать по **Коду предприятий.**

Шаг 3: выбрать поле, по которому будет проведена сортировка (**Код предприятия**). Можно задать вычисление итоговых значений по числовым полям (по **Цене**).

Шаг 4: определить **макет** отчета.

Шаг 5: задать **стиль** отчета.

Шаг 6: ввести **имя** отчета.

В качестве примера рассмотрим создание ведомости сдачи студентами экзаменов по дисциплинам:

1) открыть БД;
2) для создания ведомости сдачи экзаменов использовать данные следующих полей:
   - наименование дисциплины, семестр – из таблицы «Дисциплины»;
   - фамилия, ученая степень, ученое звание преподавателя — из таблицы «Преподаватели»;
   - фамилия, имя, отчество студента, наименование учебной группы студента — из таблицы «Студенты»;
   - оценка — из таблицы «Студенты и занятия»;
3) запустить ***Мастер*** и в диалоговом окне указать наименования нужных полей (см. п. 2);
4) установить один из вариантов группировки, например, как показано на рис. 4.76;
5) указать порядок сортировки и характер итоговых вычислений (рис. 4.77, рис. 4.78);
6) результат работы ***Мастера*** :
   - в режиме просмотра (рис. 4.79);
   - в режиме **Конструктора** (рис. 4.80);

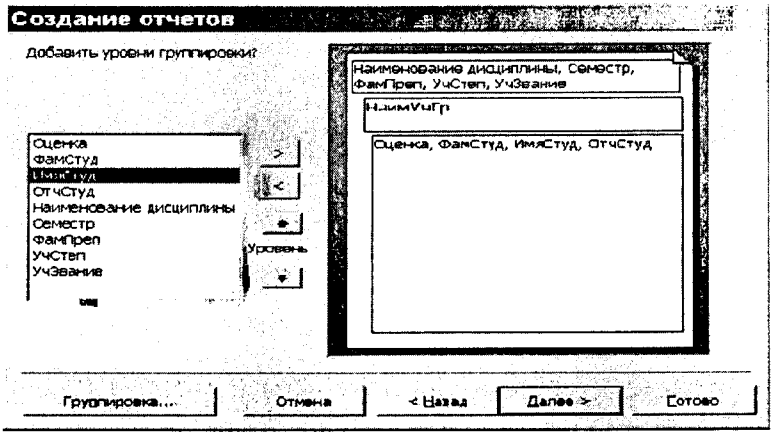


Рис. 4.76

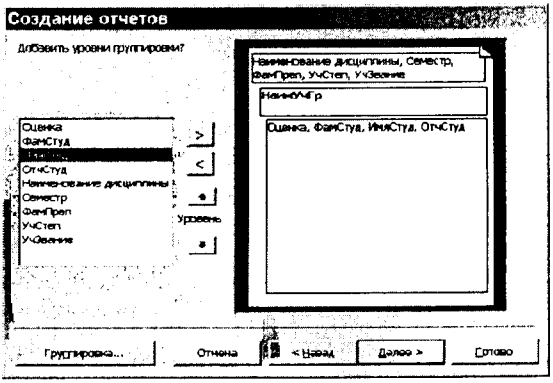


Рис. 4.77

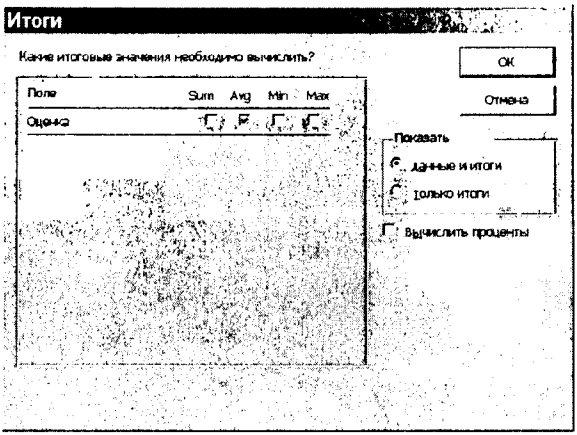


Рис. 4.78

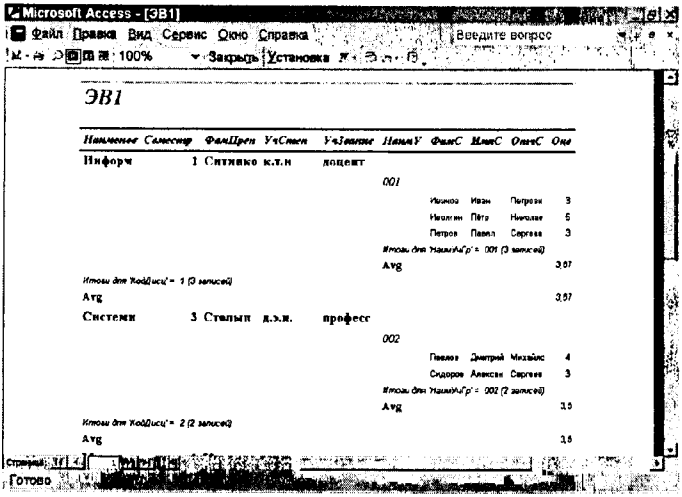


Рис. 4.79

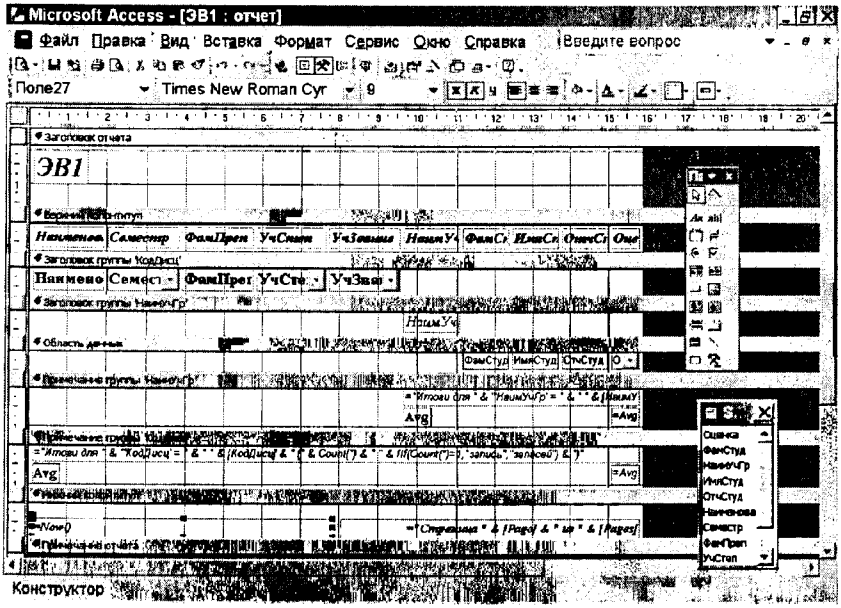


Рис. 4.80

7) перейти в режим **Конструктора** и доработать полученный отчёт с тем, чтобы он принял следующий вид (рис. 4.81).

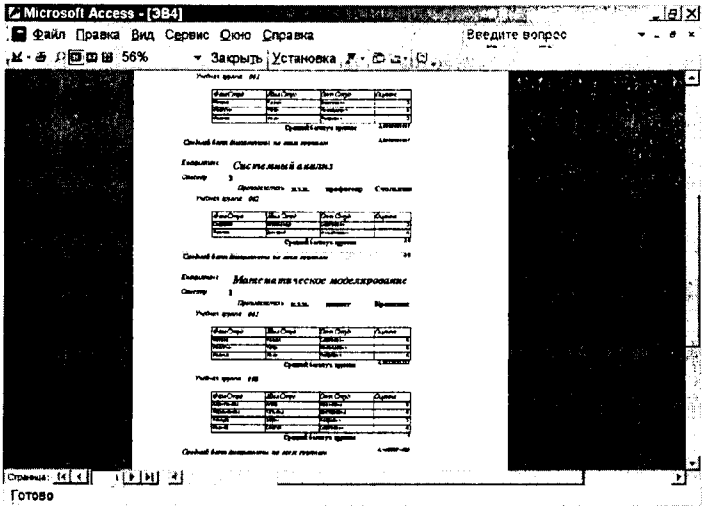


Рис. 4.81

Вид доработанного отчета в режиме ***Конструктора*** (рис. 4.82);

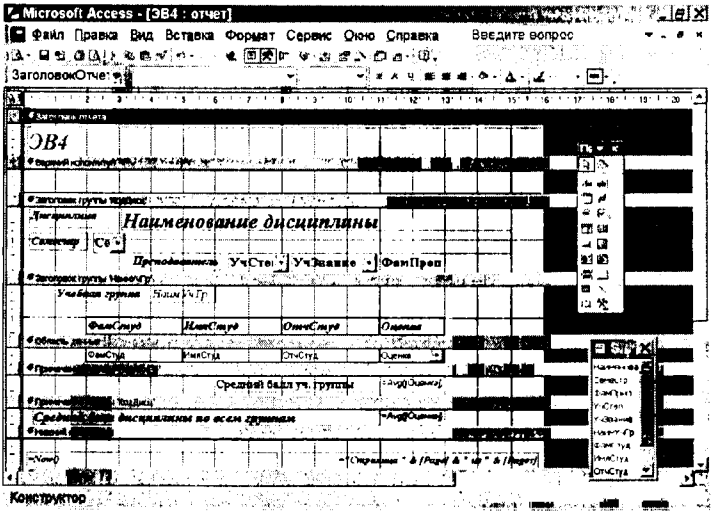


Рис. 4.82

4.6. Интеграция приложений MS OFFICE

Основные способы обмена данными между приложениями Office:

- *статическое копирование/перемещение данных* (вставленные данные становятся неотъемлемой частью документа-получателя и не сохраняют никакой связи с исходным документом);
- *связывание данных* — полная копия данных хранится только в документе-источнике; в документе-получателе находится лишь служебная информация для связывания и данные для отображения этого объекта на экране; при изменении документа-источника связанные данные в документе-получателе обновляются автоматически либо по запросу пользователя;
- *внедрение данных* — данные сохраняют связь с программой-источником, но не с документом; в документе-получателе хранится полная копия внедрённого объекта как при статическом копировании, однако благодаря связи между данными и программой-источником для их редактирования можно воспользоваться средствами программы-источника.

Выйти из режима редактирования. При работе в окне программы-получателя щёлкнуть за пределами объекта. При работе в окне программы-источника выполнить команды **Файл/Выход** или закрыть окно приложения; чтобы сохранить результаты, ответить **Да** на запрос об обновлении объекта.

Статическое копирование/перемещение данных

- Выделить данные в программе-источнике и выполнить команду **Вырезать** (**Копировать**) в меню **Правка**.
- Перейти в программу-получатель и установить курсор в месте вставки данных.
- Выполнить в программе-получателе команды **Правка/Специальная вставка**. На экране появляется диалоговое окно (рис. 4.83).
- Установить переключатель **Вставить**.
- Выбрать из списка **Как** требуемый формат данных.

Связывание

Для связывания используется либо выделенная часть документа, либо весь документ. Действия по передаче со связыванием части документа:

1) выделить данные в программе-источнике и выполнить команду **Копировать** (но не **Вырезать**) программы-источника для копирования данных в буфер обмена;

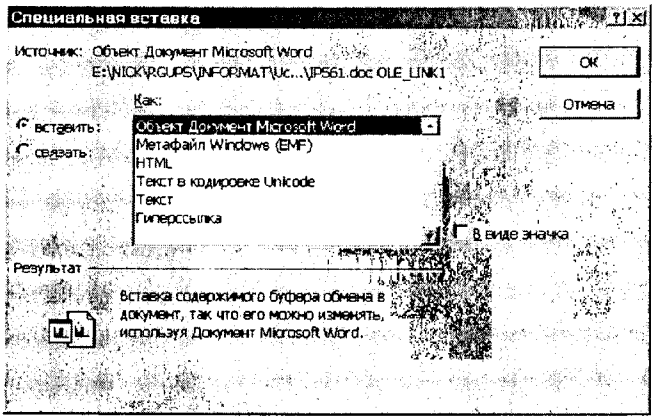


Рис. 4.83

2) подвести курсор к нужному месту документа-получателя и выполнить команды **Вставка/Специальная вставка** в программе-получателе;

3) в окне диалога **Специальная вставка** (рис. 4.83) установить переключатель **Связать**, а затем выбрать нужный формат данных из списка **Как**.

Если для связывания берётся целый документ, можно либо его выделить целиком на этапе 1, либо выполнить следующие действия:

1) установить курсор в нужном месте документа-получателя;

2) выполнить команды **Вставка/Объект** программы-получателя и перейти на вкладку **Из файла** в открывшемся окне диалога **Вставка объекта** (рис. 4.84);

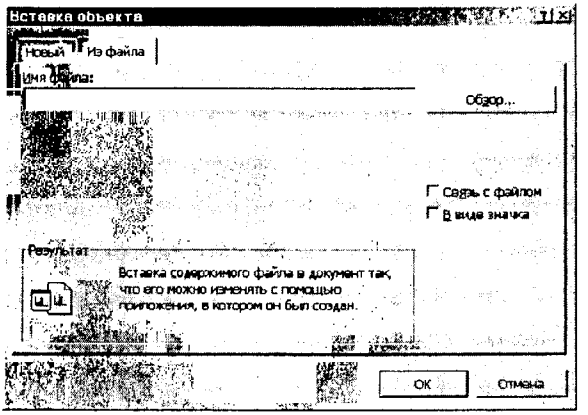


Рис. 4.84

3) установить флажок **Связь с файлом** и ввести в текстовое поле **Имя файла** наименование файла-источника.

Для изменения связанных данных необходимо вносить изменения в документ-источник одним из следующих способов:

1) запустить программу-источник и открыть в ней документ-источник;

2) выделить весь блок связанных данных в документе-получателе или установить внутри него курсор, после чего выполнить команды **Правка/Объект/Документ/Изменить** (**Открыть**, **Преобразовать тип**). В результате запускается программа-источник, в которой открывается документ-источник;

3) при работе с некоторыми форматами связанных данных (например, объект или растр в документе Word) документ-источник в программе-источнике открывается двойным щелчком мыши на связанных данных в документе-получателе.

Для изменения типов вставок в документе служат команды **Правка/Связи**. В результате открывается диалоговое окно (рис. 4.85), отражающее все связи активного документа.

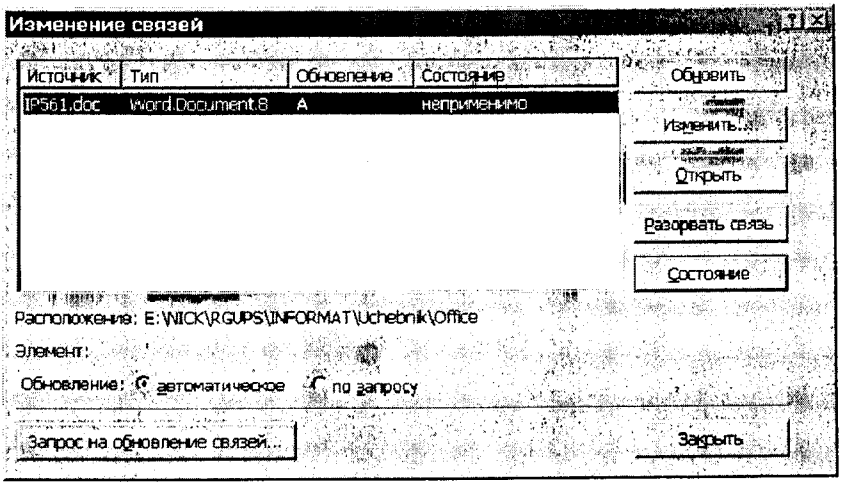


Рис. 4.85

Кнопка **Запрос на обновление связей** открывает окно диалога (рис. 4.86), устанавливающее режим обновления связей.

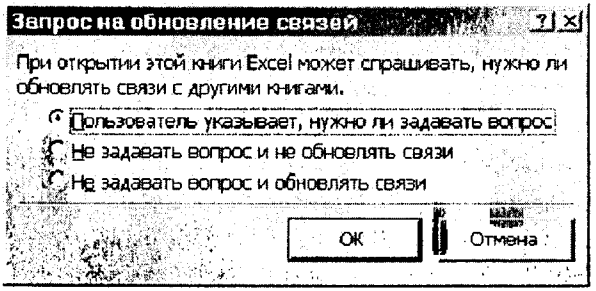


Рис. 4.86

Внедрение

Производится аналогично процедуре связывания с помощью команды **Специальная вставка** (**Вставка/Объект**), но связь с файлом-источником не устанавливается.

Использование дополнительных средств Office

1. Установить курсор в документе-получателе в месте появления внедрённого объекта.

2. Выполнить команды **Вставка/Объект** программы-получателя и перейти на вкладку **Создание** (рис. 4.87).

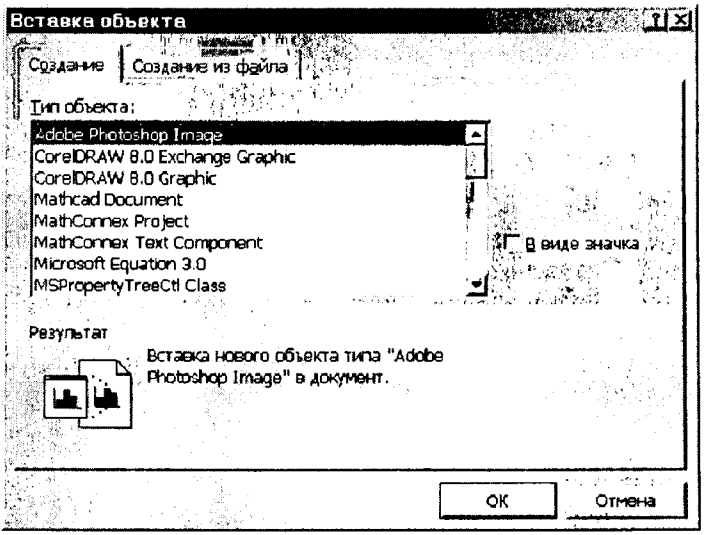


Рис. 4.87

3. Выбрать из списка Тип **объекта** подходящее описание и нажать кнопку ОК. В табл. 4.25 приведены стандартные описания объектов для дополнительных средств Office.

Таблица 4.25

Стандартные описания объектов для дополнительных средств Office

Дополнительное средство	Описание объекта, отображаемое в окне диалога Объект	Назначение программы
Clip Gallery	Microsoft Clip Gallery	Поиск картинок и вставка их в документ
Equation Editor	Microsoft Equation 3.0	Ввод математических выражений в документы
Microsoft Graph	Microsoft Graph	Создание диаграмм для отображения данных
Organization Chart	Microsoft Organization Chart 2.0	Создание организационных и других типов иерархических диаграмм
WordArt	Microsoft WordArt 2.0	Вставка фрагментов текста с применением специальных эффектов

4. Воспользоваться командами программы-источника из состава дополнительных средств для ввода данных внедрённого объекта в предоставленную рабочую область.

4.7. Модели решения функциональных и вычислительных задач средствами MS Office

Количество и многообразие задач, которые могут быть решены с помощью современного компьютера, огромно. Авторы выбрали лишь несколько задач из области математического моделирования, решенных с использованием такого доступного каждому пользователю ПК инструмента, как MS Excel.

В предлагаемых ниже примерах использована команда Excel «Поиск решения».

4.7.1. Задача о ранце

Постановка задачи

В распоряжении лица, принимающего решение, имеется транспортное средство грузоподъёмности Q. В его же распоряжении имеется на-

бор грузов, каждый из которых характеризуется своим весом q_i и стоимостью c_i. Необходимо отобрать для погрузки в имеющееся транспортное средство такие грузы, чтобы грузоподъёмность не была превышена, а стоимость погруженного была бы максимальна.

Математическая постановка задачи

Найти такие значения переменных x_i, при которых достигается максимум функции

$$F = \sum_{i=1}^{n} c_i x_i .$$

И которые удовлетворяют следующим ограничениям:

$$\sum_{i=1}^{n} q_i x_i \le Q, \text{ где } x_i \text{ – целые, } 0 \le x_i \le 1 \text{ и } i = 1...n .$$

Методика выполнения заданий

Рассмотрим выполнение задания на примере. Пусть имеется 6 предметов, веса и стоимости которых известны (ячейки ***B8***:***G8***; ***B9***:***G9*** – см. рис. 4.88). Пусть грузоподъёмность транспортного средства составляет 35 ед. (ячейка ***B10***).

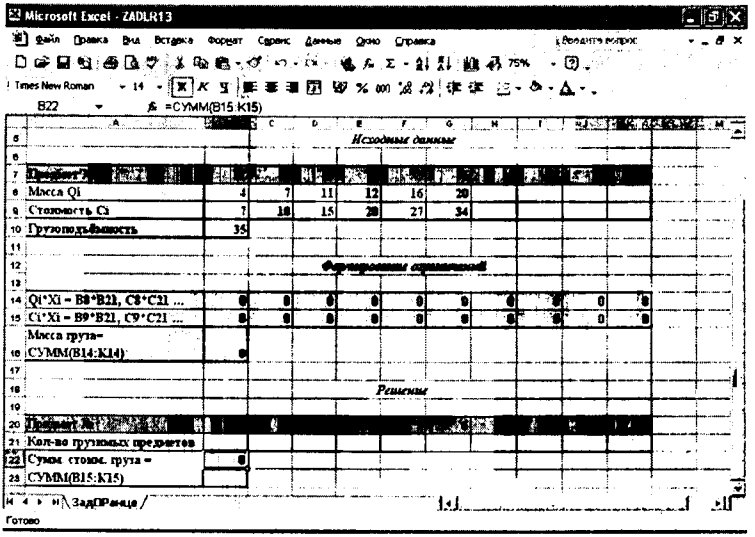


Рис. 4.88

В MS Excel постановка задачи примера формулируется в диалоговом окне команды Поиск решения в соответствии с рис. 4.89.

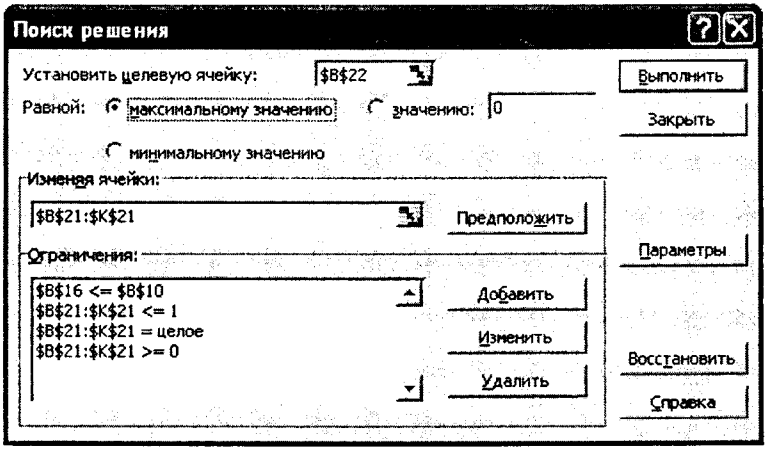


Рис. 4.89

На рис. 4.90 представлен результат поиска решения, в соответствии с которым погрузке подлежат 2-й, 4-й и 5-й предметы (единицы в строке 21).

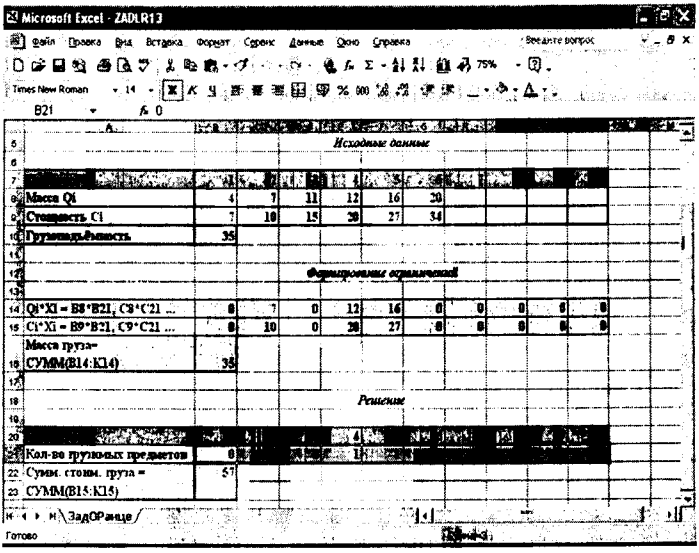


Рис. 4.90

4.7.2. Задача о распределении средств по предприятиям

Постановка задачи

В распоряжении лица, принимающего решение, имеется какой-то запас средств K, который должен быть распределён между n предприятиями $P_1, P_2, \ldots, P_n$. Каждое из предприятий при вложении в него некоторых средств x_i приносит доход, зависящий от x_i, то есть представляет собой неубывающую функцию $u(x_i)$. Эта функция чаще всего нелинейная (с определенного рубежа эффективность вложений в предприятие снижается – предприятие в состоянии «освоить» лишь количество средств, не превышающих какой-то предел). Зачастую неизвестен даже аналитический вид этой функции, а есть данные по эффективности распределения средств в периоды, предшествующие принятию решения. В качестве примера рассмотрим распределение средств по пяти предприятиям при $K=10$ (рис. 4.91). Задача заключается в определении такого плана распределения имеющихся средств, при котором ожидаемый доход будет максимален.

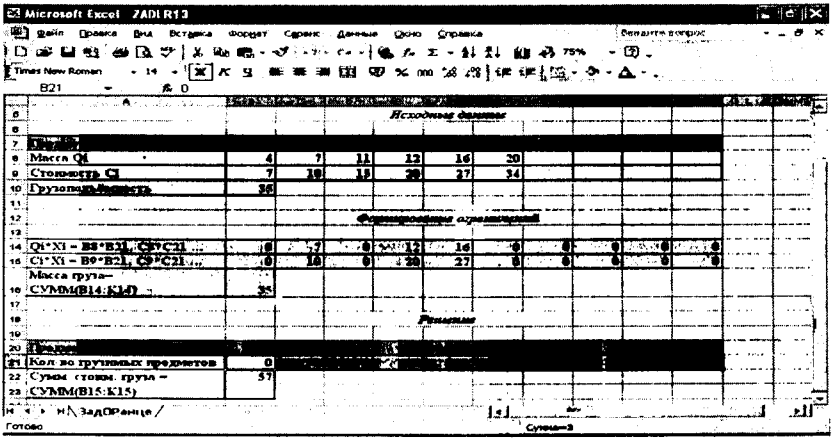


Рис. 4.91

Математическая постановка задачи

Найти такие значения переменных x_i, при которых достигается максимум функции:

$$F = \sum_{i=1}^{n} \varphi_i(x_i).$$

И которые удовлетворяют следующим ограничениям:

$$\sum_{i=1}^{n} x_i \le K, \text{ где } 0 \le x_i \le K,\ i = 1...n.$$

Методика выполнения заданий

Рассмотрим выполнение задания на примере. Пусть имеется 5 предприятий, между которыми требуется распределить с максимальной эффективностью 10 ед. денежных средств (ячейка ***A12*** примера решения).

Вначале необходимо записать исходные данные и подготовить ячейки для хранения промежуточных результатов и результатов решения (рис. 4.92, 4.93, 4.94). Количество ожидаемой прибыли предприятий от вложенных в них средств занесено в ячейки ***A3:F12*** примера решения. В ячейках ***B16:F25***, ***B30:F39*** и ***B53:F53*** содержатся формулы для вычисления промежуточных результатов. Суммарное количество распределённых средств вычисляется в ячейке ***G40***. Ячейки ***B43:F52*** отведены для хранения результатов решения. В ячейке ***F26*** вычисляется суммарная ожидаемая прибыль от всех предприятий в результате распределения средств в соответствии с найденным планом (значениями в ячейках ***B43:F52***).

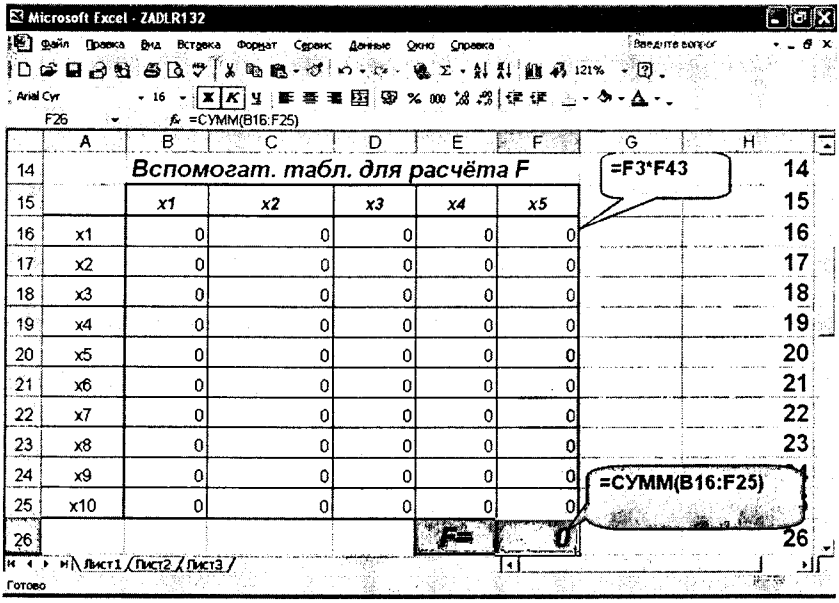


Рис. 4.92

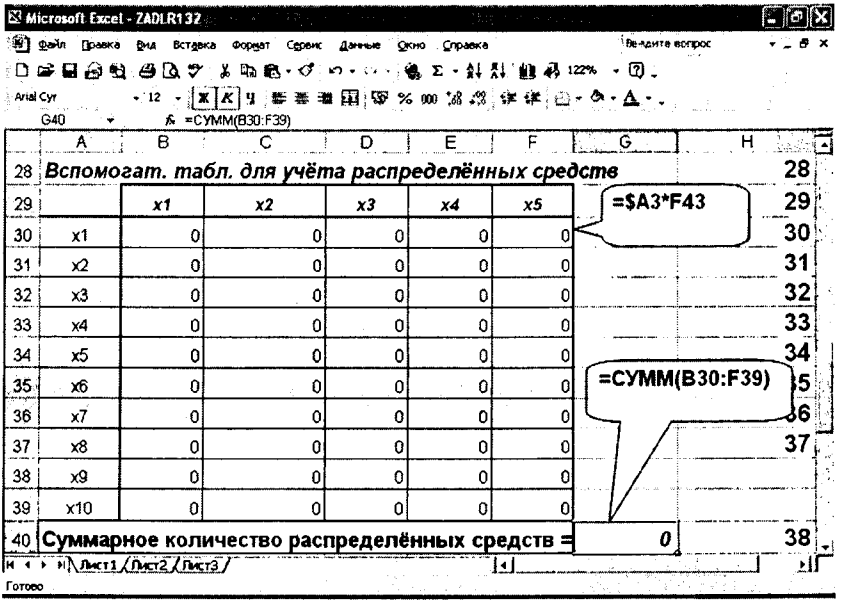


Рис. 4.93

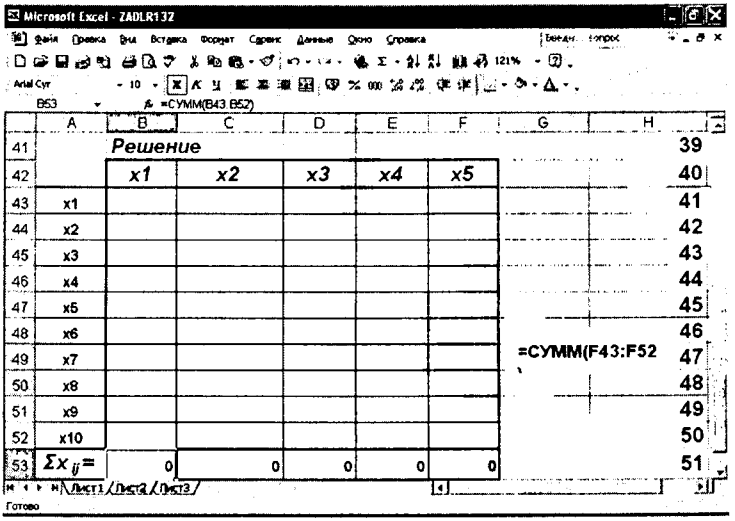


Рис. 4.94

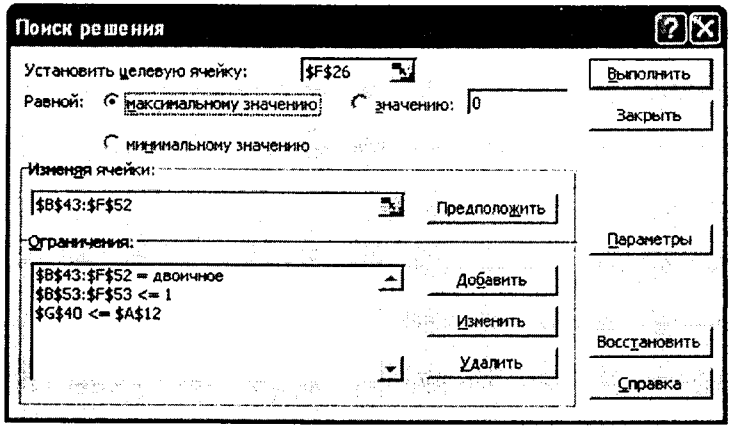


Рис. 4.95

Затем необходимо щёлкнуть левой кнопкой мыши по кнопке ПАРАМЕТРЫ этого диалогового окна и установить в открывшемся диалоговом окне опции в соответствии с рис. 4.96. После возвращения в предыдущее диалоговое окно (рис. 4.95) следует нажать на кнопку ВЫПОЛНИТЬ и из ячеек ***B43:F52*** считать полученное решение.

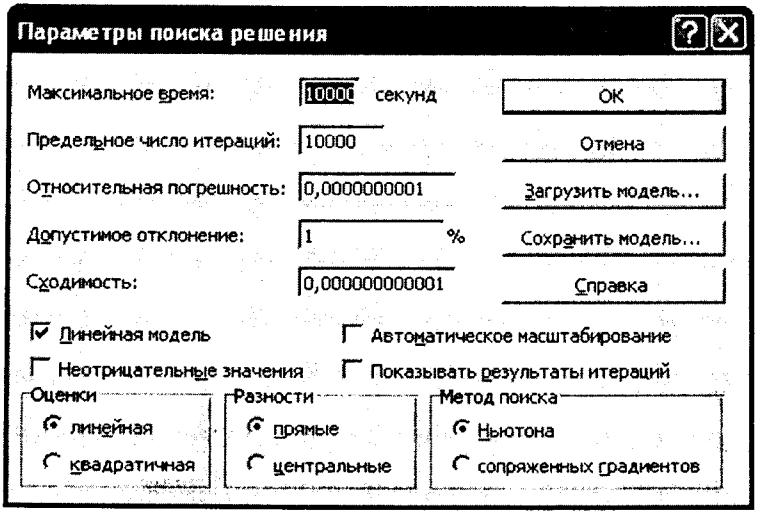


Рис. 4.96

В результате получен один из оптимальных планов распределения средств, который предусматривает направление 7денежных единиц на второе предприятие, 2 единиц – на третье и 1 единицы – на пятое. При этом ожидаемая прибыль максимальна и составляет 5,6 денежных единицы (рис. 4.97, 4.98, 4.99).

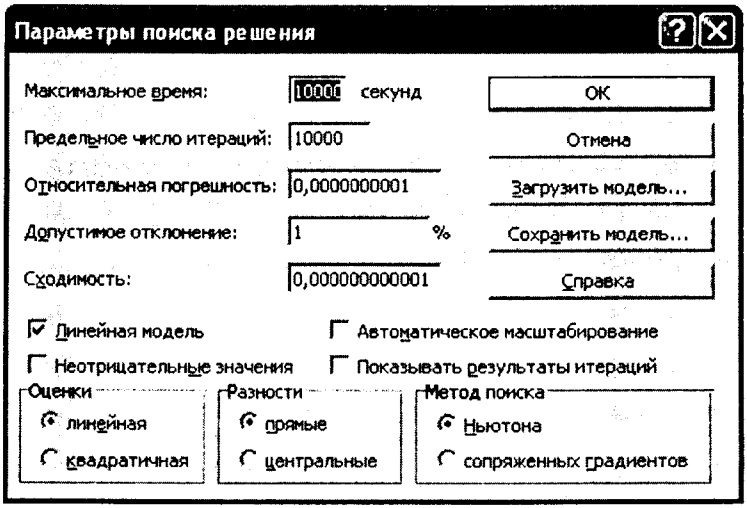


Рис. 4.97

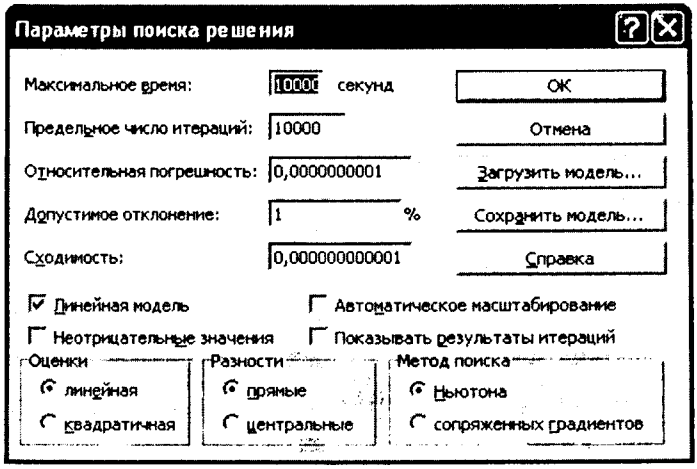
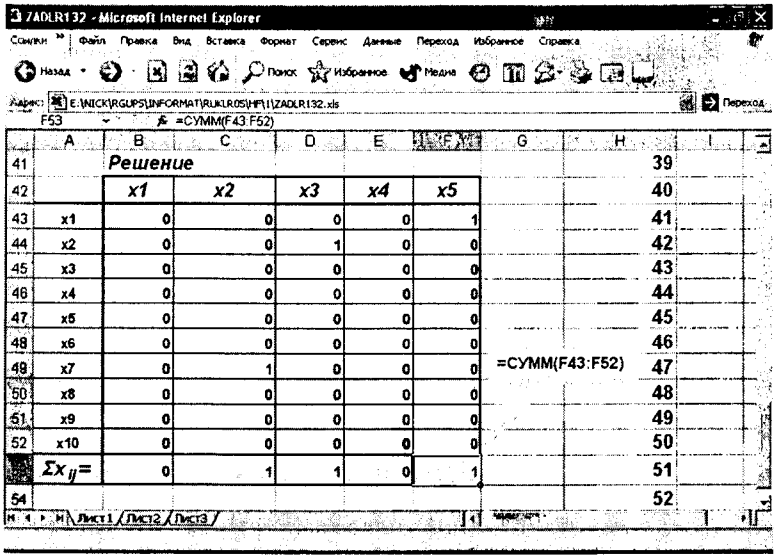


Рис. 4.98



	A	B	C	D	E	F	G	H	I
41		Решение						39	
42		x1	x2	x3	x4	x5		40	
43	x1	0	0	0	0	1		41	
44	x2	0	0	1	0	0		42	
45	x3	0	0	0	0	0		43	
46	x4	0	0	0	0	0		44	
47	x5	0	0	0	0	0		45	
48	x6	0	0	0	0	0		46	
49	x7	0	1	0	0	0	=СУММ(F43:F52)	47	
50	x8	0	0	0	0	0		48	
51	x9	0	0	0	0	0		49	
52	x10	0	0	0	0	0		50	
53	$\Sigma x_{ij}=$	0	1	1	0	1		51	
54								52	

Рис. 4.99

4.7.3. Основная задача линейного программирования (ОЗЛП)

Постановка задачи

Рассмотрим постановку задачи на примере. В ремонтном подразделении заменяется устаревший тип смазки на смазку двух новых типов. На каждый агрегат требуется 3 кг масла 1-го типа либо 4 кг 2-го типа. Ограничение: на весь период ТО выделено 1700 кг масла обоих типов. На обслуживание смазкой 1-го типа требуется 12 мин., смазкой 2-го типа – 30 мин. Общее время на замену масла во время ТО не должно превышать 160 ч. Применение 1-го типа масла взамен устаревшего позволяет сэкономить 2 ден. ед. на каждом агрегате, 2-го типа – 4 ден. ед. Необходимо решить, на каком количестве агрегатов необходимо использовать масло 1-го типа, а на каком – 2-го при проведении ТО для наибольшего экономического эффекта.

Математическая постановка задачи

Найти ***min*** или ***max*** целевой функции:

$$f = c_0 + \sum_{j=1}^{n} c_j x_j .$$

При ограничениях:

$$\sum_{j=1}^{n} a_{ij} x_j \otimes b_i ,$$

где c_j, c_j, a_{ij}, b_i – действительные числа, $x_j \geq 0$, $i = 1 \ldots m$, $j = 1 \ldots n$.

Для рассмотренного примера математическая формулировка задачи может выглядеть следующим образом:

минимизировать функцию $P = -2X_1 - 4X_2$

при ограничениях: $3X_1 + 4X_2 \leq 1700$,

$2X_1 + 5X_2 \leq 1600$,

$X_1, X_2 \geq 0$.

Методика выполнения заданий

В MS Excel постановка задачи примера (рис. 4.100) формулируется следующим образом:

- установить целевую ячейку ***G16,*** равной максимальному значению, изменяя ячейки ***B15:E15***;
- ограничения: ***B15:E15 >= 0; L7<=F7; L8<=F8.***

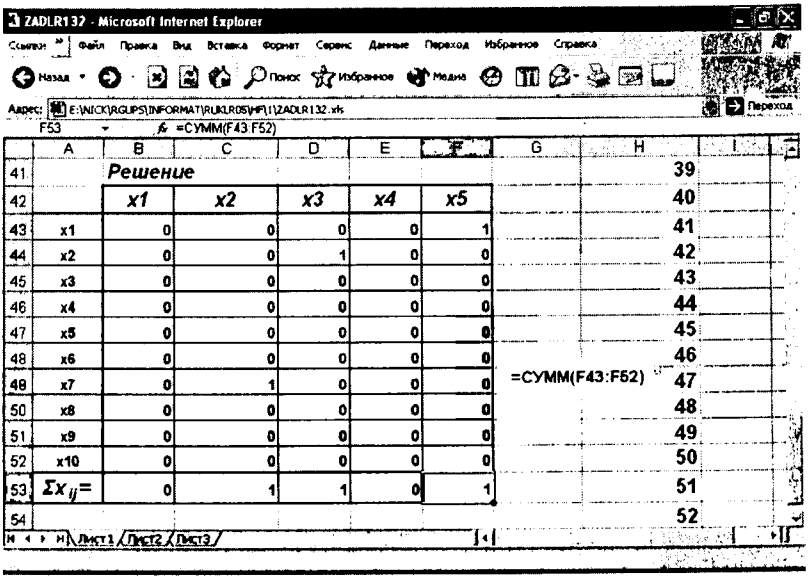


Рис.4.100

4.7.4. Транспортная задача

Постановка задачи

Имеется m исходных пунктов, на которых сосредоточен однородный продукт (уголь на m шахтах, зерно на m элеваторах, вооружение на m складах, информация в m пунктах). Известно количество продукта a_i на каждом пункте $(i=1...m)$. Имеется n конечных пунктов (пунктов назначения), в которые должен быть доставлен продукт в количестве b_j $(j=1...n)$ на каждый пункт. Известна также стоимость c_{ij} доставки ед. груза по маршруту A_i — B_j.

Задача заключается в определении такого плана доставки грузов, при котором либо вывозятся все ресурсы из исходных пунктов, либо удовлетворяются потребности всех пунктов назначения и при котором суммарные затраты на транспортировку минимальны.

Задача может быть представлена в виде графа (рис. 4.101). Рассмотрим линейную транспортную задачу (ЛТЗ) с $m = 2$, $n = 3$.

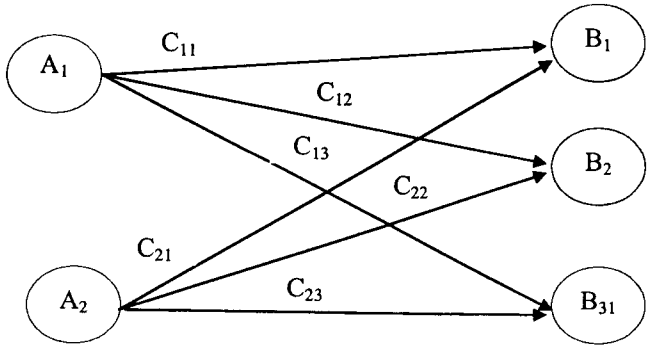


Рис. 4.101

Над стрелками — стоимости, которые обычно зависят от средств транспортирования, расстояния, тарифов, других факторов. Очевидно, что число маршрутов примера = 6, а в общем случае = $m \times n$.

Математическая постановка задачи

Обозначим переменной x_{ij} объем перевозок по маршруту A_i — B_j. Тогда ограничения могут быть сформулированы следующим образом:

• из любого исходного пункта не может быть вывезено больше продукта, чем в нем имеется:

$$\sum_{j=1}^{n} x_{ij} \le a_i \,,\ i=1\ldots m;$$

• в любой пункт назначения не может быть доставлено количество продукта, превышающее потребности этого пункта:

$$\sum_{i=1}^{m} x_{ij} \le b_j \,,\ j=1\ldots n;$$

• перевозки осуществляются только из исходных пунктов в пункты назначения: $x_{ij} \ge 0,\ \ i=1\ldots m,\ \ j=1\ldots n.$

Целевая функция имеет вид $F = \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij}$ и подлежит минимизации.

Методика выполнения задания

Рассмотрим выполнение задания на примере. Пусть имеется 4 исходных пункта со следующими запасами продукта (например, в тоннах): ***a1***=15; ***a2*** =10; ***a3*** =25; ***a4*** =30 (ячейки ***C5***:***C8*** на рис. 4.102); имеется также 6 пунктов назначения с потребностями: ***b1*** =10; ***b2*** =15; ***b3*** =5; ***b4*** =35; ***b5*** =5; ***b6*** =10 (ячейки ***D4***:***I4***). Стоимость перевозки единицы груза по маршруту A_i — B_j представлена матрицей стоимостей (ячейки ***D5***:***I8*** на рис. 4.102).

Ресурсы, Потребности и Матрица стоимостей

			b1	b2	b3	b4	b5	b6	b7	b8
4			10	15	5	35	5	10	0	0
5	a1	15	2	1	3	7	5	4	0	0
6	a2	10	3	2	1	4	6	1	0	0
7	a3	25	4	5	1	2	3	1	0	0
8	a4	30	7	1	2	1	4	4	0	0
9	a5	0	0	0	0	0	0	0	0	0
10	a6	0	0	0	0	0	0	0	0	0
11	a7	0	0	0	0	0	0	0	0	0
12	a8	0	0	0	0	0	0	0	0	0

Рис. 4.102

На рис. 4.103 представлены матрица перевозок и вспомогательная матрица для расчёта целевой функции, а также формулы, внесённые в соответствующие ячейки.

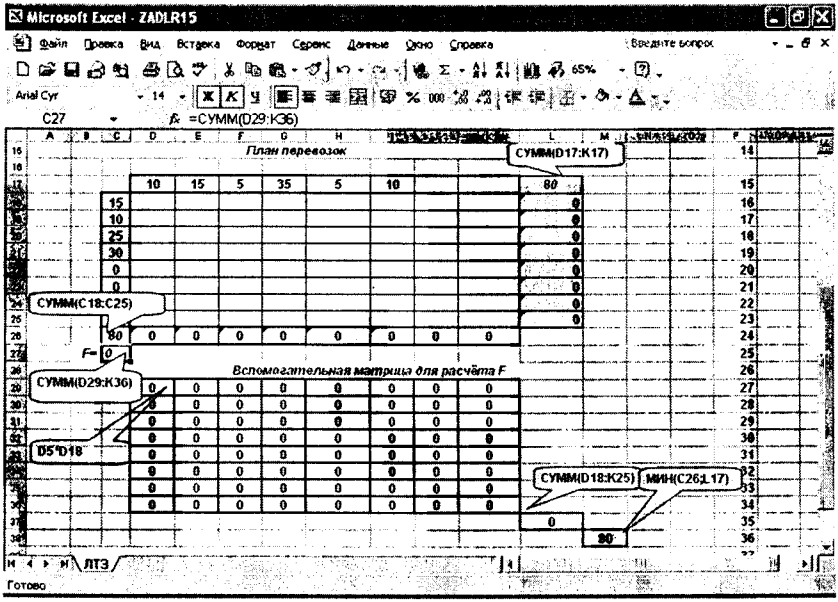


Рис. 4.103

В MS Excel постановка задачи примера (указание на целевую ячейку и формулировка условий) показана на рис. 4.104.

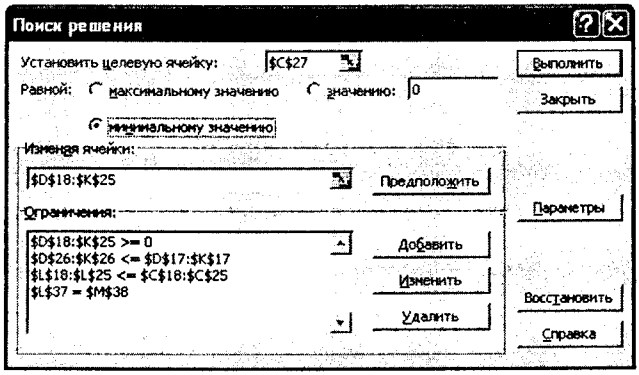


Рис. 4.104

Перед запуском поиска решения необходимо установить в разделе «Параметры» признак «Линейная модель» (рис. 4.105).

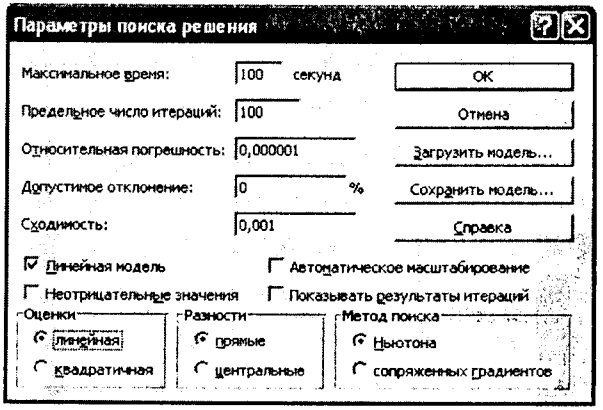


Рис. 4.105

На рис. 4.106 показан результат поиска решения (F=115).

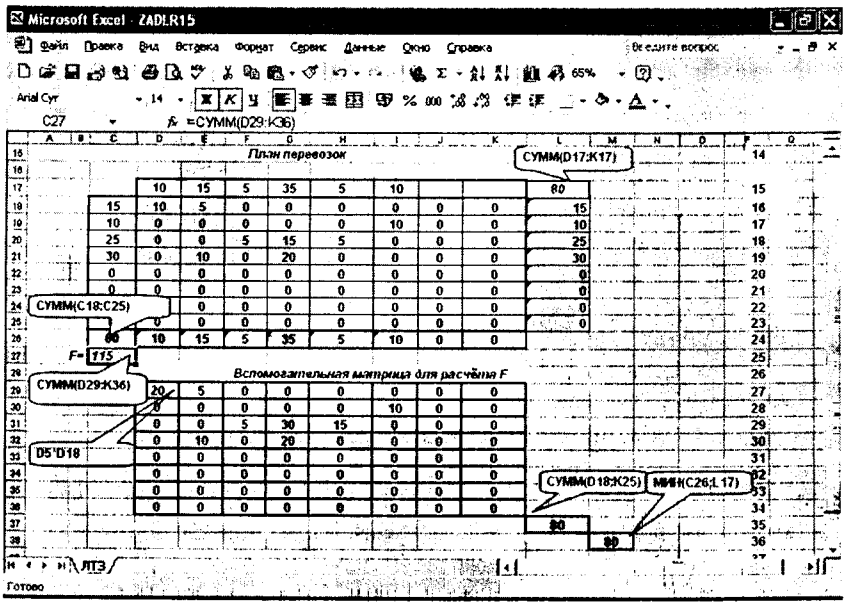


Рис. 4.106

ГЛАВА 5

БАЗЫ ДАННЫХ

5.1. Введение в базы данных

Современный пользователь ПК неизбежно сталкивается с проблемой сохранения и обработки больших массивов данных. В теории их принято называть *банками данных*. Когда осуществляется автоматизация хранения пользовательских банков данных, тогда говорят об их компьютерном представлении в виде *баз данных*. Но база данных сама по себе не может предоставить пользователю удобства хранения и доступа к информации, поэтому более верным является определение *системы управления базами данных* (СУБД). Дадим определения основным понятиям.

▶ *Банк данных* (БнД) – форма организации, хранения и доступа к информации в виде системы специальным образом организованных данных.

▶ *Базы данных* (БД) представляют собой совокупность программных, языковых, технических, организационно-методических средств, предназначенных для обеспечения централизованного накопления и коллективного многоцелевого использования данных.

Отличительные особенности БнД состоят в следующем:

1. Они создаются для многоцелевого использования (т.е. для нескольких пользователей и нескольких программ).

2. Имеется наличие специальных языковых и программных средств, которые облегчают пользователям выполнение таких операций, как хранение данных, корректировка, доступ к данным.

Использование БнД для организации и управления информационными системами имеет следующие *преимущества*:

1. Интегрированное хранение данных в банках обеспечивает непротиворечивость информации, сокращает избыточность хранимых данных. Это приводит к сокращению затрат на создание и хранение данных и поддержания их в актуальном состоянии.

2. Централизованное управление данными. От этих функций освобождаются все пользователи кроме администраторов банка данных.

3. БнД обеспечивают независимость прикладных программ от данных. При создании локальных программ пользователь только обращается в БнД за данными.

4. Работать с базой данных могут любые пользователи, в том числе и непрофессионалы в области обработки данных.

К банкам данных предъявляется ряд *требований*.

1. Полнота, целостность и непротиворечивость данных, актуальность информации.
2. Возможность взаимодействия пользователей разных категорий и в разных режимах.
3. Малое время на освоение системы, особенно для конечных пользователей.
4. Обеспечение секретности для некоторой части данных.
5. Обеспечение взаимной независимости программ и данных.
6. Обеспечение надежности функционирования БнД: защита данных от случайного преднамеренного разрушения, возможность быстрого и полного восстановления данных при разрушении.

5.2. Структура и пользователи банка данных

Состав банка данных (компоненты) представлены на рис. 5.1.

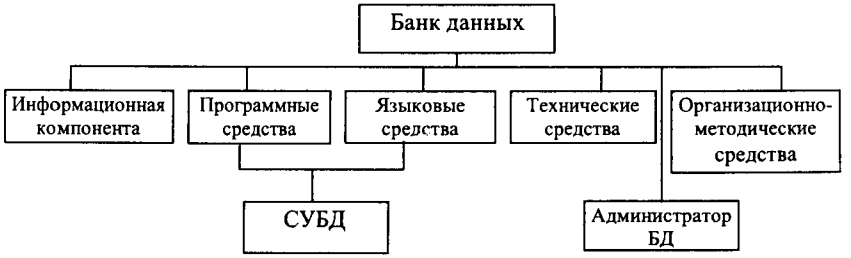


Рис. 5.1

БД включают основные и вспомогательные компоненты.

- К основным компонентам БнД относятся: информационная составляющая, программные средства, языковые средства. Последние два формируют СУБД.
- К вспомогательным компонентам относятся: Организационно-методические средства, технические средства и администратор БнД.

Информационная составляющая определяет ядро БнД – базу данных. Кроме этого, в БнД входят и описания БД – метаинформация (ин-

формация об информации). Описание БД называют *схемой*. Кроме всего, в БнД может храниться информация о пользователях БнД.

Программные средства обеспечивают взаимодействие всех частей системы при функционировании (рис. 5.2).

Рис. 5.2

Основу программных средств составляет СУБД, которая включает:

- *ядро* СУБД, обеспечивающее функции организации ввода, обработки и хранения данных;
- *трансляторы* — средства обеспечения настройки системы и средства тестирования;
- *утилиты* для обеспечения вспомогательных функций.

Почти все СУБД работают в среде универсальных ОС и взаимодействуют с ними, используя их возможности. Прикладные программы служат для обработки запросов в базах данных.

Языковые средства обеспечивают взаимодействие пользователей с БнД и предназначены для пользователей разных категорий: для конечных пользователей, системных аналитиков и профессиональных программистов.

По своим функциональным возможностям выделяют следующие категории языков:

- *языки обеспечения запросов* на вывод и печать данных в нужном виде (в чистом виде такие языки редко используются);
- *комплексные языки запросов* – обновлений (запрос с нескольких файлов, формирования промежуточных своих файлов);
- *генераторы отчетов*, служащие для формирования и вывода в требуемом виде отчетов и других документов по данным БД;

- *графические языки*, позволяющие представить информацию в графической форме (самая удобная форма для анализа), получают в последнее время широкое распространение;
- инструментальные средства *поддержки решений* для создания систем принятия решений и др.

По форме представления языки бывают: аналитические, графические, табличные. В качестве технических средств почти всегда используются универсальные ЭВМ и периферийные средства для ввода информации в БД и отображения выводимой информации. Редко используются специальные технические средства БнД – машины баз данных.

Организационно-методические средства БнД представляют собой различные инструкции, методические и регламентирующие материалы.

Администраторы банка данных (АБД) – это группа лиц (специалистов), обеспечивающих создание, функционирование и развитие БнД.

Среди пользователей БнД можно выделить две группы: конечные пользователи и администраторы банка данных.

К конечным пользователям относятся: основная категория конечных пользователей (для которых БнД создается); случайные пользователи; регулярные пользователи.

Администраторы банков данных включают: проектировщиков технологических процессов обработки данных; системные программисты; прикладные программисты; операторы; специалисты по техническому обслуживанию.

К основным функциям, выполняемым этими группами работников, можно отнести:

1) анализ предметной области (описание, ограничения, статус информации, потребности пользователей и их статус, определение временных характеристик обработки данных);

2) проектирование структуры базы данных (состав и структура БД, связей между ними, методы упорядочения информации);

3) задание ограничений целостности при описании структуры базы данных и процедур обработки БД;

4) первоначальная загрузка и ведение БД (загрузка и ведение БД, формы ввода, контроля и корректировки);

5) защита данных (пароли, тестирование защиты, фиксация попыток несанкционированного доступа);

6) обеспечение восстановления БД;

7) анализ обращений пользователей к БД (сбор статистики обращений пользователя к БД, ее хранение и анализ);

8) анализ эффективности функционирования БнД и развитие системы;

9) работа с пользователями (изменения предметной области БД, оценка пользователями работы БД, обучение и консультации);

10) подготовка и поддержание системных программных средств;

11) организационно-методическая работа (направления развития системы, этапы развития, выпуск различных материалов по БД).

Каждая функция включает:

1) определение границ информации, включаемой в БнД;

2) формы ввода документов, контроля, корректировки их;

3) организация паролей, проверка несанкционированного доступа, защита от разрушения информации;

4) обеспечения средствами восстановления от случайного разрушения баз;

5) сбор статистики обращения пользователей к БД, ее хранение и т. д.

6) определение показателя эффективности, прибыльности, перспектив развития системы;

7) обеспечивает изменение предметной области для пользователей, оценка полезности БнД, проведение консультаций, обучение;

8) покупки новых СУБД и их внедрение, новых пакетов прикладных программ;

9) планирование развития системы, определение этапов развития, выпуск методических и других материалов по работе с БнД и т. д.

К основным активным связям – взаимодействиям в работе по созданию и поддержанию БнД можно отнести связи администратора с руководителями предприятия (перспективы развития предприятия, требования к БнД, отзыв о работе БнД, динамичность развития, очередность подключения конечных пользователей, уточнение особенностей и др.), с внешними по отношению к нему группами специалистов и, прежде всего, с поставщиками СУБД и прикладными программными продуктами, администраторами других БнД.

5.3. Классификация банков и баз данных

Центральным компонентом банка данных является база данных, и большинство классификационных признаков относится именно к ней.

По *форме представления* информации БнД делятся на: видеосистемы; аудиосистемы; мультимедиа.

БД, содержащие *символьные данные,* могут быть: неструктурированные; частично структурированные; структурированные.

Структурированные БД по *типу используемой модели* делятся на: иерархические; сетевые; реляционные (базы данных на основе двумерных таблиц); смешанные; мультимодельные, объектно-ориентированные.

По *типу хранимой информации* БД бывают: документальные; фактографические; лексикографические.

Документальные в свою очередь подразделяются на: библиографические; реферативные; полнотекстовые.

По *характеру организации хранения данных* и *обращения* к ним БД принято делить на локальные; общие; распределенные.

БД могут классифицироваться *по охвату предметной области* по разным признакам. Например, по территориальному (всемирная, страна, город и т.д.); по временному (год, месяц, с начала века и т. д.); по ведомственному, по проблемному.

Различают также *экстенсиональные* и *интенсиональные* БД (ЭБД и ИБД). ЭБД — это просто реляционная база данных. ИБД строится из ЭБД с помощью правил, определяющих ее содержание, а не с помощью явного хранения записей в таблицах.

По *языкам общения* СУБД классифицируют на: открытые; замкнутые; смешанные.

Открытые СУБД построены с помощью языков высокого уровня.

Закрытые СУБД имеют собственные языки общения с пользователем.

По *числу уровней в архитектуре* СУБД делятся на: одноуровневые; двухуровневые; трехуровневые.

Под *архитектурным уровнем* СУБД понимают функциональный компонент, механизмы которого служат для поддержки некоторого уровня абстракции данных (логический и физический уровень, а также «взгляд» пользователя на внешний уровень).

СУБД классифицируют также и по *выполняемым функциям*: на информационные и операционные, и по *сфере возможного применения*: на универсальные и специализированные.

СУБД поддерживают разные типы данных. Набор типов данных, допустимых в разных типах СУБД, различен. В настоящее время наблюдается тенденция к расширению числа используемых типов данных. Кроме того, ряд СУБД позволяет разработчику добавлять новые типы данных и новые операции над этими данными. Такие системы называются расширяемыми системами баз данных (РСБД). Дальнейшим развитием концепции РСБД являются объектно-ориентированные системы баз данных, обладающие достаточно мощными выразительными возможностями, чтобы непосредственно моделировать сложные объекты.

Новым направлением в развитии программного обеспечения БнД являются *генераторы систем баз данных*. Они позволяют разработчику строить собственную СУБД нового типа без полного переписывания программного кода. К таким средствам можно отнести возможности, например, систем программирования *Borland Delphi*, *Oracle* и др.

5.4. Этапы проектирования баз данных

В базе данных всегда отражается информация об определенной предметной области.

▶ *Предметной областью* называется часть реального мира, представляющая интерес для данного исследования (использования).

Процесс проектирования БД представляет собой сложный процесс проектирования отображения: «Описание предметной области» <=> «Схема внутренней модели базы данных».

Этот процесс представляют последовательностью более простых, обычно итеративных, процессов проектирования менее сложных отображений между промежуточными моделями данных, т.е. последовательностью моделей уровней абстрагирования. Независимо от того, поддерживаются ли явно модели каждого из уровней методологически, необходимо выделить для каждого уровня модели физического и логического уровней абстрагирования.

Основные этапы проектирования подразделяются на два больших этапа:

Инфологическая модель предметной области (ИЛМ);

Датологическая модель данных (ДЛМ).

▶ *Инфологической моделью* предметной области называется описание предметной области, выполненное без ориентации на используемые в дальнейшем программные и технические средства.

На уровне ИЛМ осуществляется анализ информации о предметной области и ее описание. Следует отметить, что эта информация мало зависит от выбранной СУБД.

▶ *Датологическая модель* является моделью логического уровня и представляет собой отображение логических связей между элементами данных безотносительно к их содержанию и среде хранения. Эта модель строится в терминах информационных единиц, допустимых в той конкретной СУБД, в среде которой мы проектируем БД. ДЛМ подразделяется, в свою очередь, на физический и логический уровни. Этап создания ДЛМ называется *датологическим проектированием*.

Описание логической структуры БД на языке СУБД называется *схемой*. Для привязки ДЛМ к среде хранения используется модель данных физического уровня (для краткости часто называемая *физической моделью*). Модель физического уровня также строится с учетом возможностей, предоставляемых СУБД.

Описание физической структуры БД называется *схемой хранения*. Соответствующий этап проектирования БД называется *физическим проектированием*. К числу работ физического уровня относятся:

- выбор типа носителя,
- выбор способа организации данных,
- выбор методов доступа,
- определение размера физического блока,
- управление размещением данных на внешнем носителе,
- управление свободной памятью,
- определение целесообразности сжатия данных,
- определение методов сжатия данных,
- оценка физической модели данных.

К физическому проектированию относятся и проблемы, связанные с буферизацией (определение числа и размера буферов, используемых при передаче данных из внешней памяти во внутреннюю, закрепление файлов за буферами).

В настоящее время наблюдается тенденция к сокращению работ на стадии физического проектирования, так как в большинстве случаев современные СУБД предоставляют средства, называемые *конструкторами* (*Microsoft Access*). Иногда эти работы вообще бывают скрыты от проектировщика.

В некоторых СУБД, помимо описания общей логической структуры БД, имеется возможность описать логическую структуру БД с точки зрения конкретного пользователя. Такая модель называется *внешней*, а ее описание называется *подсхемой*. Если СУБД «поддерживает» схему хранения и подсхему, то она является СУБД с *трехуровневой архитектурой* и состоит из: схемы-ДЛМ данных; схемы строения; подсхемы-описания хранения базы данных с точки зрения пользователя. Если СУБД поддерживает уровень подсхем, то перед проектировщиком встает задача их определения. Это тоже можно рассматривать как этап проектирования БД. Одну и ту же БД можно разработать для нескольких пользователей.

Внешняя модель не всегда является точным подмножеством схемы. Некоторые СУБД допускают различия в типах данных, определенных в схеме и подсхеме, и обеспечивают их преобразование, допускаются раз-

личный логический порядок следования элементов в схеме и подсхеме, введение в подсхему виртуальных полей и т.д. Если определена подсхема, то пользователь имеет доступ только к тем данным, которые отражены в соответствующей подсхеме, что является одним из способов защиты информации от несанкционированного доступа. В подсхемах часто задаются и допустимые режимы обработки, что является дополнительным механизмом защиты информации от разрушения.

Использование аппарата подсхем облегчает работу пользователя, т.к. он должен знать структуру не всей БД, а только той ее части, которая имеет непосредственное отношение к нему; кроме того, эта структура приспособлена к его потребностям. В тех случаях, когда СУБД в явном виде не поддерживает подсхемы, перечисленные функции могут выполнять другие компоненты системы. Близким к понятию подсхемы является понятие «взгляд» (*view*), которое в настоящее время широко используется в англоязычной литературе по реляционным СУБД. В качестве общей схемы взаимодействия этапов можно принять следующую схему (рис. 5.3).

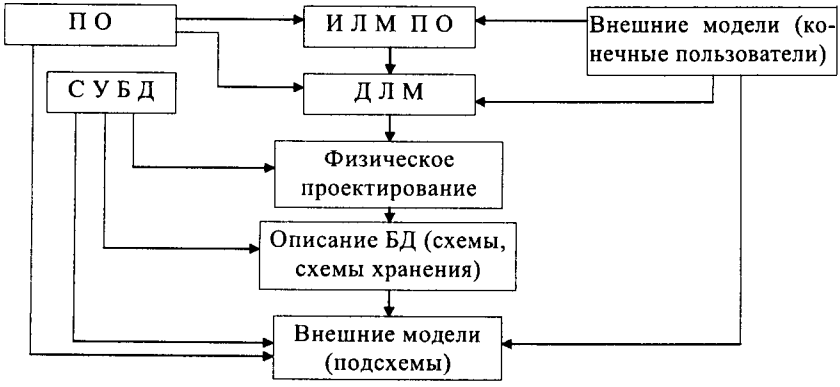


Рис. 5.3

При проектировании БД возможен возврат на предыдущие уровни. При этом возможны два вида возвратов: первый тип обусловлен необходимостью пересмотра результата проектирования (например, для улучшения полученных характеристик, обхода ограничений и т.п.), второй тип вызван необходимостью уточнения предыдущей модели (обычно инфологической) с целью получения дополнительной информации для проектирования или при выявлении противоречий в модели.

5.4.1. Инфологическое моделирование

Для отображения предметной области при ИЛМ необходимо, чтобы описание не зависело от исследователя. Хотя не возбраняется для этих целей использовать естественный язык, рекомендуется все же применять искусственные формализованные языковые средства, т.е. ИЛМ должна строится вне зависимости от той СУБД, которая будет в дальнейшем использоваться для работы с БД. Таким образом, требования, предъявляемые к инфологической модели, относятся также к языку.

Основные требования к ИЛМ являются:

- адекватное отображение предметной области (философское требование);
- язык для представления ИЛМ должен быть выразительным,
- отражение интересов всех пользователей;
- не должны допускать двоякую трактовку;
- обладать свойством легкой расширяемости;
- возможностями удаления данных;
- возможности композиции и декомпозиции.

Требования к языку описания ИЛМ заключаются в вычисляемости, обеспечении дружественного интерфейса, независимости от технического обеспечения, использовании средств тестирования ИЛМ, а также средств генерации БД после завершения построения ИЛМ.

К дополнительным требования ИЛМ можно отнести:

- автоматически изменение БД при изменении ИЛМ,
- читабельность ИЛМ разными категориями пользователей и проектировщиков,
- однозначность восприятия ИЛМ всеми категориями специалистов.

К компонентам ИЛМ относятся (рис. 5.4):

- Описание объектов и связи между ними (ER-модель, *Entity-Relashionship*).

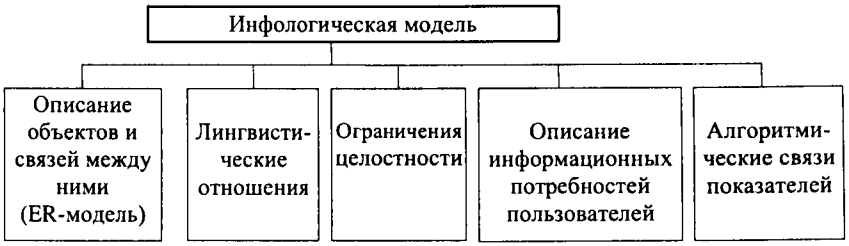


Рис. 5.4

- Лингвистические отношения.
- Ограничение условности.
- Описание информационных потребностей пользователей.
- Алгоритмические связи пользователей.

Этапы построение модели «объект-свойство-отношение» рассмотрим на примере предметной области «высшее учебное заведение», вуз.

Нельзя сказать, что в настоящее время существует какой-либо стандарт или хотя бы общепринятый способ построения инфологической модели. Для описания ИЛМ используются как языки аналитического (описательного) типа, так и графические средства. Графические средства в последнее время приобретают все большую популярность, связанное с наглядностью и простотой восприятия и анализа. Далее для отображения модели «объект – свойство – отношение» воспользуемся графическим способом. В предметной области выделяются классы объектов. *Классом* объектов называют совокупность объектов, обладающих одинаковым набором свойств. Например, для предметной области ВУЗ можно определить классы объектов: учащиеся, преподаватели, аудитории, дисциплины и т.д. Объекты могут быть реальными или абстрактными («дисциплины»).

При отражении в информационной системе каждый объект представляется своим идентификатором, который отличает один объект класса от другого, а каждый класс объектов представляется именем этого класса.

Каждый объект обладает определенным набором свойств. Для объектов набор этих свойств одинаков, а их значения, естественно, могут отличаться. Так для объектов класса «СТУДЕНТ» таким набором свойств может быть «ГОД РОЖДЕНИЯ», «ПОЛ» и др.

При построении ИЛМ объект изображается:

а свойство изображается:

Каждому классу объектов ИЛМ присваивается уникальное имя. Кроме этого может использоваться его короткое кодовое обозначение.

Связь между объектом и его свойством может быть различной.

1. *Единичное свойство*, когда объект обладает одним из значений свойства (например: год рождения). Для изображения используется одинарная стрелка: →

2. *Множественное свойство*, когда одному объекту соответствует более одного значения свойства (например: сотрудник – иностранный язык). Изображение: ↔

3. По постоянству во времени свойства подразделяются на *статистические* (год рождения, место рождения) и *динамические* (образование).

4. Некоторые свойства могут носить *условный характер*, т.е. когда одним объектам они присущи, а у других могут отсутствовать (ученая степень, иностранный язык, воинская обязанность).

5. В ИЛМ можно определить *составное свойство*, которое определяется через другие свойства (Адрес = Город + Улица + Дом + Квартира; Дата рождения = День + Месяц + Год).

Кроме связи между объектами и некоторыми свойствами в ИЛМ фиксируются связи между объектами разных классов.

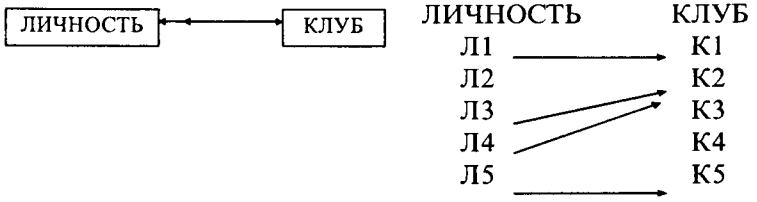
Различают связи типа:

1 : 1	(один к одному) обозначение	⟷
М : 1	(многие к одному) обозначение	⟵⟷
1 : М	(один ко многим) обозначение	⟷⟶
М : М	(многие ко многим) обозначение	⟵⟷⟶

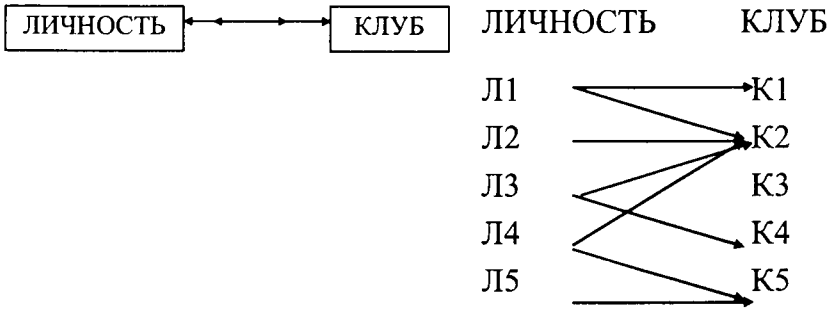
Кроме степени связи в ИЛМ для характеристики связи между разными сущностями надо указывать так называемый *класс принадлежности*, который устанавливает возможность связи объекта данного класса с каким-либо объектом другого класса (не для всех обязательная).

Отношения принадлежности рассмотрим на примере между «ЛИЧНОСТЬ» (некоторый индивидуум) и «КЛУБ» (общество, клуб, ассоциация, партия и т.п.).

1. Пусть каждая «ЛИЧНОСТЬ» может входить не более чем в один «КЛУБ» (партию). В этом случае мы получим отношение М:1. Заметим, в этом случае не каждая личность имеет связь с клубом (партией), а каждый клуб (партия) может иметь более чем одного члена.



2. Пусть теперь каждая «ЛИЧНОСТЬ» может входить в любое число «КЛУБОВ» (общества). В этом случае мы получим отношение М:М.



Объекты подразделяются на следующие классы:

1. *Простые объекты*, объекты которые не допускают дальнейшей детализации в данной предметной области (например, «День», «Сумма», «Фамилия»).

2. *Сложные объекты,* объекты связанные отношением «целое–часть» («Дата – День», «Оплата – Сумма», «Ф.И.О. – Фамилия»).

3. *Обобщенные объекты*, которые описываются отношением «род – вид» (учащиеся: школьники, студенты, аспиранты; сотрудники: руководители, рабочие, ученики).

4. *Агрегированные объекты* – объединение объектов и отношений между ними («Экзамен»: «Дисциплина», «Дата», «Время», «Форма оценки», «Экзаменатор»). Агрегированные объекты обозначаются в виде ромба.

При построении *ER*-моделей можно обнаружить различия в изображении объектов, свойств и связей, различия, связанные с графическим изображением отношений между объектами. Если связь (отношение) для некоторых объектов отсутствует или может отсутствовать, то она обозначается пунктирной линией.

5.4.2. Датологическое моделирование

Исходными данными для ДЛМ служат:

1. Инфологическая модель описания предметной области.
2. Характеристики СУБД для построения ДЛМ:
   - формы представления информационных единиц;
   - формы отображения отношений между ними;
   - методики проектирования на СУБД и их особенности;

- возможности автоматизации проектирования;
- ограничения на количественные характеристики СУБД (длина записи, число полей записи).

Результатом ДЛМ-проектирования является язык описания данных логической структуры БД, приведенный в ИЛМ. При построении ДЛМ надо обращать внимание на соответствие логической структуры, представленной в ИЛМ, полученному результату.

БД включает всю информацию о предметной области, которая вводится и вычисляется (хранится) с учетом требований — предметной области. При этом возможно два различных подхода к построению БД: 1) хранение только исходной информации, а получение выходных форм по мере необходимости к непосредственным расчетам; 2) хранение полностью всей информации.

Первый подход является более популярным потому, что существует однозначность восприятия и представления предметной области. Дублирование информации (экономия места, отсутствие разночтений) в таком случае отсутствует. Имеется возможность получить любую информацию на основе расчета.

Особенностями ДЛМ является то, что информация хранится в файлах БД.

Различают следующие виды особенностей ДЛМ:

1. Особенности *внутризаписной* структуры:
   - линейная – информация располагается в строчку, поля располагаются одно за другой и находятся в одинаковом соотношении друг к другу;
   - иерархическая – присутствуют составные объекты.
2. Особенности по структуре записи:
   - с постоянным составом – все записи содержат одинаковое количество объектов;
   - с переменным составом – записи имеют отличные от других компоненты, или не содержат компоненты, содержащиеся в других. Например, запись с вариантами в языке программирования Паскаль.
3. Особенности по длине записи:
   - с фиксированной длиной;
   - с переменной длиной;
   - с неопределенной длиной.

5.5. Проектирование реляционных баз данных

Датологическое проектирование определяет логическую структуру БД с использованием конкретной СУБД.

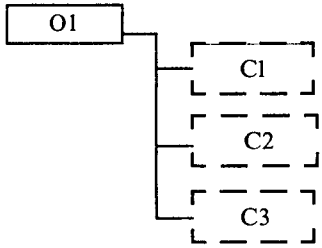
Рассмотрим СУБД реляционного типа.

В зависимости от ИЛМ, форм связей между объектами, форм связей объектов и его свойств могут получиться один или несколько файлов БД от одной конструкции.

Объекты (О) будем обозначать прямоугольниками со сплошными границами. Свойства объектов (С) или *атрибуты* будем обозначать в виде прямоугольников со штриховыми границами. Связи между объектами будем обозначать в виде различных стрелок.

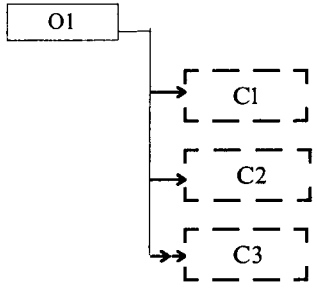
Возможны несколько конструкций.

Конструкция 1-го типа:



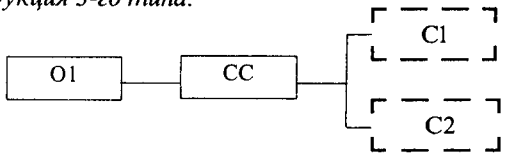
Для каждого единичного объекта с его единичными свойствами строится файл с идентификатором объекта и полями для отображения его единичных свойств.

Конструкция 2-го типа:



Объект имеет единичные и множественные свойства. В этом случае отдельно создается файл базы БД для единичных свойств объекта и отдельно для каждого множественного свойства по одному файлу.

Конструкция 3-го типа:



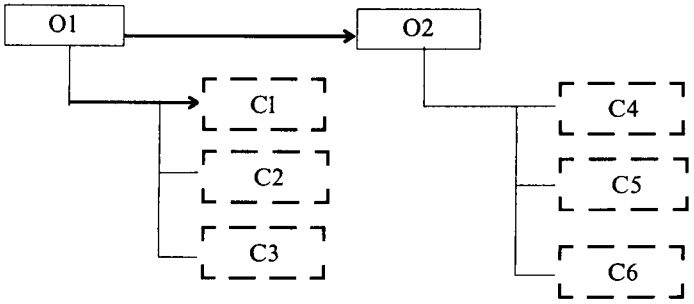
Для хранения составного свойства CC используется один из вариантов:

1) файл с идентификатором объекта и с его элементарными (простыми) свойствами;

2) файл с одним составным свойством;

Принятие того или иного вида хранения информации зависит от конкретной ситуации.

Конструкция 4-го типа:



Тогда два объекта O1 и O2 связаны между собой и у каждого есть свойство, то хранение осуществляется следующими вариантами:

1) ключевым идентификатором будет первый объект, затем второй объект и его свойство;

2) второй объект и его свойство можно выразить через первый объект.

ГЛАВА 6

АЛГОРИТМИЗАЦИЯ И ПРОГРАММИРОВАНИЕ

6.1. Основные этапы решения задач на ЭВМ

Процесс решения задач на компьютере — это совместная деятельность человека и ЭВМ. На долю человека приходятся этапы, связанные с творческой деятельностью — постановкой, алгоритмизацией, программированием задач и анализом результатов, а на долю персонального компьютера — этапы обработки информации в соответствии с разработанным алгоритмом.

Первый этап — *постановка задачи.* На этом этапе участвует человек, хорошо представляющий предметную область задачи (биолог, экономист, инженер). Он должен чётко определить цель задачи, дать словесное описание содержания задачи и предложить общий подход к её решению.

Второй этап — *выбор метода решения* (*математическое или информационное моделирование*). Цель данного этапа — создать такую математическую модель решаемой задачи, которая могла быть реализована в компьютере. Существует целый ряд задач, где математическая постановка сводится к простому перечислению формул и логических условий.

Этот этап тесно связан с первым, и его можно отдельно не рассматривать. Однако возможно, что для полученной модели известны несколько методов решения, и необходимо выбрать лучший. Заметим, что появление средств визуального моделирования объектов позволяет в некоторых случаях освободить программиста от выполнения данного этапа.

Третий этап — *алгоритмизация задачи.* На основе математического описания необходимо разработать алгоритм решения.

▶ *Алгоритм* — система точных и понятных предписаний о содержании и последовательности выполнения конечного числа действий, необходимых для решения любой задачи данного типа (класса).

Понятие алгоритма возникло и используется давно. Сам термин «алгоритм» ведёт начало от перевода на европейские языки имени арабского математика Аль-Хорезми (IX век). Им были описаны правила (в нашем понимании — алгоритмы) выполнения основных арифметических действий в десятичной системе счисления.

Задача составления алгоритма не имеет смысла, если не известны или не учитываются возможности его исполнителя (ребёнок может прочесть, но не может решить сложную задачу).

Исполнителем может быть не только человек, но и автомат. Компьютер — лишь частный, но наиболее впечатляющий пример исполнителя, чьё поведение основано на реализации алгоритма. Более того, создание персонального компьютера оказало воздействие на развитие теории алгоритмов, одной из областей дискретной математики.

Эффективный метод построения алгоритма — *метод пошаговой детализации* (последовательного построения). При этом сложная задача разбивается на ряд более простых. Для каждой подзадачи разрабатывается свой алгоритм. Универсальный эффективный метод построения алгоритма является основой структурного программирования (см. п. 6.16).

Если алгоритм разработан, то его можно вручить разным людям (пусть и не знакомым с сутью решаемой задачи), и они, следуя системе правил, будут действовать одинаково и получат (при безошибочных действиях) одинаковый результат.

Используются различные способы записи алгоритмов:

- словесный (запись рецептов в кулинарной книге, инструкции по использованию технических устройств...);
- графический — в виде блок-схемы;
- структурно-стилизованный (для записи используется язык псевдокода).

При составлении и записи алгоритма необходимо обеспечить, чтобы он обладал рядом свойств:

Однозначность алгоритма — единственность толкования исполнителем правил выполнения действий и порядка их выполнения. Чтобы алгоритм обладал этим свойством, он должен быть записан командами из системы команд исполнителя.

Конечность алгоритма — обязательность завершения каждого из действий, составляющих алгоритм, и завершимость алгоритма в целом.

Результативность алгоритма — предполагает, что выполнение алгоритма должно завершиться получением определённых результатов.

Массовость — возможность применения данного алгоритма для решения целого класса задач, отвечающих общей постановке задачи.

Правильность алгоритма — способность алгоритма давать правильные результаты решения поставленных задач.

Четвёртый этап — *программирование.* Программой называется план действий, подлежащих выполнению некоторым исполнителем, в каче-

стве которого может выступать компьютер. Программа позволяет реализовать разработанный алгоритм.

Пятый этап — *ввод программы и исходных данных в ЭВМ* с клавиатуры с помощью редактора текстов. Для постоянного хранения осуществляется их запись на гибкий или жёсткий диск.

Шестой этап — *тестирование и отладка программы.* Исполнение алгоритма с помощью ЭВМ, поиск и исключение ошибок. При этом программисту приходится выполнять рутинную работу по проверке работы программы, поиску и исключению ошибок, и поэтому для сложных программ этот этап часто требует гораздо больше времени и сил, чем написание первоначального текста программы.

Отладка программы — сложный и нестандартный процесс, который заключается в том, чтобы протестировать программу на контрольных примерах.

Контрольные примеры стремятся выбрать так, чтобы при работе с ними программа прошла все основные пути алгоритма, поскольку на каждом из путей могут встретиться свои ошибки, а детализация плана зависит от того, как поведёт себя программа на этих примерах. На одном она может «зациклиться», на другом — дать бессмысленный результат.

Сложные программы отлаживают отдельными фрагментами.

Для повышения качества выполнения этого этапа используются специальные программы — отладчики, которые позволяют исполнить программу «по шагам» с наблюдением за изменением значений переменных, выражений и других объектов программы с отслеживанием выполнения операторов.

Седьмой этап — *исполнение отлаженной программы и анализ результатов.* На этом этапе программист запускает программу и задаёт исходные данные, требуемые по условию задачи.

Полученные результаты анализируются постановщиком задачи, и на основании этого анализа вырабатываются соответствующие решения, рекомендации, выводы.

Языки программирования

Чтобы компьютер выполнил решение какой-либо задачи, ему необходимо получить от человека инструкции, как её решать. Набор таких инструкций для компьютера, направленный на решение конкретной задачи, называется компьютерной программой.

Современные компьютеры не настолько совершенны, чтобы понимать программы, написанные на каком-либо употребляемом человеком языке.

Команды, предназначенные для ЭВМ, необходимо записывать в понятной компьютеру форме. С этой целью применяют *языки программирования* — искусственные языки, алфавит, словарный запас и структура которых удобны и понятны компьютеру.

▶ В самом общем смысле *языком программирования* называется фиксированная система обозначений и правил для описания алгоритмов и структур данных.

Языки программирования должны быть понятны и человеку, и ЭВМ. Они делятся на языки низкого и высокого уровня.

Язык низкого уровня — средство записи программы простыми приказами — командами на аппаратном уровне. Такой язык отражает структуру данного класса ЭВМ, и поэтому иногда называется *машинно-ориентированным языком.* Пользуясь системой команд, понятной ПК, можно описать алгоритм любой сложности, но такая запись для сложных задач будет очень громоздкой и мало приспособленной для использования человеком.

Существенной особенностью языков низкого уровня является жесткая ориентация на определённый тип аппаратуры (систему команд процессора).

Чтобы приспособить язык программирования низкого уровня к человеку, был разработан язык символического кодирования — язык *Ассемблер*. Структура команд Ассемблера определяется форматами команд и данных машинного языка. Программа на Ассемблере ближе человеку, потому что операторы этого языка — те же коды, но они имеют мнемонические названия; используются не конкретные адреса, а их символьные имена.

Многочисленную группу составляют языки программирования высокого уровня. Средства таких языков допускают описание задачи в наглядном, легко воспринимаемом виде. Отличительной особенностью этих языков является ориентация не на систему команд той или иной ЭВМ, а на систему операторов, характерных для записи определённого класса алгоритмов.

К языкам программирования этого типа относятся *Бейсик, Фортран, Паскаль, Си* и другие. Программа на языках высокого уровня записывается системой обозначений, понятной человеку (например, фиксированным набором слов английского языка).

Все вышеперечисленные языки — вычислительные. Более молодые — декларативные (непроцедурные) языки. Отличительная черта их — задание связей и отношений между объектами и величинами и отсутствие определенной последовательности действий (один из первых — *Пролог*, затем *C++*, *Delphi*, *Visual Basic*). Эти языки дали толчок к разработке

специальных языков искусственного интеллекта и языков представления знаний.

Трансляторы

Текст программы, записанный, например, на Паскале, не может быть воспринят ЭВМ непосредственно, требуется перевести его на машинный язык.

Перевод программы с языка программирования на язык машинных кодов называется *трансляцией* (*translation* — перевод), а выполняется специальными программами — *трансляторами*. Существует три вида трансляторов: интерпретаторы, компиляторы, ассемблеры.

Интерпретатором называется транслятор, производящий покомандную обработку и выполнение исходной программы.

Компилятор преобразует (транслирует) всю программу в модуль на машинном языке, после этого программа записывается в память ПК и лишь потом выполняется.

Ассемблеры переводят программу, записанную на языке автокода, в программу на машинном языке.

Любой транслятор решает следующие основные задачи:

- анализирует транслируемую программу, в частности, проверяет, содержит ли она синтаксические ошибки;
- генерирует выходную программу (её часто называют объектной или рабочей) на языке команд ЭВМ;
- распределяет память выходной программы, в простейшем случае назначает каждому фрагменту программы: переменным, константам и другим объектам свои адреса в памяти.

6.2. Общие сведения о языке Паскаль

Язык Паскаль создавался в 1968–1971 годах Никлаусом Виртом (Швейцария) для обучения программированию и был назван в честь выдающегося французского математика и философа Блеза Паскаля (1623–1662). Позже язык Паскаль получил широкое распространение в виду ряда преимуществ:

- достаточно лёгок в изучении благодаря компактности и удачному описанию;
- отражает фундаментальные и наиболее важные идеи алгоритмов в очевидной и легко воспринимаемой форме, что предоставляет программисту средства, помогающие проектировать программы;
- позволяет чётко реализовать идеи структурного программирования и структурной организации данных;

- язык Паскаль сыграл большую роль в развитии методов аналитического доказательства правильности программ, позволил перейти к автоматической проверке;
- повысилась надёжность разработанных программ за счёт требований Паскаля к описанию переменных (при компилировании без выполнения).

Алфавит языка. Идентификаторы и зарезервированные слова

Набор символов языка Паскаль является подмножеством набора символов кода ASCII, в котором используются следующие символы:

- прописные и строчные буквы латинского алфавита (не различаются), а также символ подчеркивания, который используется наравне с буквами;
- цифры от 0 до 9;
- специальные символы: #, $, ‘, (,), *, +, ,, —, ., /, :, ;, <, >, =, @, [,], ^, {, };
- символ пробела.

▶ *Идентификаторы* используются для обозначения имен переменных, констант, типов, процедур, функций, модулей, программ и представляют собой последовательность букв, цифр и символа подчеркивания. Первым символом должна быть буква или символ подчеркивания.

Пример правильной записи идентификатора: Prog1, PeremX, _a2.

Пример ошибочной записи: 1_Program, Perem X, a!2.

Часто вместо слова «идентификатор» используется термин «имя». Различают стандартные идентификаторы и идентификаторы пользователя. Стандартным идентификаторам разработчиками заранее приписывается определенный смысл, например, обозначение типов данных (Integer, Boolean и т. п.), констант (False, True, Maxint), функций (Abs, Sqr), процедур (Read, Write). Идентификаторы пользователя задаются программистом и служат для обозначения определенных им объектов.

▶ *Зарезервированные слова* служат для определенной цели и имеют один единственный фиксированный смысл. В Паскале имеются следующие зарезервированные слова:

```
And             Goto            Program
Asm             If              Record
Array           Implementation  Repeat
Begin           In              Set
```

Case	Inherited	Shl
Const	Inline	Shr
Constructor	Interface	String
Destructor	Label	Then
Div	Library	To
Do	Mod	Type
Downto	Nil	Unit
Else	Not	Until
End	Object	Uses
Exports	Of	Var
File	Or	While
For	Packed	With
Function	Procedure	Xor

6.3. Данные в Паскале. Простые типы данных

Решение любой задачи на ЭВМ сводится к определенным действиям над данными с целью получения конечного результата.

Под *данными* понимают представление фактов или идей в формализованном виде, пригодном для передачи и обработки в процессе, реализуемом на ЭВМ.

В программе данные представляют собой значения констант или переменных.

▶ *Переменными* называют элементы данных, которые могут менять свои значения в процессе выполнения программы.

▶ *Константы* — это элементы данных, значения которых известны заранее и в ходе выполнения программы не меняются. Использование констант облегчает чтение программы и упрощает внесение исправлений, так как отпадает необходимость многократно исправлять значения по тексту программы: достаточно ввести новое значение при определении константы.

На рис. 6.1 приведена структура типов данных Паскаля.

▶ *Типы данных* определяют множество значений, которые могут принимать объекты программы (переменные, константы, функции, выражения), и множество операций, допустимых над этими значениями.

Целочисленный тип

В таблице 6.1 приведены названия целых типов, длина их внутреннего представления в байтах и диапазон возможных значений.

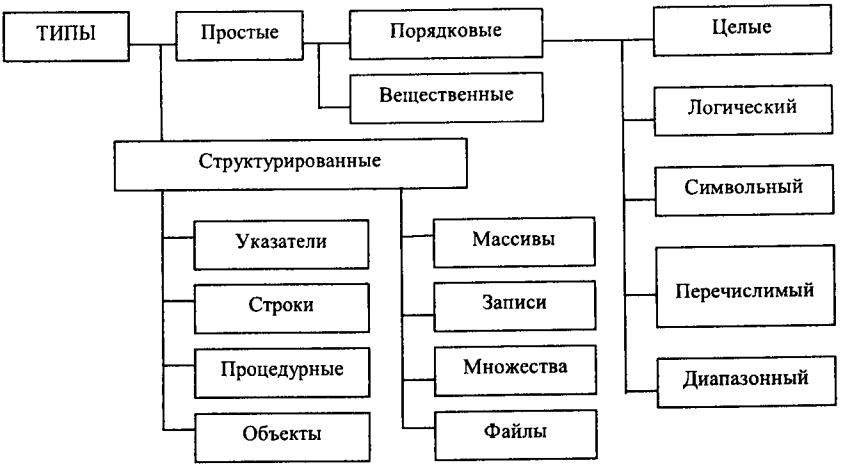


Рис. 6.1

Таблица 6.1

Целые типы

Название	Длина,байт	Диапазон значений
Byte	1	0...255
ShortInt	1	-128...+127
Word	2	0...65535
Integer	2	-32768...+32767
LongInt	4	-2147483648...+2147483647

Примеры значений целочисленного типа: -81, 0, 99.

Вещественный тип

Включает в себя вещественные числа (положительные, отрицательные и ноль), модуль которых лежит в определенном диапазоне (табл. 6.2).

Таблица 6.2

Вещественные типы

Название	Длина, байт	Количество значащих цифр	Диапазон десятичного порядка
Real	6	11...22	-39...38
Double	8	15...16	-324...+308
Extended	10	19...20	-4951...+4932
Comp	8	19...20	$-2\times10^{63}+1$...$+2\times10^{63}-1$

Примеры значений вещественного типа:

С фиксированной точкой:	С плавающей точкой:
21.18	2.118E+01
0.0	0.00000000E+00
-8.59	-8.59E+00

Логический тип

Обозначение — *Boolean*.

Переменные и константы этого типа принимают одно из двух логических значений, обозначенных стандартным именем True (истина) и False (ложь). При этом считается, что False<True.

Символьный тип

Обозначение — *Char*.

Литерный (или символьный) тип состоит из определенной упорядоченной последовательности символов, определяемой реализацией языка. Значения переменных и констант литерного типа включают в апострофы.

К типу Char применимы операции отношения, а также встроенные функции:

Chr(C) – функция типа Char; преобразует выражение С типа Byte (код символа в таблице ASCII) в символ;

Ord(C) – возвращает код символа С в таблице ASCII.

Примеры символьных констант: '8', '-', 'A', '!'.

Перечисляемый тип

Рассмотренные выше типы данных являются предопределенными. В языке Паскаль пользователь может определить новые типы переменных в виде упорядоченного множества значений – так называемые перечисляемый (перечислимый) и ограниченный (диапазонный) типы.

Определение перечисляемого типа заключается в непосредственном перечислении всех значений, которые может принимать переменная такого типа. Список возможных значений переменной заключается в круглые скобки, а сами значения разделяются символом «запятая». Нельзя одно и то же имя включать в определения разных перечисляемых типов. Введение нового типа осуществляется в разделе определения типов.

Пример:

```
Type
  OPERATORS=(PLUS, MINUS, DIVIDE);
  SIM=(A, C, D, E);
  METALL=(Fe, Na, Cu, Co);
```

```
Var M1, M2: METALL;
    OP1, OP2, OP3:   OPERATORS;
```

В рассмотренном примере переменные с именами *M1*, *M2* могут принимать только значения *Fe*, *Na*, *Cu* или *Co*, переменные *OP1*, *OP2*, *OP3* — только значения *PLUS*, *MINUS*, *DIVIDE* и т. д. Других значений этим переменным присваивать нельзя. Указываемые в круглых скобках имена являются константами, порядковый номер первой из них равен нулю, следующей – единице и т. д. В перечисляемом типе частные значения упорядочены, что означает, к примеру, что для рассмотренных выше значений *Fe<Na*, *MINUS<DIVIDE*, *A<E* и т. д. При этом старшинство связано с местом слова при перечислении. Не допускается применять операцию сравнения к операндам разного типа, например, сравнивать *MINUS* и *Fe* и т.п.

К перечисляемым типам можно применять операции отношения (оба операнда должны иметь один тип), а также использовать их в качестве аргументов стандартных функций succ(x), pred(x), ord(x).

Рассмотренный ранее тип boolean может быть определен как перечисляемый следующим образом:

```
Type Boolean=(FALSE, TRUE);
```

Определение предполагает, что для имен констант FALSE и TRUE выполняется отношение: FALSE<TRUE.

Ограниченный (диапазонный) тип данных

Ограниченный (диапазонный, интервальный) тип определяется на базе уже заданного типа (например, INTEGER, CHAR, перечисляемого). Для описания ограниченного типа нужно задать нижнюю и верхнюю границу значений:

TYPE <имя типа>=<нижняя граница>..<верхняя граница>;

Первая константа определяет нижнюю границу, затем идет знак «две точки»(..), после которого указывается вторая константа. При этом нижняя граница не должна превышать верхнюю, а их тип должен быть один и тот же.

Примеры:

```
Type  MONTH = (JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG,
   SEP, OCT, NOV, DEC);     {перечисляемый тип}
   LETO = JUN..AUG;         {диапазон месяцев}
INDEX = (0..90);            {диапазон для INTEGER}
SIMVOL = 'A'.. 'G';         {диапазон для CHAR}
```

К значениям ограниченного типа применимы те же операции, что и к значениям базового типа, используемым при задании в разделе определения типов. Если, к примеру, задано определение:

```
Type BUKVA = 'A'.. 'Z';
Var Y: BUKVA;
```

то можно выполнить присваивание:

```
Y:= 'D';
```

6.4. Операции и выражения

Выражения

Они определяют порядок выполнения действий над элементами данных. Выражения строятся из операторов, соединенных знаками операций. В качестве операндов используются переменные, константы, обращения к функциям. Как и в математике, для управления порядком выполнения операций в выражениях можно использовать круглые скобки. Операции задают действия, которые необходимо выполнить над операндами. Тип выражения совпадает с типом результата. В отдельных случаях выражение может состоять из одной переменной или константы, или обращения к функции.

Примеры.

1. A+C-10/12 – выражение, в котором A, C, 10, 12 – операнды; +, -, / — знаки операций сложения и деления.
2. B*10.12*sin(x) – выражение, в котором в качестве операндов выступают переменная, константа, обращение к функции.
3. 18 – выражение, состоящие из одной целочисленной константы.
4. STEM – выражение, представляющее собой имя переменной или константы.

Арифметические выражения и операции

В результате выполнения арифметического выражения получится целое или вещественное значение. В таблице 6.3 сведены арифметические операции языка Паскаль.

Операции возведения в степень в Паскале нет. Для выполнения этой операции можно использовать стандартные функции Exp(x) и Ln(a), воспользовавшись соотношением

$y = a^n = e^{\ln a^n}$, тогда $a = Exp(n*Ln(a))$.

Таблица 6.3

Арифметические операции

Знак операции	Операция
*	Умножение
/	Деление
Div	Целочисленное деление
Mod	Остаток от деления
–	Минус
+	Сложение
–	Вычитание

При написании арифметических выражений необходимо помнить следующее:

- наивысший приоритет имеют операции умножения и деления, соответственно они выполняются раньше, чем операции с более низким приоритетом (сложение и вычитание); если все операции имеют одинаковый приоритет, то они выполняются последовательно слева направо, выражения в скобках выполняются в первую очередь;
- нельзя записывать подряд знаки двух операций;
- не допускается запись выражений, не имеющих смысла.

Примеры.

1. C+–D — неверная запись.
2. C+(–D) — верная запись.
3. P*2/C. Здесь P умножается на 2, результат делится на C.
4. (A*C)*(2+M) — вначале выполняются действия в скобках, и только потом умножение.
5. D/(A–A) — неверная запись, ноль в знаменателе.

Логические операции и выражения

Логические (булевские) выражения строятся из логических констант и переменных, логических функций, выражений отношения путем применения круглых скобок и логических операций. В результате выполнения логического выражения получается логическое значение – одно из двух True или False.

В Паскале определены следующие логические операции (приведены впорядке убывания приоритета):

NOT — логическое отрицание.

AND — логическое «и»;

OR — логическое «или»;

Результаты применения данных операций определяются таблицами истинности (см. табл. 6.4).

Таблица 6.4

Значение A	Значение B	Результат
A **and** B		
False	False	False
True	False	False
False	True	False
True	True	True
A **or** B		
False	False	False
True	False	True
False	True	True
True	True	True
Not A		
False		True
True		False

Операции отношения

Операции отношения выполняют сравнение двух операндов. В результате выполнения операции отношения получают значение булевского типа — True или False. Операции отношения можно применять к данным типа Real, Integer, Boolean, Char. При этом в одном выражении можно одновременно использовать типы Real и Integer. Если в одном выражении имеются арифметические, логические операции и операции отношения, то вначале выполняются арифметические и логические, а затем операции отношения.

Стандартные функции

Стандартные встроенные функции необходимы для выполнения часто встречающихся действий, к примеру, вычисления значений элементарных функций (*sin* x, *tg* x и т.д.). Для обращения к стандартным функциям в выражениях используется идентификатор функции, после которого в круглых скобках указывается фактический параметр. Фактический параметр может быть константой, переменной или представлять сложное выражение. В табл. 6.5 и 6.6 приведены стандартные функции Паскаля.

Таблица 6.5

Математические функции в Паскале

Функция	Запись в Паскале	Смысл. Тип результата
x^2	Sqr(x)	Квадрат числа. Целый или вещественный, в зависимости от типа аргумента
$\sqrt{x}$	Sqrt(x)	Квадратный корень из x. Вещественный
Sin x	Sin(x)	Синус x. Вещественный
Cos x	Cos(x)	Косинус x. Вещественный
Arctg x	Arctan(x)	Арктангенс x. Вещественный
Ln x	Ln(x)	Натуральный логарифм от x. Вещественный.
e^x	Exp(x)	Экспонента от x. Вещественный

Приоритеты операций

Операции в выражениях выполняются в следующем порядке:

- вызовы функций;
- операция NOT;
- операции типа умножения *, /, div, mod, and;
- операции типа сложения +, -, or;
- операции отношения =, < >, <, >, <=, >=.

Операции с одинаковым приоритетом выполняются слева направо. Порядок выполнения операций можно изменить с помощью скобок.

Таблица 6.6

Стандартные функции Паскаля

Функция	Тип Результата	Смысл, результат выполнения (его тип)	Пример
1	2	3	4
Ord(*x*)	Целые Char Boolean (дискретные)	Порядковый номер аргумента x. Целочисленный	Ord(50)=20 Ord(' ')=32 Ord(False)=0 Ord(true)=1
Pred(*x*)	(дискретные)	Предыдущее значение	Pred(-6)=-7 Pred('z')='y'
Succ(*x*)	(дискретные)	Следующее значение. Возвращаемое значение совпадает с аргументом	Pred(True)= =False Succ(100)=101 Succ('f')='g' Succ(False)= =True

Окончание табл. 6.6

1	2	3	4
Chr(*x*)	Byte	Символ, код которого x. Литерный	Chr(32)=' '
Dec(*x*,[*i*]) Inc(*x*,[*i*])	Целые	Увеличивает x на i. Уменьшает x на i. Совпадает	Dec(100,10)=90 Inc(50)=51 Inc(50,-5)=45
y Div *x* *y* Mod *x*	Целые Целые	Целая часть от деления. Остаток от деления. Целочисленный	13 Div 5=2 13 Mod 5=3
Odd(*x*)	Целые	True при x нечетном, False при x четном	Odd(25)=True Odd(100)=False
Trunc(*x*)	Вещественный	Отбрасывает дробную часть, целое.	Trunc(5.7)=5 Trunc(-4.6)=-4
Round(*x*)	Вещественный	Округляет по правилам, целое.	Round(5.7)=6 Round(-4.6)=-5
Int(*x*)	Вещественный	Целая часть от x, вещественное.	Int(-4.6)=-4.0
Frac(*x*)	Вещественный	Дробная часть от x, вещественное	Frac(-4.6)=-0.6 Frac(8.91)=0.91
Radom[(*i*)]	Real, Word	Генерирует случайные целые числа из диапазона (от 0 до i-1)	Random – случайное вещественное
	---	или (если без параметра) вещественное число от 0 до 1.	число от 0 до 1. (0.583)
Randomize		Инициализирует случайным значением (текущим системным временем), генератор случайных чисел	Random(100) – случайное целое число от 0 до 99

6.5. Структура программы

Программа, составленная на языке Паскаль, состоит из заголовка и «тела» программы, называемого блоком. Оканчивается программа точкой. Заголовок программы начинается с зарезервированного слова PROGRAM, после которого следует имя программы, за именем программы указывается список параметров (обычно это файлы INPUT и OUTPUT). Список параметров может отсутствовать. Имя программы представляет собой идентификатор.

Примеры:

```
Program STUD(INPUT, OUTPUT);
Program LAB1(INPUT, OUTPUT);
Program RASCHET;
```

Блок состоит из двух частей (разделов): раздела описания и раздела операторов. Описания определяют объекты (данные) программы, а операторы предписывают действия, которые необходимо совершить над введенными объектами. Всего в блоке может быть шесть разделов. Любой из них (кроме последнего — раздела операторов) может отсутствовать.

В общем виде структуру программы на Паскале можно представить следующим образом:

1) раздел определения меток;
2) раздел определения констант;
3) раздел определения типов;
4) раздел описания переменных;
5) раздел описания процедур и функций;
6) раздел операторов.

Поскольку современные концепции программирования считают «плохим стилем» использование операторов безусловного перехода, а следовательно, отпадает необходимость в метках, в данном учебнике раздел определения меток и оператор GOTO не рассматриваются. Читатель может найти описание этих разделов Паскаля, например, в [15, 17].

Раздел определения констант

Как уже отмечалось, можно обозначить константы именами и в дальнейшем в программе использовать только имена констант. Раздел начинается с зарезервированного слова CONST, после которого следует список определений констант. В конце списка ставится символ «точка с запятой». Каждое определение константы представляет собой идентификатор, за ним следует знак равенства (=) и само значение константы. Определения констант отделяются друг от друга символом «точка с запятой».

Примеры:

```
CONST maxsn=1000;
CONST maxn=100; s='a'; d=4.2;
```

Раздел определения типов

В этом разделе пользователь может по своему усмотрению определять собственные типы и давать им любые имена. Раздел начинается со слова TYPE, за которым следует одно или несколько определений типов, отделяемых друг от друга символом «точка с запятой». Каждое определение состоит из идентификатора, за которым следует знак равенства (=), и задания типа или его имени. В конце раздела ставится символ «точка с запятой».

Пример:

```
Type
NN=integer;
Vector=array[1..10] of real;
Color=(RED, YELLOW, GREEN, BLUE);
```

Раздел описания переменных

Все используемые в программе переменные должны быть описаны в этом разделе, начинающемся с ключевого слова VAR, за которым идет последовательность объявлений переменных. Объявления переменных отделяются друг от друга точкой с запятой. В объявлении через запятые перечисляются имена переменных, затем следует символ «двоеточие» и их тип.

Пример:

```
Var
    A,B,C: integer;
        D: real;
        V: boolean;
        F: vector;
```

Раздел описания процедур и функций

В нем размещаются процедуры и функции пользователя (см. п. 6.10).

Раздел операторов

Он начинается с зарезервированного слова BEGIN и оканчивается словом END, после которого должна стоять точка:

```
BEGIN
<оператор;>
...
<оператор>
END.
```

В разделе операторов выполняются действия над предварительно описанными переменными, константами, значениями функций. Операторы отделяются друг от друга символом «точка с запятой». Допускается не ставить «;» перед END.

Ниже приводится п р и м е р простой программы на языке Паскаль, предназначенной для вычисления факториала числа N (N!).

```
Program prim1 (INPUT, OUTPUT); {заголовок программы}
  Var Fact, X, N: integer;   {раздел описаний и
   определений}
  Begin                {начало раздела операторов}
       Read(N);
       Fact:=1; X:=0;
       While X<>N DO
            Begin
                 X:=X+1; Fact:=Fact*X;
            End;
       Writeln(' N=', N, ' N!=', Fact)
  End.                          {конец программы}
```

Директивы компилятора и управляющие символы

Директивы компилятора используются программистом для управления режимами компиляции.

Директива — сообщение в повелительной форме, вводимое оператором и содержащая указания о том, какие необходимо выполнить действия.

Директива компилятора — компонент программы, управляющий последующей компиляцией программы. Директива компилятора имеет следующий формат записи: заключается в фигурные скобки «{}», символ $ <латинская буква> {±}.

Пример: {$R-}, {$V+, K-}.

По умолчанию директивы гарантируют минимальный объем объектного модуля и минимальное время компиляции.

Управляющие символы — знак «#» или «^».

«#» — и следующее за ним целочисленное значение обозначают символ кодовой таблицы ПК, имеющий соответствующее десятичное значение.

Например: #10#13 — конец строки, переход на новую.

«^» — указывает, что за ним следует управляющий символ.

Например: ^Y#10#13.

Библиотечные модули — это библиотеки подпрограмм (процедур и функций).

Библиотечные модули — результат компиляции в режиме compile с директивой **destination = disk** одной или нескольких процедур и функций. Имя модуля может указываться в разделе USES для возможности использования каждой из находящихся в нём процедур или функций.

Создание библиотечного модуля требует определённой организации с применением специальных зарезервированных слов.

6.6. Операторы языка Паскаль

Операторы служат для задания действий над данными. Для однозначной интерпретации программы набор допустимых операторов зафиксирован, и четко определены правила их записи, которые нельзя нарушать.

6.6.1. Оператор присваивания

Общий вид оператора:

<идентификатор> := <выражение>;

Оператор присваивания выполняется следующим образом: выполняется выражение, стоящее в правой части, и результат присваивается переменной, имя которой расположено в левой части. При этом переменная и выражение должны иметь один и тот же тип. Исключение составляет случай, когда переменной типа real присваивается переменная типа integer. Допускается присваивание всех типов данных, кроме типа файла.

Пример:

```
KOR := 2;
RESULT := SIN(A)+2;
```

6.6.2. Операторы вывода и вывода информации

Вывод данных

▶ Вывод информации представляет собой передачу данных после обработки на внешние носители, которыми могут быть видеомонитор, печатающее устройство, файл (о выводе данных в файлы и чтении из них см. п. 6.9).

Для выполнения операций вывода используются операторы WRITE и WRITELN.

Формат оператора вывода (записи):

WRITE (V1, V2,...,VN);

где V1, V2,..., VN — выражение целочисленного типа, вещественного, символьного, строкового, логического.

В операторе вывода можно указывать также формат данных. Для данных типа REAL это делается следующим образом:

WRITE(VAR1: P: Q);

где VAR1 — выражение типа REAL, P — общее число знакомест, Q — число цифр, выводимых после десятичной точки.

Примеры:

Значение A	Оператор	Результат
210.11	WRITE(A:8:4)	210.0400
–21.7822	WRITE(A:7:2)	–21.78

В последнем примере символ «_» означает пробел.

Если выражение имеет тип integer, char, boolean, то указывается общее число позиций:

WRITE(V1: P);

Для случая вещественного типа выражения и отсутствия указателя числа позиций вывод осуществляется в формате с плавающей точкой, а под число отводится поле шириной 18 символов. Данные типа BOOLEAN, INTEGER, CHAR при отсутствии указателя числа отводимых под результат позиций выводятся, начиная с позиции расположения курсора.

Примеры:

Значение A	Оператор	Результат
123	WRITE(A,A:4,A:4)	123_123_123
'D'	WRITE(A,A)	DD
TRUE	WRITE(A,A)	TRUETRUE
834.218	WRITE(A)	8.3421800000E+02
–2.111E+01	WRITE(A)	–2.1110000000E+01

Оператор вывода WRITELN действует подобно оператору WRITE, но в отличие от него, после вывода значения последнего элемента списка выполняется перевод курсора к началу следующей строки. WRITELN

без параметров выполняет перевод курсора к началу следующей строки. Таким образом, оператор WRITELN (V1,...,VN) эквивалентен двум операторам WRITE(V1,...,VN); WRITELN.

Ввод данных

▶ Операторы ввода данных позволяют задать значения переменным во время выполнения программы с клавиатуры или из файла.

Формат оператора:

READ (VAR1, VAR2, ... , VARN);

где VAR1, VAR2, ... , VARN — идентификаторы переменных. Значения переменных VAR1, VAR2, ... , VARN вводятся с клавиатуры (друг от друга они должны отделяться хотя бы одним пробелом). После набора данных для одного оператора READ нажимается клавиша «Enter». При этом необходимо, чтобы вводимые данные и переменные в операторе READ имели один и тот же тип или совместимый тип.

Пример:

Оператор	Набираемый на клавиатуре текст
1. READ (A,B); (A, B – INTEGER)	10 12 <ввод>
2. READ (A,B); READ (C,D); (C, D - INTEGER)	10 12 <ввод>14 16 <ввод>

Оператор READLN действует аналогично оператору READ с той разницей, что после считывания последнего значения в списке для оператора READLN осуществляется автоматический переход к считыванию следующей строки данных.

Пример:

Оператор	Набираемый на клавиатуре текст
READLN (A,B); READLN (C,D);	10 12 <ввод> 14 16 <ввод>

Пример программы расчета площади прямоугольника по известным его сторонам.

```
Program prim5;
Var A, B, S: real; {A, B – стороны, S – площадь}
Begin
Readln (A, B);
S := A*B;
```

```
Write ('для сторон A= ', A:10:2, 'B= ', B:10:2, '-');
Writeln ('площадь прямоугольника S=', S:12:4)
End.
```

6.6.3. Составной оператор

Часто при некотором условии необходимо выполнить определенную последовательность операторов, а по правилам языка допускается использование только одного оператора. В этом случае последовательность операторов объединяют в один составной оператор, который воспринимается как единое целое и может располагаться в любом месте программы.

Составной оператор начинается с зарезервированного слова BEGIN и заканчивается словом END, между которыми размещаются требуемые операторы, отделенные друг от друга символом «точка с запятой». После BEGIN ставятся пробелы, а после END — символ «точка с запятой». Операторы, входящие в составной оператор, выполняются последовательно (если среди них нет операторов перехода).

Пример составного оператора:

```
Begin I:= 8; C:=16; writeln (I*C: 10)  end;
```

6.6.4. Условный оператор

▶ *Условный оператор* позволяет выполнить некоторый оператор только в том случае, если истинно некоторое условие (логическое выражение).

Формат условного оператора:

IF <выражение> THEN <оператор1> [ELSE <оператор2>];

Здесь «выражение» — выражение логического типа. Если оно истинно, выполняется оператор1. Если оно ложно, то либо управление передается на оператор, следующий сразу за оператором IF, либо выполняется оператор2, расположенный после слова ELSE. Здесь и далее «Операторы» — любые операторы Паскаля, простые или составные.

Примеры:

```
If A>B then C:=D else C:=8;
If R then C:=10; {R имеет тип boolean}
If (A>B) and (D<>K)
  Then
       Begin A:=B; D:=K end
  Else A:=B-C+K;
```

Пример программы расчета корней квадратного уравнения $ax^2+bx+c=0$.

```
Program prim6;
Var A, B, C, D, X1, X2: real;
  Begin
   Readln (A, B, C);
   Writeln ('A=', A, ' B=', B, ' C=', C);
   If sqr(B)-4*A*C<0
     Then writeln ('Действительных корней нет')
     Else
       Begin
        D:=sqrt(sqr(B)-4*A*C);
        X1:=(-B-D)/(2*A); X2:=(-B+D)/(2*A);
        Writeln('корни уравнения X1=', X1, 'X2=',X2)
       End
  End.
```

6.6.5. Оператор варианта CASE

▶ CASE используется, когда необходимо выбрать вариант направления расчетов не из двух, а из большего числа вариантов.

Формат оператора:

CASE <выражение> OF
X1: <оператор 1>;
X2: <оператор 2>;
...
XN: <оператор N>
ELSE <оператор>
END;

Метки операторов X1, X2, ... , XN представляют собой либо отдельную константу, либо список констант, перечисленных через запятые, либо диапазон. Тип выражения и констант должен быть одним и тем же. Используемые в операторе CASE метки X1, X2, ... , XN не описываются в разделе описания меток и отличаются по смыслу от меток, там описанных.

Оператор выполняется следующим образом: вычисляется значение выражения, далее выполняется только тот оператор, константа выбора которого равна значению выражения. Если среди констант нет равной

значению выражения, то выполняется оператор, следующий за словом ELSE (при отсутствии слова ELSE выполняется оператор, следующий за словом END). Выражение может иметь любой дискретный тип.

Пример программы вывода дня недели в зависимости от значения введенного числа:

```
Program prim7;
  Var den: integer;
  Begin
       Readln (den);
       Case den of
            1: writeln ('Понедельник');
            2: writeln ('Вторник');
            3: writeln ('Среда');
            4: writeln ('Четверг');
            5: writeln ('Пятница');
            6: writeln ('Суббота');
            7: writeln ('Воскресенье');
           Else writeln ('Вы неверно ввели число')
       End
End.
```

6.6.6. Операторы цикла

▶ *Циклом* называется многократное повторение определенного действия или группы действий. В Паскале различают 3 вида циклов.

6.6.6.1 Оператор цикла по счетчику (цикл с параметром)

Оператор цикла с параметром используется, когда число повторений цикла заранее известно. Формат:

FOR <Параметр цикла> := <Выражение 1> $\begin{Bmatrix} \text{TO} \\ \text{DOWNTO} \end{Bmatrix}$ <Выражение 2> DO <Оператор>;

Данный оператор называют «циклом с параметром», так как число повторений подсчитывается в переменной, называемой параметром цикла.

«Параметр цикла» — это имя, описанное в разделе VAR *дискретного* типа (Boolean, Integer, Char, Byte и т.д.). В ней подсчитывается число повторений цикла. «Выражение1», «Выражение2» — выражения, оп-

ределяющие, соответственно, начало и конец значения параметра цикла; по типу они должны совпадать с типом параметра цикла (обозначим их, соответственно, Выр. 1 и Выр. 2).

TO — означает возрастание параметра цикла (Выр. 1 > Выр. 2) с шагом 1.

DOWNTO — убывание параметра цикла (Выр. 1 > Выр. 2) с шагом 1;

«Оператор» — любой оператор (в том числе и составной).

Порядок выполнения цикла FOR:

1. Переменной — параметру цикла присваивается значение Выражения 1.
2. Выполняется оператор.
3. Параметр увеличивается (уменьшается) на 1.
4. Значение параметра сравнивается с Выражением 2.

Цикл прекращается, выполняется следующий за ним оператор:

— если значение параметра < Выражения 1 (для TO);

— если значение параметра > Выражения 2 (для DOWNTO);

— иначе повторяются действия, начиная с п. 2.

Пример 1. Вычислить $n!$ $(1 \times 2 \times 3 \times 4 \ldots \times n)$.

```
Program fact_1;
  Var i,n : integer; {i – параметр цикла, n – его
                      конечное значение}
        f: longint; {результат}
  Begin
    Write ('Введите n '); readln(n);  f:=1;
     For i:=1 to n do      f:=f * i;
     Writeln ('при n= ', n,'n!=' ,f)
  End.
```

Пример 2. Распечатать буквы латинского алфавита в обратном порядке.

```
Program for_2;
  Var c: char;
  Begin
  For   c := 'Z'  downto 'A' do
    Write (c)
  End.
```

Пример 3. Найти значение выражений: $\sum_{x=1}^{10} x^2$; $\prod_{x=1}^{10} \frac{1-x^2}{|x|}$ (сумму S и произведение П считать в одном цикле).

```
Program for_3;
  Var X, S: integer;
      P: real;
  Begin S := 0;  P := 1;
    For X :=1  to 10 do
      Begin
        S := S + sqr(X);
        P := P * (1 - sqr(X))/abs(X)
      End;
    Writeln ('S=', S, 'P=', P )
  End.
```

6.6.6.2. Оператор цикла с предусловием

Проверка условия выполнения тела цикла с предусловием производится в самом начале оператора (следовательно, может не выполняться ни разу). Формат оператора:

WHILE <Выражение> DO <Оператор>;

«Выражение» — условие логического типа.

«Оператор» — любой оператор Паскаля, простой или составной.

Оператор WHILE задает многократное выполнение оператора, стоящего после DO. Перед каждым выполнением тела цикла вычисляется значение выражения — условия. Если результат равен True (истина), то выполняется оператор после DO, если False (ложь), то действие оператора WHILE прекращается.

Программа вычисления факториала с использованием данного вида цикла будет выглядеть следующим образом:

```
{фрагмент программы}
Begin
  i := 1; f:=1; readln(n);
  While i < =n  do
    Begin
      f : = f * i; inc(i);
    End;
```

Пример. Подсчитать сумму чисел в интервале от 100 до 150, вводимых с клавиатуры. Как только введено отрицательное число, ввод и суммирование прекратить.

```
{Текст программы}
Program  while_2;
  Const    Amin = 100; Amax = 150;
  Var      A, Summa : integer;
Begin
  Summa:=0;
  Writeln("Введите A "); Readln(A);
  While A >= 0  do  {заголовок цикла}
    Begin
     If (A > 100) and (A < 150) then Summa := Summa + A;
      Inc (Summa, A);
      Readln (A) {читаем новое значение A}
    End;   {конец цикла}
  Writeln( 'Значение суммы = ', Summa)
End.
```

6.6.6.3. Оператор цикла с постусловием

Формат:

REPEAT
 <Оператор>
UNTIL <Выражение>;

«Оператор» — любой оператор Паскаля.
«Выражение» — выражение логического типа.

Данный оператор аналогичен оператору цикла с предусловием. Отличия состоят в том, что:

1) условие проверяется после выполнения оператора, следовательно, хотя бы один раз оператор выполнится;

2) цикл прекращает выполняться, когда значение выражения равно true (истина). Если результат логического выражения false (ложь), то тело цикла активизируется (выполняется) еще раз.

Факториал *n*! с помощью цикла Repeat можно вычислить следующим образом:

```
{фрагмент программы}
i := 1; f := 1; readln(n);
Repeat
  f := f * i; inc(i)
Until i > n;
```

Пример. С клавиатуры вводятся числа. Найти сумму этой последовательности. Как только введено число большее 999, суммирование выполнить последний раз и выдать результат на печать.

```
{Программа подсчета суммы}
Program  sum;
   Const  PRK = 999;
   Var     X, Summa: real;
   Begin
     Summa := 0;
     Repeat
       Readln(X) ;
       Summa := Summa + X
     Until X > PRK;
     Writeln (Summa:10:4)
   End.
```

6.7. Структурированные типы данных

6.7.1. Массивы

▶ *Массив* представляет собой структуру, состоящую из фиксированного числа компонент одного типа. В качестве компонент можно использовать как ранее описанные типы, так и следующие: массивы, записи, множества, указатели и т. п. Число элементов в массиве фиксируется при описании и далее при выполнении программы не меняется.

Определение типа, значения которого являются массивами, выполняется следующим образом:

TYPE <имя типа> = ARRAY[<диапазон первого индекса>,...,
<диапазон n-го индекса>] OF <тип компонент>;

Количество индексов n определяет размерность массива, а сами индексы разделяются запятыми и заключаются в квадратные скобки.

Пример:

```
Type matr=array[1..2,1..12] of real;
Var A,B,C: matr;
```

Массив можно описать в разделе Var следующим образом:

<идентификатор>: ARRAY [<диапазон первого индекса>,..., <диапазон *n*-го индекса>] OF <тип компонент>;

Пример:

```
Var A, B, C: array[1..10] of integer;
```

Для обращения к элементам массива используются конкретные значения индексов. Индекс представляет собой выражение любого простого (скалярного) типа (кроме real). К примеру, оператор B[3] := 10; присваивает третьему элементу одномерного массива с именем B значение 10.

Пример. Пусть двумерный массив описан следующим образом: `Var A : array[1..2,1..4] of integer;` а в памяти ЭВМ записана таблица чисел, представляющая этот массив:

17	11	4	5
22	8	16	12

Все элементы в таблице имеют тип integer. При обращении к элементам матрицы A первый индекс указывает номер строки таблицы (изменяется в данном случае от 1 до 2), второй — номер столбца (в нашем примере изменяется от 1 до 4). Если задать оператор присваивания в виде X := A[2,3]; то после его выполнения значение некоторой переменной X будет равно 16. Ввод и вывод значений элементов массива производится поэлементно.

Рассмотрим несколько типичных задач, связанных с применением массивов.

1. {*Программа, позволяющая найти сумму элементов одномерного массива*}

```
Program msg1;
Const n=15; {число элементов массива}
Var    a : array [1..n] of real;
summa: real;
i:integer;
```

```
Begin
  summa := 0;
  For i:=1 to n do
    Begin
      Readln (a[i]);
      summa := summa+a[i]
    End;
 Writeln('сумма ',n,' элементов массива равна ', summa)
End.
```

2. {*Программа поиска наибольшего элемента одномерного массива и его порядкового номера*}

```
Program msg2;
Const n=20;
Var a:array [1..n] of real;
      amax: real;
      i, ne: integer;
Begin
  Write ('введите элемент 1 '); readln (a[1]);
  amax := a[1]; ne:=1;
  For i:=2 to n do
    Begin
     Write ('введите элемент № ', i); readln (a[i]);
      if a[i] > amax then
        Begin
          amax:=a[i]; ne:=i
        End
    End;
  Writeln ('максимальный элемент равен ', amax);
  Writeln ('и имеет порядковый номер ', ne)
End.
```

3. {*Программа нахождения произведения двух матриц: A размером M×N и B размером N×K. Элементы результирующей матрицы C размером M×K рассчитываются по формуле* $C_{ij} = \sum_{k=1}^{N} a_{ik} \cdot b_{kj}$. *Значения M, N, K не превышают* 10}

```
Program msg3;
Const em=10;
```

```
Type matr=array[1..em, 1..em] of real;
Var a, b, c: matr;
      m, n, k, i, j, t: integer;
Begin
  Write('введите значения m, n, k '); readln (m, n, k);
  {ввод элементов матрицы a}
  For i:=1 to m do
    For j:=1 to n do readln (a[i, j]);
  {ввод элементов матрицы b}
  For i:=1 to n do
    For j:=1 to k do readln (b[i, j]);
  {умножение матриц}
  For i:=1 to n do
    For j:= 1 to k do
      Begin
        c[i, j]:=0;
        For t:=1 to n do
          c[i, j]:=c[i, j]+a[i, t]* b[t, j];
      End
End.
```

В Паскале разрешается присваивать значения одной переменной массива другой (если элементы массива имеют один тип и одинаковую размерность). К примеру, если массивы A и B имеют одинаковую размерность и тип элементов real, то допустимо присваивание: $A := B$.

Сортировка массивов

▶ *Сортировка* — распределение элементов множества по группам в соответствии с определенными правилами.

Например, сортировка «по невозрастанию» — это сортировка элементов массива, в результате которой получается массив, каждый элемент которого, начиная со второго, не больше стоящего от него слева. Существует достаточно большое число методов сортировки. Приведем лишь простейшие из них.

Линейная сортировка (отбором). Пусть необходимо упорядочить массив по возрастанию (убыванию) элементов.

Алгоритм:

1. Просматриваем элементы, начиная с 1-го. Ищем минимальный (максимальный). Меняем его местами с 1-м элементом.

2. Просматриваем элементы, начиная со 2-го. Ищем минимальный (максимальный). Меняем его местами со 2-м элементом.

3. И так далее до предпоследнего элемента.

Пример. Отсортировать массив символов по алфавиту.

```
Program linsort;
Const maxkol=26;
Var c : array[1..maxkol] of char;
      k, i, j: integer;
      vr : char;
Begin
  Write('Введите количество символов <= 26'); readln(k);
   Writeln('Введите',k, 'символов');
   For i:=1 to k do readln(c[i]);
     For i:=1 to k-1 do
     {Изменять размер неотсортированной части массива}
       For j:=i+1 to k do {Сравнивать по очереди i-й
элемент неотсортированной части массива со всеми
от i+1-го до конца}
         If c[j]<c[i] then
        {если в неотсортированной части массива нашли
       элемент, больший i-го, то обменять их местами}
           Begin
             vr:=c[i]; c[i]:=c[j]; c[j]:=vr;
           End;
  For i:=1 to k do write(c[i]); writeln
End.
```

Сортировка методом «пузырька». Данный метод получил такое название по аналогии с пузырьками воздуха в стакане воды. Более «легкие» (максимальные или минимальные) элементы постепенно «всплывают». В отличие от линейной сортировки, сравниваются только пары соседних элементов, а не каждый элемент со всеми (поэтому такая сортировка выполняется за меньшее число шагов, а следовательно, быстрее).

Пример. Отсортировать по убыванию массив методом «пузырька».

Алгоритм:

1. Последовательно просматриваем пары соседних элементов массива (m).

2. Если для соседних элементов выполняется условие $m[i-1]<m[i]$, то значения меняются местами.

```
Program sortpuz;
Const maxkol = 26;
Var c: array[1..maxkol] of char;
    k,i,j: integer;
    vr : char;
Begin
  Write('Введите количество символов'); readln(k);
  Writeln('Введите',k, 'символов');
  For i:=1 to k do readln(c[i]);
  For i:=2 to k do
    For j:=k downto i do
      If c[j-1]<c[j] then
      {вытеснить элемент справа влево -
       пузырек «всплывает»}
        Begin
          vr:=c[j-1]; c[j-1]:=c[j]; c[j]:=vr
        End;
  For i:=1 to k do write(c[i]);
  Writeln
End.
```

6.7.2. Строки

Для обработки текстов в Паскале используется тип String (строка).

▶ *Строка* трактуется как цепочка символов.

К любому символу можно обратиться так же как, к элементу одномерного массива Array[0..N] of Char. Количество символов в строке может меняться от 0 до N, где N — максимальное количество символов в строке. Значение N объявляется определением типа String[N] и может быть любой константой порядкового типа, но не больше 255. Самый первый байт в строке имеет индекс 0 и содержит текущую длину строки.

Действия над строками реализуются с помощью стандартных процедур и функций.

LENGTH(ST) — функция типа Integer, возвращает длину строки ST.

CONCAT (S1...SN) — функция типа String, возвращает строку, представляющую собой сцепление строк S1...SN.

COPY (ST, INDEX, COUNT) — функция типа String, копирует из строки ST Count символов, начиная с символа с номером Index.

DELETE (ST, INDEX, COUNT) — процедура, удаляющая Count символов из строки ST, начиная с символа с номером Index.

INSERT (ST1, ST, INDEX) — процедура, вставляющая в строку ST подстроку ST1, начиная с символа с номером Index.

POS (SUBST, ST) — функция типа Integer, отыскивает в строке ST первое вхождение подстроки SubST и возвращает номер позиции, с которой она начинается. Если подстрока не найдена, возвращается ноль.

STR (X,ST) — процедура, преобразует число X любого вещественного или целого типа в строку символов ST так, как это делает процедура WriteLn перед выводом. После X можно задать формат преобразования (как в процедуре вывода).

VAL (ST,X,CODE) — процедура, преобразует строку ST во внутреннее представление целой или вещественной переменной X, которое определяется типом этой переменной. Параметр Code содержит ноль, если преобразование прошло успешно и порядковый номер первого ошибочного символа в строке ST в противном случае.

Примеры использования приведенных процедур и функций:

```
Program Prim_St;
Var
  X: real;
  Y: integer;
  St,St1: string;
Begin
 St:=concat('12', '345'); {строка St содержит 12345}
  St1:=copy(st,3,Length(St)-2); {St1 содержит 345}
  Insert('-', st1,2);  {строка St1 содержит 3-45}
 Delete(St,Pos('2',St),3); {строка St содержит 15}
  Str(pi:6:2,St); {строка St содержит 3.14}
  Val('3,1415',X,Y); {y содержит 2, x остался без
   изменений}
End.
```

Примеры программ с использованием строкового типа:

1. {*Подсчитать количество цифр в произвольной строке*}

```
Program St_1;
Var s : string;
```

```
      i, nd : byte;
Begin
   Write('Введите произвольную строку'); readln(s);
     nd := 0;
     For i := 1 to length(s) do
   If s[i] in ['0'..'9'] then inc(nd);
     Writeln(nd);
     Readln
End.
```

2. *{Подсчитать сумму цифр в целом положительном числе}*

```
Program St_2;
Var x:word;
      i, d, sum : byte;
      code : integer;
      s : string;
Begin
Write('Введите целое положительное число');
readln(x);
    Str(x, S);
    For i := 1 to length(s) do
      Begin
        Val(s[i], d, code);
        sum := sum + d;
      End;
    Writeln(sum); Readln
End.
```

6.7.3. Множества

Наряду с числом, множество является фундаментальным математическим понятием. К операциям со множествами сводится большинство математических моделей. Паскаль — один из немногих языков, который имеет встроенные средства для работы со множествами. Давайте посмотрим, как «переводятся» на Паскаль некоторые понятия теории множеств.

В математике рассматривают конечные и бесконечные множества, состоящие из произвольных элементов. В Паскале множества всегда конечные, причем состоят не более чем из 256 элементов. Все элементы

множества должны быть одного порядкового типа (например, Integer, Word, Longint).

Обозначения, принятые

в математике	в Паскале
{1, 2, 3}	[1, 2, 3]
0	[]
{1, 2,...N}	[1...N].

В Паскале множество может быть задано выражениями, например: [2+x, 8-3].

Внутреннее представление множеств

Все значения множества представляются в памяти последовательностями битов одинаковой длины. За каждое значение базового типа «отвечает» 1 бит. Если множество содержит некоторый элемент, в соответствующем бите хранится 1, если не содержит — хранится 0.

Например:

```
Var  x, y : set of 1..10;
```

Внутреннее представление:

x := []	0000000000
x := [2,3,9]	0110000010
y := [2..7]	0111111000

Операции над множествами сводятся к поразрядным логическим операциям над последовательностями битов. Например, объединение множеств выполняется путем поразрядного логического сложения битов. Объединение множеств $x \cup y$

x:	0110000010
y:	0111111000
$x \cup y$:	0111111010

Поскольку поразрядные операции входят в набор команд процессора ЭВМ, они выполняются очень быстро.

▶ *Множества* в Паскале — это наборы однотипных, логически связанных между собой объектов, которые рассматриваются как единое целое. Причем характер связи подразумевается программистом и никак не контролируется Паскалем. Например, множество согласных букв кириллицы; множество простых чисел от 1 до 100.

Каждый объект в множестве называется *элементом множества.* Все элементы множества должны принадлежать к одному из скалярных типов. Этот тип называется *базовым типом.* Базовый тип задается диапазоном или перечислением. Если множество не имеет элементов, оно называется *пустым* ([ ]).

Для изображения множеств используют квадратные скобки, в которые заключается перечень элементов.

Например:
```
type color=(white, red, black);
Number=set of 1..31; {явное описание}
Var col: set of color; {неявное описание}
n1, n2: number;
letter: set of char; {неявное описание}
```

Попытка присвоить недопустимый символ вызовет прерывание. Контроль диапазонов можно установить опцией компилятора {$R+}.

Операции над множествами

В Паскале разрешены следующие операции над множествами: сравнения (=, < >, > =, < =), ∩ (and), ∪ (or), разность множеств (), включение в множество — in. Рассмотрим их назначение.

«=»: два *множества равны* тогда и только тогда, когда они имеют одинаковые элементы. Порядок следования роли не играет.

A: set of 1,2,3,4 B: set of 1,3,2,4 A=B true.
A: set of “a”,’b’,’c’ B: set of “a”,’b’ A=B false.

«< >»: *множества не равны*, если они отличаются по значению хотя бы одного элемента.

В случае 1) A < > B — false;
в случае 2) A < > B — true.

«> =» используется для определения *принадлежности множеств.*
A > = B, если все элементы B содержатся в множестве A.

«<=» — аналогично.

A: =1,2,3,4 B: =2,3,4 A>=B true
B<=A true

«in» используется для проверки *принадлежности какого-либо значения множеству*. Пусть x=’a’, y=’z’, B — множество символов “a”..’n’. Тогда x in B — true, y in B — false.

Обычно операция in используется в условном операторе. Например, вместо:

```
If(a=1) or (a=5) or (a=7) or (a=11) or (a=15) then…,
```

можно записать: if a in [1,5,7,11,15] then…,
т. е. «in» позволяет более эффективно производить сложные проверки условий. При этом множество не обязательно предварительно описывать в разделе описаний.

Если необходимо проверить, не принадлежит ли n множеству A, можно записать: Not (n in A) (неверно: n Not in A).

Объединение множеств ∪: «+». Объединением двух множеств является третье множество, содержащее элементы обоих множеств (выполняется путем поразрядного логического сложения).

Пусть A:=[1,2,3], B:=[4,5], C:=A+B, тогда C=[1,2,3,4,5].

Пересечение множеств ∩: «*».

Пусть A:=[1,2,3], B:=[1,4,2,5], тогда A*B=[1,2].

Пусть A:= ['A'..'Z'], B:= ["B"..'Y'], тогда A*B=['B'..'Y'].

— т. е. третье множество, которое содержит элементы, входящие одновременно в оба множества.

Разность множеств «-» (\- в математике) — третье множество, которое содержит элементы первого множества, не входящие во второе:

Пусть A := [1,2,3,4], B:= [3,4,1], тогда A-B=[2].

Пусть A := [x1,x2,x3,x4], B:= [x2,x3], тогда A-B=[x1,x4].

Преимущества использования типа set: значительно упрощаются сложные условия в операторе if, увеличивается степень наглядности, экономятся память, время компиляции и выполнения.

Отрицательные стороны: отсутствуют средства ввода-вывода элементов множеств.

Примеры:

1. В считанном с клавиатуры тексте из строчных латинских букв удалить все гласные буквы, а согласные заменить на прописные. Признак конца ввода — точка. Например: pointer —> PNTR.

```
Program Sets;
Type  let='a'..'z';
Var   gl : set of let;
      tx : string;
      i : byte;
Begin
      readln (tx); gl:=['a','e','i','o','u','y'];
```

```
        i:=1;
        Repeat
        While tx[i] in gl do delete(tx,i,1);
        tx[i]:=Upcase(tx[i]);
        Inc(i)
        Until tx[i]='.';
        Writeln(tx)
End.
```

2. Заполнить множество A путем ввода *n* значений:

```
Const  n=20;
Var    A : set of 1..200;
       j, x : byte;
Begin A:=[];
     For j := 1 to n do begin readln(x); A:=A+[x] end;
     For x := 1 to 200 do if  x in A  then writeln(x);
End.
```

3. Пусть введена последовательность 2,0,4,-2,1000,7,9,100,100,40. Вывести в порядке возрастания элементы множества.

```
Const   n=100;
Const d:set of byte=[1]; {типизированная константа}
Type   num=1..n;
Var    a, c : set of num;
       b : set of 1..10;
       x : integer;
       i, k : byte;
Begin
  k := 10; {Заполнить множество a путем ввода
            k значений}
  a := [ ];
  For i := 1 to k do
    Begin readln(x); a := a + [x]  end;
  For x := 1 to n do
    If x in a then write(x);
  Writeln;
End.
```

4. Что будет выведено на экран?

```
a) a := [1,2,4,6,7];
   b := [1,2,4,7];
   if a >= b then writeln('True');

b) x:=5;
   if x in [0..3,10..15,20..25] {Проверить принадлежность одному из интервалов} then writeln("True");
```

5. Какие элементы будут входить в множество *c* в результате следующих операций:

```
a) a := [1,4,7]; b := [2,4,7,15]; c := a * b;
b) a := [1,2,4,6,7]; b := [1,2,4,7]; c := a+b;
c) a := [1,2,4,6,7]; b := [1,2,4,7,10]; c := a - b;
d) c := b - a;
```

6. Проверить, является ли введенное с клавиатуры слово правильной записью идентификатора.

```
Type letter= set of char;
Var   ident : letter;
        wd : string[10];
        I : byte;
        fl : boolean;
Begin
  Readln(wd); fl := false;
  Ident := ['A'..'Z', 'a'..'z', '_'];
  If not (wd[1] in ident)
    Then fl := true  {ошибка}
    Else
      Begin
        Ident := ident + ['0'..'9'];
        For i := 2 to length(wd) do
           If not (wd[i] in ident) then fl:=true
                                          {ошибка}
      End;
  If fl then writeln('Не верный идентификатор!');
  Readln
End.
```

6.7.4. Комбинированный тип (записи)

▶ *Запись* — это структура данных, состоящая из фиксированного числа компонентов, называемых полями записи. В отличие от массива, поля записи могут быть различного типа. Чтобы можно было ссылаться на тот или иной компонент записи, поля именуются.

Объявление типа записи выглядит следующим образом:

TYPE <имя типа> = RECORD <Список полей> END;

Здесь <имя типа> — правильный идентификатор; <Список полей> — список полей; представляет собой последовательность разделов записи, разделяемых точкой с запятой. Каждый раздел записи представляет собой один или несколько идентификаторов полей, отделяемых друг от друга запятыми. За идентификаторами ставится двоеточие и описание типа поля.

Как было указано в п. 6.7.1, запись может быть элементом массива.

Пример:

```
Type rec = record
           a: integer;
           b: real;
           c: string[10]
           end;
Var    elem : rec;
       tabl : array[1..100] of rec;
```

Обращение к элементам записи осуществляется с помощью составных имен. Составное имя начинается с имени записи и содержит список имен полей (разделенных точками), входящих в цепочку, ведущую к требуемому элементу.

Например, к полям записи, описанной выше, можно обратиться по именам:

```
elem.a, elem.b, elem.c, elem.c[5],
```

а к полям *i*-го компонента массива записей — по именам:

```
tabl[i].a, tabl[i].b, tabl[i].c, tabl[i].c[1].
```

Для сокращения обозначения полей записи (когда ведется работа с несколькими полями одной и той же записи) используется оператор присоединения:

WITH <переменная запись> DO <оператор>;

Внутри оператора, входящего в оператор присоединения, компоненты записи обозначаются с помощью только имен полей (имя переменной-записи перед ними не указывается). Заголовок операторов может содержать список переменных-записей, разделенных запятыми:

WITH <переменная-запись_1>, ...<переменная-запись_N> DO <оператор>;

Примеры:

```
1. With elem do
Begin
  a:=1998; b:=37.5, c:='И.И. Петров'
End;

2. For i:=1 to 10 do
   With tabl[i] do begin read(a,b); readln(c) end;
```

Пример. Пусть запись содержит сведения о студенте и состоит из полей: личный номер студента, фамилия и инициалы, номер курса, номер группы, средние оценки за каждый год учебы. Необходимо ввести исходные данные.

```
{Ввод элементов записи}
Const  Nmax=30;
Type   Zap=Record
       nom : byte;
       fio : string[20];
       kurs, group : byte;
       ocen : array[1..5] of real;
       end;
  arr_zap = array[1..Nmax] of zap;
Var list:arr_zap;
    n:1..Nmax;
    i:integer;
Begin
  Write('Задайте количество студентов в списке ...');
  Readln(n);
  For i:=1 to n do
  With list[i] do
```

```
    Begin
      Write('Задайте личный номер студента');
      Readln(nom);
      Write('Задайте фамилию');
      Readln(fio);
      Write('курс и группа');
      Readln(kurs, group);
      Write('средние оценки за каждый год');
      For i:=1 to 5 do readln(ocen[i]);
      Readln
    End
End.
```

6.8. Типизированные константы

Типизированные константы задаются в разделе описания CONST следующим образом:

CONST < Имя > := < Тип > = < Значение >;

Таким образом, типизированные константы *инициализируются* некоторым начальным значением. Типизированная константа фактически ничем не отличается от переменной соответствующего типа. Это *переменные с начальными значениями*, которые присваиваются им только один раз (при инициализации).

Если типизированная константа была объявлена в некотором блоке, то при повторном входе в этот блок она сохраняет то значение, которое имела при выходе из блока.

Нельзя использовать типизированную константу в качестве значения при объявлении других констант или границ типа-диапазона.

Типизированные константы скалярных типов

Примеры:

```
Type week = (mn, tu, we, th, fr, st, sn);
                Const day: week = mn;
                        x: real = 0.25;
                        y: integer= 100;
                      min: byte= 0;
                      max: byte= 99;
```

Типизированные константы-массивы и строки

Строки задаются так же, как и скалярные типы:

Пример:

```
Const suname: string[20] = 'Баланс';
```

Массивы. В качестве их значений используется список констант C_n соответствующего типа, отделенных друг от друга запятыми, заключённый в круглые скобки: $(C_1, C_2,\ldots,C_n)$.

Примеры:

a)
```
Const
  CN: array[1..5] of integer = (5, 15, -38, 40, 4);
  name: array[1..4] of string[12] =
       ('Алла','Борис','Сергей','Юрий');
  symbols: array[1..4] of char = ('+', '-', '*', '/');
```

b)
```
Const
  N = 100;
Type
  NS = Set of 0..255;
  NB: array[1..N] of string[20] =
                  ('Грипп','ОРЗ','Корь'…);
  B: array[1..N] of NS=([17,19,95,96], [17,19,90],
[3,6,25]…);
{количество значений должно совпадать с количеством
элементов описываемого массива}
  math: array[1..3, 1..2] of byte
= ((3,5), (0,16), (14,9));
```

Типизированные константы-множества

Константы — элементы множеств задаются по правилам, описанным для множеств, то есть перечислением констант либо диапазоном, например:

```
a) Type NS = set of byte;
   const B1 : NS = [17,19,95,96]; B2: NS = [1,8,10,26];
b) Type letter = set of char;
   const L1 : letter = ['a'..'g','R','2','x'..'z'];
c) Type dig = set of 0..9;
   const d1: dig = [1..3,6..9]; d2: dig = [0,3,6,9];
```

```
d)
Program typ_con;
   Type hvor = set of 1..100;
    const B:array [1..5] of hvor=
([1,4..7,10,45],[44,56..60,78],[1,2,7],[12,14,16,19..21],
[8,9,11..14]);
   rus: array [1..5] of string [22] =
   ('грипп','ангина','коклюш','краснуха','корь');
Var a: hvor;
    i,j,k,simp: byte;
Begin
  a:= [ ];
  Repeat
    Write ('Очередной симптом'); readln (simp);
    If simp > 0 then a := a+[simp]
  Until simp = 0;
  j:=1;
  While (j<=5) and (a<>b[j]) do inc(j);
  If j <= 5
    Then writeln ('Выявлен(а)', rus[j])
    Else writeln ('Диагноз не установлен');
End.
```

Типизированные константы-записи

В них указываются имена и начальные значения всех полей в том порядке, в котором они следуют в описании типа.

Пример:

```
const n = 24;
type
  person = record
                   fio : string [20];
                   age: 17.. 22;
                   rost: byte
                 end;
  gruppa = array [1..n] of person;
const gr_1:gmuppa =
      ((fio: 'Иванов И.И.'; Age: 17; Rost:176),
      (fio: 'Петров П.П.'; Age: 18; Rost:164),
     (fio: 'Сидоров С.С.'; Age: 18;Rost: 175) … );
{как указывалось выше, количество перечисленных
записей должно совпадать с объявленным числом компо-
нентов массива}
```

Можно также описать типизированные константы процедурного типа, объектного и типизированные константы — указатели.

Для чего нужны типизированные константы? Они освобождают пользователя от ввода данных. Это особенно удобно на этапах отладки и модернизации программ. Типизированные константы позволяют сэкономить время и избежать ошибок.

6.9. Файлы

▶ *Файл* — поименованная область памяти на внешнем носителе, предназначенная для хранения информации. В файлах могут храниться программы, данные, тексты документов, изображения и т. д.

Преимущества использования файлов следующие:

1) данные, организованные в виде файла, могут использоваться в нескольких программах;

2) файл сохраняет свои значения по окончании работы программы;

3) файл — единственный способ размещения данных очень большого объема (если оперативная память не позволяет этого сделать).

Для того, чтобы использовать файл в программе на Паскале, необходимо выполнить следующие действия:

■ описать переменную файлового типа одним из способов (в зависимости от типа создаваемого файла):

```
f : FILE OF <тип>;     {типизированный файл}
f : TEXT;              {текстовый файл}
f : FILE;              {безтиповый файл}
```

■ поместить имя файла в переменную символьного типа (например, name);

■ связать файловую переменную f с именем файла name:

```
ASSIGN (f, name);
```

■ открыть файл для чтения / записи операторами соответственно:

```
RESET(f) или REWRITE(f);
```

■ открыть файл для дополнения (только для текстовых файлов):

```
APPEND(f);
```

■ читать / писать запись из файла / в файл, используя переменную (например, zap):

```
READ(f, zap) или WRITE(f, zap);
```

■ закрыть файл по окончании работы с ним:

```
CLOSE(f).
```

Для обнаружения конца файла используется функция логического типа EOF(f).

Одновременно могут быть открыты несколько файлов. В ходе выполнения программы один и тот же файл может быть открыт для записи, а затем использован для чтения. Открытый на запись файл изначально является пустым, он содержит лишь маркер конца файла. Каждый оператор write или writeln осуществляет добавление новой информации, после чего маркер сдвигается к новому концу файла. Оператор writeln (в отличие от write) добавляет в файл литеру конца строки. По смыслу, маркер конца файла — это следующая доступная компонента, в которую будет помещен следующий элемент (если он есть).

6.9.1. Типизированные файлы

Типизированными (двоичными) файлами называются дисковые файлы, состоящие из нумерованной последовательности записей (компонент) одинакового типа. Тип записей в файле задается при его объявлении. Длина каждой записи постоянна. Можно определить позицию каждой записи в файле и напрямую считать (или записать) эту запись.

Типизированные файлы полезны для временного хранения информации в процессе выполнения программы или для передачи большого объема промежуточных данных, полученных в одной программе, другой программе.

Примеры:

1. Создать файл, состоящий из записей с полями: ф.и.о. студента, номер курса и номер группы. Признаком конца вводимых записей будем считать пустую строку (пустая фамилия).

```
{Создание файла}
Type rec = record
   fio: string[20];
   kurs, group: integer;
End;
Var
  zap: rec;
  f: file of rec;
  name: string;
Begin
  Writeln('Задайте имя файла');
  Readln(name);
  Assign(f,name);
```

```
    {связывает файловую переменную с конкретным именем}
     Rewrite(f); {создает новый пустой файл}
     Write ('Введите фамилию и.о.'); readln (zap.fio);
     While zap.fio <> '' do
       Begin
         Write ('курс и группа   ');
         Readln (zap.kurs, zap.group);
         Write(f, zap);
   {занесение содержимого записи zap в файл в двоичном
коде}
         Write ('фамилия и.о.');
         Readln (zap.fio)
       End;
     Close (f);
     Writeln('Файл создан')
   End.
```

2. Выполнить просмотр ранее созданного файла, выдав на экран те записи, фамилии которых начинаются и заканчиваются на одну и ту же букву.

```
   {Обработка файла}
   {фрагмент программы}
   Reset (f);  {открыть файл для работы с ним}
   While not eof(f) do
        {проверить, не достигнут ли конец файла}
     Begin
     Read(f,zap); {читать запись из файла в переменную zap}
     With zap do
     begin
      n:=length(fio);
       if fio[1]=fio[n] then writeln(fio:20, kurs:3,
group:3)
     End
   End;
```

3. Дополнить ранее созданный файл. Для этого необходимо воспользоваться следующими стандартными процедурами и функциями:

SEEK (f, n) — установить указатель файла на компоненту с номером n.

Указатель перемещается к компоненте с номером n, начиная счет с нуля, т. е. первая компонента файла имеет номер 0, вторая — 1 и т. д.

FILESIZE(f) — определить количество компонент в файле.

```
{Добавление записей (в конец файла)}
Reset (f);
Seek (f, filesize(f));
{установить указатель за последней компонентой файла}
Writeln('задайте фамилию и. о. '); readln (zap.fio);
While zap.fio < > '' do
  Begin
   Write('курс и группа'); readln(zap.kurs,
zap.group);
   Write (f,zap);
   Write ('фамилия и.о. '); readln (zap.fio)
  End;
Close(f);
```

6.9.2. Текстовые файлы

Текстовые файлы состоят из символов, объединенных в строки. Длина строки текстового файла переменная (от 0 до 255 символов). В конце каждой строки файла размещается признак конца строки: это последовательность кодов ASCII — 13 (CR) и 10 (LF). В конце всего файла — признак конца файла: код ASCII — 26 (CTRL — Z).

Текстовый файл, в отличие от типизированного, является последовательным. Он может быть открыт для записи, чтения или дополнения.

Для доступа к записям текстового файла используются процедуры read, readln, write, writeln. В них можно указывать переменное число параметров. Параметры могут иметь тип: integer (или другой целочисленный), real, char, string, boolean (последний — в процедурах вывода).

Формат определения операторов ввода-вывода:

READ[LN] ([f,] <список ввода>);

WRITE[LN] ([f,] <список вывода>);

здесь — f — имя файловой переменной.

Элементы, заключенные в квадратные скобки, не являются обязательными в операторах ввода-вывода. Если файловая переменная указана, осуществляется обращение к дисковому файлу или к логическому устройству. Примеры логических устройств: CON —клавиатура или экран дисплея; PRN — принтер.

Если файловая переменная не указана, происходит обращение к стандартным файлам INPUT или OUTPUT (чт о соответствует вводу с клавиатуры или выводу на экран дисплея).

Для работы с текстовыми файлами используются следующие стандартные логические функции:

EOLN(f) — возвращает значение true, если в файле достигнут маркер конца строки, false — в противном случае.

SEEKEOLN(f) — пропускает пробелы и знаки табуляции до маркера конца строки или до первого значащего символа и возвращает значение true, если маркер обнаружен, false — в противном случае.

Пример. Создать текстовый файл из произвольного числа строк; добавить еще 3 строки; вывести файл на экран.

```
Program fil_tex;
Var f:text;
    name,w:string;
    n,i:byte;
Begin
{Создание файла}
Readln(name);
Assign(f,name);
Rewrite(f);
Readln(w);
While w<>'' do { Ввод строк, пока не введена пустая }
  Begin
  Writeln(f,w);
  Readln(w)
  End;
Writeln('файл создан');
Close(f);
{Добавление файла}
Append(f);
For i:=1 to 3 do
  Begin
  Readln(w);
  Writeln(f,w);
  End;
Writeln('файл добавлен');
```

```
Close(f);
{Вывод содержимого файла на экран}
reset(f);
writeln('вывод файла');
while not eof(f) do
  begin
  readln(f,w);
  writeln(w)
  end;
Close(f);
readln
end.
```

6.10. Подпрограммы

▶ *Подпрограмма* — обособленная, сформированная в виде отдельной синтаксической конструкции и снабженная именем часть программы.

Использование подпрограмм позволяет, подробно описав в них некоторые операции, в остальной программе указывать только имена подпрограмм, чтобы выполнить эти операции.

Такие вызовы подпрограмм возможно осуществлять неоднократно из разных участков программы, причем при вызове подпрограмме можно передать некоторую информацию (различную в различных вызовах, чтобы одна и та же подпрограмма могла выполнять решения для разных случаев).

Повышение сложности задач, решаемых с помощью ЭВМ, приводит к увеличению размеров и сложности программ, следовательно, возникают дополнительные трудности при разработке и отладке. Увеличение продолжительности жизненного цикла программ приводит с течением времени к необходимости их модификации (с целью повышения их эффективности и удобства пользования ими). Для разрешения возникших при этом проблем в практике программирования выработан ряд приемов и методов структурного программирования.

Под *структурным программированием* понимают такие методы разработки и записи программы, которые ориентированы на максимальные удобства для восприятия и понимания ее человеком. При прочтении программы в ее фрагментах должна четко прослеживаться логика ее работы, т.е. не должно быть «скачков». Структурное программирование — программирование «без goto», т.е. не используются операторы

перехода без необходимости. В связи с этим отдельные фрагменты программы представляют собой некоторые логические (управляющие) структуры, которые определяют порядок выполнения содержащихся в них правил обработки данных. Любая программа получается построенной из стандартных логических структур, число типов которых невелико.

Основные логические структуры: *следование, ветвление, повторение* (каждая имеет один вход и один выход).

Простота и надежность программы существенно зависят от того, насколько удобно обрабатывать данные и правила их обработки и как они объединены в логические структуры.

Решение отдельного фрагмента сложной задачи может представлять собой самостоятельный программный блок — подпрограмму.

6.10.1. Процедуры и функции

Язык Паскаль называется процедурно-ориентированным за наличие подпрограмм как средства структурирования программы. Подпрограммы в Паскале реализованы посредством процедур и функций. Имея один и тот же смысл и аналогичную структуру, процедуры и функции различаются назначением и способом использования.

▶ *Процедура* — независимая именованная часть программы, которую можно вызвать по имени для выполнения определенных действий. Структура процедуры повторяет структуру программы. Процедура не может выступать как операнд в выражении. Например, Writeln — встроенная процедура Паскаля.

▶ *Функция* — аналогична процедуре, но имеются 2 отличия:

1) функция передает в точку вызова скалярное значение (возвращает значение);

2) имя функции может входить в выражение как операнд.

Например, ArcTan(x: real): real — передает в точку вызова arctg(x).

Вызов процедуры или функции — указание ее имени в тексте программы, приводящее к ее активизации.

Все подпрограммы Паскаля делятся на две группы: встроенные (стандартные) и определенные пользователем.

Все стандартные средства расположены в специализированных библиотечных модулях, основные из которых следующие:

System — содержащиеся в нем подпрограммы обеспечивают работу всех остальных модулей системы. Подключается к программе автоматически, поэтому его имя не указывается в разделе Uses. Поэтому любой программе всегда доступны его процедуры и функции.

Crt — средства управления монитором и клавиатурой;
Dos — средства Dos;
Printer — быстрый доступ к печатающему устройству;
Graph — пакет графических средств.

Процедуры и функции пользователя

Если в программе возникает необходимость частого обращения к некоторой группе операторов (выполняющих действия или вычисляющих значение выражения), то рационально выделить такую группу операторов в самостоятельный блок, к которому можно обращаться, указав его имя. Такие разработанные программистом самостоятельные программные блоки называются *подпрограммами пользователя.*

Они являются основой *модульного программирования*. Разбивая задачу на части, формируя логически обособленные модули как процедуры и функции, программист реализует основные принципы системного подхода и методов нисходящего программирования.

При вызове подпрограммы (процедуры или функции), определенной программистом, работа главной программы на некоторое время приостанавливается, и начинает выполняться вызванная подпрограмма. Она обрабатывает данные, переданные ей из главной программы. По завершении подпрограммы функция возвращает главной программе результат. Передача данных из главной программы в подпрограмму и возврат результата функции осуществляется с помощью параметров.

Параметром называется переменная, которой присваивается некоторое значение. Различают *формальные параметры* — определенные в заголовке подпрограммы и *фактические параметры* — выражения, задающие конкретные значения при обращении к подпрограмме.

При обращении к подпрограмме ее формальные параметры заменяются фактическими, переданными из главной программы.

Название «формальные» эти параметры получили в связи с тем, что они задают только имена для обозначения исходных данных и результатов работы подпрограммы. При вызове же подпрограммы на их место будут подставлены конкретные значения.

Подпрограмма определяется в разделе описания процедур и функций программы.

Формат описания процедуры:

PROCEDURE <имя процедуры> [(список формальных параметров)];

Формат описания функции:

FUNCTION <имя функции> [(список формальных параметров)]:
<тип результата>;

Здесь имя процедуры или функции — идентификатор; список формальных параметров — последовательность имен формальных параметров, их типов и способов подстановки, отделенных друг от друга точкой с запятой.

Описание формальных параметров может отсутствовать.

Блок (тело) подпрограммы имеет ту же структуру, что и блок, являющийся телом программы, то есть начинается зарезервированным словом Begin и заканчивается словом End.

При написании обращений к процедуре и к функции необходимо обеспечить соответствие фактических и формальных параметров по количеству, типу и порядку следования. В теле функции должен присутствовать хотя бы один оператор присваивания с именем этой функции в левой части. Функция может возвращать в качестве результата значение скалярного, строкового или ссылочного типа.

Пример. Описать подпрограмму, определяющую возможность построения треугольника по трем сторонам a, b и c и вычисляющую его площадь.

Вариант 1 — процедура:

```
Procedure TRIANGLE (a,b,c:real; var s:real);
Var p: real;
Begin
  If (a+b > c) and (a+c > b) and (b+c > a)
  {если треугольник существует}
    Then
      Begin
     p := (a + b + c) / 2;
     p := p * (p - a) * (p - b)*(p - c);
     s := sqrt(p);
      End
    Else s := 0
End;
```

Тогда значение площади треугольника может быть использовано в алгоритме, например, следующим образом:

```
{фрагмент программы}
TRIANGLE(2,3,4,Q);
If Q <> 0 then p:=5.7 + 2*Q;
```

```
Вариант 2 — функция.
Function TRIANGLE(a,b,c:real):real;
Var p:real;
Begin
  If (a+b > c) and (a+c > b) and (b+c > a)
    Then
      Begin
    p := (a + b + c) / 2;
    p := p * (p - a) * (p - b)*(p - c);
    If p > 0 then TRIANGLE := sqrt(p) else TRIANGLE := 0
      End;
End;
```

В этом случае два оператора в разделе операторов программы примут вид:

```
Q := TRIANGLE(2,3,4);
If Q <> 0 then p:=5.7 + 2*Q;
```

6.10.2. Параметры подпрограмм

В заголовке программы необходимо указать способ подстановки фактических параметров. Принято различать два способа подстановки параметров:

- подстановка значения (параметр-значение);
- подстановка переменной (параметр-переменная).

Параметры-значения передаются основной программой в подпрограмму через стек в виде их копий, поэтому фактический параметр подпрограммой измениться не может.

Параметры, которые называют *параметрами-переменными*, указываются заданием зарезервированного слова Var перед их идентификаторами в списке формальных параметров. При передаче параметров-переменных в подпрограмму фактически через стек передаются их адреса в порядке, объявленном в заголовке подпрограммы. Следовательно, подпрограмма имеет доступ к этим параметрам и может их изменять.

Входные параметры подпрограммы могут быть как параметрами-значениями, так и параметрами-переменными. Выходные (модифицируемые) — только параметрами-переменными.

Фактическими параметрами, соответствующими параметрам-значениям, могут быть имена переменных, константы, выражения. Фактичес-

кими параметрами, соответствующими параметрам-переменным, — только имена переменных.

Пример. Отпечатать таблицу значений суммы:
$\sum_{i=1}^{m} \frac{1}{i}$ для $m = 1, 2, \ldots, 1024$.
Опишем процедуру вычисления суммы S.

```
Program Summa;
Const n=1024;
Var x: real; m: integer;
Procedure Sum(n: integer; var S:real);
  Var I : integer;
  Begin
   s:=0;
   For i := 1 to n do S := S+1/i
  End;
Begin
   For m := 1 to n do
     Begin
       sum(m, x); write(m, x)
     End
End.
```

Здесь в списке параметров процедуры Sum n — параметр значение, S — параметр-переменная.

6.11. Рекурсии

6.11.1. Рекурсивные алгоритмы и рекурсивные определения

Программист обычно разрабатывает программу, сводя исходную задачу к более простым. Среди этих задач может оказаться и первоначальная, но в упрощенной форме.

Например, вычисление функции $F(N)$ может потребовать вычисления $F(N — 1)$ и еще каких-то операций. Иными словами, частью алгоритма вычисления функции будет вычисление этой же функции.

▶ Алгоритм, который является своей собственной частью, называется *рекурсивным*. Часто в основе такого алгоритма лежит рекурсивное определение какого-то понятия.

Например, о факториале числа N можно сказать, что
(Определение 1) $N! = 1 \times 2 \times \ldots (N-1) \times N$ или 1, если $N = 0$;
(Определение 2) $N! = (N-1)! \times N$, или 1, если $N = 0$.
Второе определение рекурсивное.

Любое рекурсивное определение состоит из 2-х частей. Одна часть определяет понятие через него же, другая часть — через иные понятия.

Записать рекурсивный алгоритм на Паскале можно с помощью рекурсивной процедуры (функции).

6.11.2. Рекурсивные процедуры и функции

▶ *Рекурсия* — это способ организации вычислительного процесса, при котором процедура или функция в ходе выполнения составляющих ее операторов обращается сама к себе (прямо или косвенно, через другие процедуры).

Рекурсивная процедура осуществляет многократный переход от некоторого текущего уровня организации алгоритма к более низкому последовательно до тех пор, пока не будет получено тривиальное решение поставленной задачи.

Рассмотрим программу с рекурсивным вызовом на примере факториала: ($n! = 1 \times 2 \times 3 \times \ldots \times n$).

```
Program   rec_1;
     Var f : longint ;
Function fact (f : integer) : longint ;
{описание функции, f – формальный параметр, значение типа
 inteqer, результат функции – типа longint}
Begin
     If f = 0 then fact := 1
     Else fact := f * fact (f - 1)
End;
{ Начало основной программы }
Begin
     Write ('Введите число f > 0');
     Readln (f);
     If  f > 0 then
  writeln('Для числа ' ,f,' значение факториала =',
fact(f))
```

```
        Else writeln("Число неверное !")
end.
```

Использование рекурсивной формы организации алгоритма дает более компактный текст программы, но выполняется медленнее и может вызвать переполнение стека.

▶ *Стек* — это специальным образом организованная область памяти, в которой размещаются при каждом входе в подпрограмму ее локальные переменные.

Вызов процедурой самой себя (рекурсивный вызов) ничем не отличается от вызова другой процедуры.

Что происходит, если одна процедура (или программа) вызывает другую? В общих чертах следующее:

1) в памяти размещаются параметры, передаваемые процедуре (но не параметры-переменные);

2) в другом месте памяти сохраняются значения внутренних переменных вызывающей процедуры;

3) запоминается адрес возврата в вызывающую процедуру (или программу);

4) управление передается вызванной процедуре.

Если процедуру (функцию) вызвать повторно из другой процедуры или из нее самой, то будет выполняться тот же алгоритм, но работать он будет с другими значениями параметров и внутренних переменных. Это и дает возможность рекурсии.

Пример. Пусть рекурсивная функция step(a : real ; n : inteder) : real; возводит число a в степень n (a^n). Вычислить : $Z = X^k + Y^m$

```
Program rec_2;
Var    x,y,z:real;
       k,m:integer;
Function step(a:real; n:integer):real;
Begin
 if n = 0 then step := 1
 else step := a * step(a, N-1)
End;
Begin
  Readln(x,k); readln(y,m);
  z := step(x,k) + step(y,m);
   Writeln(z:0:3)
End.
```

Возникает вопрос: «Можно ли использовать рекурсивные процедуры с бесконечным «самовызовом»? Нет, так как не существует бесконечной памяти! Следовательно, условие, по которому выполняется вызов процедуры, должно на некотором уровне рекурсии стать ложным.

Пока условие истинно, рекурсивный спуск продолжается. Когда условие становится ложным, спуск заканчивается и начинается поочередный рекурсивный возврат из всех вызванных на данный момент копий рекурсивной процедуры.

6.11.3. Виды рекурсивных процедур

В общем случае рекурсивная процедура *Recur* включает в себя некоторое множество операторов *Op* и один или несколько рекурсивных вызовов *Recur*.

1. Действия выполняются на рекурсивном спуске (до рекурсивного вызова):

```
Procedure Recur;
  Begin
    Op;
    If условие then Recur [else Recur];
  End;
```

2. Действия выполняются на рекурсивном возврате (после рекурсивного вызова):

```
Procedure Recur;
  Begin
    If условие then Recur [else Recur];
    Op;
  End;
```

1. Действия выполняются как на рекурсивном спуске, так и на рекурсивном возврате:

а)
```
Procedure Recur;
Begin
 Op1;
 If условие then Recur;
 Op2 ;
End.
```

б)
```
Procedure Recur;
Begin
If условие then
 Begin
   Op1; Recur; Op2
 End;
 End.
```

Все виды практически используются. Причем есть классы задач, при решении которых программисту требуется сознательно управлять ходом рекурсивных процедур и функций.

Примеры рекурсивных подпрограмм.

1. Описать рекурсивную функцию digit без параметров, которая подсчитывает количество цифр в тексте, заданном во входном файле (за текстом следует точка).

```
Program rec_4;
Var  k:integer;
Function digits:integer;
  Var
    c:char;
    d:integer;
Begin
  Read (c);
  d := 0;
  If c<> ' . '
    Then
      If (c <= '9') and (c >= '0')
        Then d := 1 + digits
        Else d := digits;
  digits := d;
End;
Begin
  Write ('Введите текст, последний символ – точка ');
  k := digits;
  writeln(k);
end.
```

Напечатать в обратном порядке заданный во входном файле текст (за текстом следует точка).

```
Program rec_5;
  Procedure perevert;
    Var
      c:char;
    Begin
```

```
        Read (c);
        if c <> '.' then perevert;
        write(c);
      End;
Begin
  Write('Введите текст');
  Perevert;
End.
```

6.12. Программные модули

Известно, что при написании серьёзной программы нельзя обойтись без процедур, которые заключают в себе её отдельные части. Крупными строительными единицами в Паскале являются *программные модули*.

Модуль имеет имя и может содержать описание многих процедур и функций, а также описания констант, типов данных и переменных. Каждый программный модуль транслируется отдельно, оттранслированные модули объединяются в выполнимую программу — этот процесс называется «сборкой».

Однажды написанный и оттранслированный модуль можно многократно использовать в различных программах. Это позволяет:

- сэкономить время и силы программиста;
- сократить время трансляции;
- уберечь от искажений исходный текст модуля.

Чтобы подключить модуль к программе и сделать видимым его содержимое, достаточно упомянуть его имя в разделе USES <имя модуля> (должно быть первым предложением программы).

Необходимость использования модулей обусловлена следующими причинами:

- для размещения в памяти большой программы может не хватить одного сегмента памяти (его размер максимум 64 Кб). Количество используемых модулей ограничивается лишь доступной памятью;
- в большинстве реальных применений ЭВМ нужны библиотеки блоков (процедур и функций) с простым доступом к блокам.

▶ *Модуль* — это автономно компилируемая программная единица, включающая в себя различные компоненты раздела описаний и, возможно, некоторые исполняемые операторы инициирующей части.

6.12.1. Структура программного модуля

UNIT <имя модуля> (заголовок модуля)

. . . .

INTERFACE — интерфейсные раздел («видимая» часть модуля)

. . . .

IMPLEMENTATION — раздел реализации («черный ящик»)

. . . .

[BEGIN] — раздел инициализации (необязательный)

. . . .

END.

Имя модуля должно совпадать с именем дискового файла, в который помещается исходный текст модуля. Имя модуля служит для его связи с другими модулями и основной программой. Эта связь устанавливается предложением UNIT USES <список модулей>.

В основной программе с USES должен начинаться раздел описаний.

В модулях USES может следовать сразу за зарезервированным словом INTERFACE, либо за IMPLEMENTATION (либо и там, и там).

Интерфейсный раздел содержит объявления всех глобальных объектов, констант, переменных, типов, подпрограмм модуля, которые должны быть доступны основной программе и другим модулям. При объявлении глобальных подпрограмм в интерфейсной части указывается только их заголовок. Зачем? Без информации о формальных параметрах блоков нельзя правильно «собрать» программу из модулей.

Описанные здесь объекты необязательно должны использоваться в блоках данного модуля. Например, модуль может не содержать ни одного блока, а включать в себя описание сложных типов, используемых во многих программах пользователя. Это сокращает размер программы и ускоряет подготовку исходных текстов.

Например:

```
UNIT M1;
Interface
  Var  x:integer;
  Procedure Sum(a,b: integer, var s: integer);
  . . .
```

Если теперь в программе написать предложение USES M1;, то в основной программе станет доступным переменная X и процедура Sum.

Раздел реализации — в него помещают блоки, заголовки которых приведены в интерфейсной части, и вспомогательные (локальные для модуля) объекты (типы, константы, переменные, подпрограммы), используемые только в данном модуле. Эти объекты недоступны другим модулям и основной программе.

Подпрограммы, объявленные в интерфейсной части модуля, в разделе реализации должны содержать заголовок, в котором могут быть опущены список формальных параметров и тип результата (для функций).

Например:

```
. . .
Implementation
  Procedure  Sum;
    Begin
      S:= a + b
    End;
```

Раздел инициализации завершает модуль, может отсутствовать (тогда нет и BEGIN) или быть пустым. Описывает «разовые» подготовительные действия, восполняемые при загрузке программы в память. Содержит исполняемые операторы, которые выполняются до передачи управления основной программе и обычно используются для подготовки её работы. Если модулей несколько, то операторы данного раздела выполняются в порядке указания имён в USES. Например, здесь могут инициализироваться (задаваться начальными значениями) переменные, открываться файлы и т. д.

Пример:

```
{Модуль}
UNIT M1;
Interface
  Var x:integer;
  Procedure Sum(a,b:integer; var s:integer);
Implementation
  Procedure Sum;
    Begin S := a + b;
    End;
end.
```

```
{Основная программа}
Program main;
Uses M1;
Var  y,z:integer;
Begin
  Readln(x,y);
  sum(x,y,z);
  Writeln(z)
end.
```

Модуль следует хранить в одноимённом файле с расширением PAS.

6.12.2. Трансляция модуля. «Сборка» программы

Результатом трансляции модуля является файл с тем же именем и расширением TRU. Он должен записываться на диск, тогда как результат трансляции программы в целом (EXE — файл) может оставаться в оперативной памяти.

Интерфейсная часть нужна для того, чтобы при получении файла проверить правильность обращения к блокам модуля.

В интерфейсной среде Паскаля в главном меню следует использовать подменю *Compile*. Пункты этого меню :

Compile (Alt — F9)
Make F9
Build
Destination Memory
Primary file

Первые три предназначены для запуска трансляции.

I. Если использовать *Compile*, то необходимо установить «Destination Disk», оттранслировать модуль, затем оттранслировать программу. Файл с расширением TRU должен находиться в каталоге, указанном в опции UNIT DIRECTORIES.

Пункты Make и Build — удобнее.

II. *Make* — для каждого модуля проверяет:

- существование TRU-файла. Если его нет, то он создаётся путём трансляции исходного текста модуля;
- соответствие TRU-файла исходному тексту модуля (в него могут быть внесены изменения, в этом случае TRU-файл создаётся заново автоматически);

• неизменность интерфейсного раздела модуля (в противном случае перекомпилируются все модули, в начале которых данный модуль указан в разделе USES).

Эти пункты не требуют обязательных исходных текстов модулей.

III. *Build* — в отличии от Make, требует наличия исходных текстов всех модулей, так как все они компилируются (дольше!), в остальном совпадает.

IV. *Run* (выполнение).

В случаях *II* — *IV* программа и модули транслируются совместно. Активным должно быть окно основной программы. (Иначе — имя файла основной программы должно быть указано в опции Primary file).

В общем случае, ссылки модулей друг на друга могут образовывать сложные структуры.

6.12.3. Ссылки на модули

Модуль как программа может использовать объекты, описанные в других модулях. Если эти объекты находятся в интерфейсном разделе, то он должен начинаться со слова USES, в котором перечисляются имена всех модулей.

Если на объекты есть ссылки только из раздела реализации, предложение USES может находиться в разделе реализации.

В общем случае, ссылки модулей друг на друга могут образовывать сложные структуры (рис. 6.2). При этом *запрещается* явное или косвенное обращение модуля к самому себе.

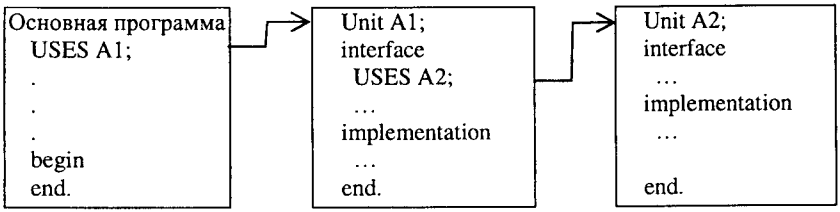


Рис. 6.2

Взаимные ссылки двух модулей возможны только из разделов реализации (рис. 6.3).

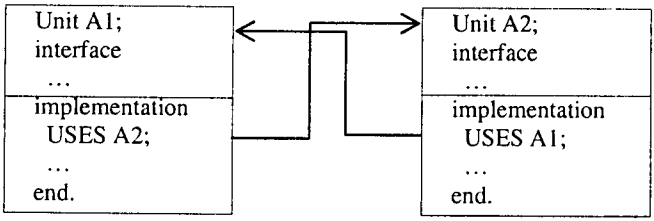


Рис. 6.3

Пример:

```
  {Модуль hlp_sr – вспомогательные расчёты}
UNIT hlp_sr;
Interface
USES crt;
  Var a,b,c:real;
 {глобальные переменные будут «видны»
  основной программе}
  Function S(x1,y1,x2,y2,x3,y3:integer):real;
{функция расчёта треугольника с заданными вершинами}
Implementation
  Var p: real;     {локальная переменная}
  Function r(x1,y1,x2,y2:integer):real;
{локальная функция расчёта расстояния между
 двумя точками}
    Begin
      R := SQRT(SQR(x1 - x2) + SQR (y1 - y2))
    End;
  Function S; {можно не описывать параметры}
    Begin
      a:= r (x1, y1, x2, y2); b:= r (x1, y1, x3, y3);
      c:= r (x3, y3, x2, y2); p:= (a + b + c)/2;
      S:= SQRT(p*(p – a)*(p – b)*(p – c));
    End;
End.
{Основная программа}
Program main;
USES  hlp_rs;
   Var xa, ya, xb, yb, xc, yc: integer;
   S_ tr: real;
Begin
  Readln (xa, ya, xb, yb, xc, yc);
```

```
  S_tr:= S;
  Writeln(a, b, c);
          {a,b,c – могут быть далее использованы }
  Writeln(S_tr);
End.
```

6.13. Динамическая память

В любой вычислительной системе память относится к тем ресурсам, которых всегда не хватает. Управление памятью — одна из главных забот программиста, так как *во время выполнения программы память необходима для следующих элементов программ и данных:*

- сама программа пользователя;
- системные программы, которые осуществляют вспомогательные действия при выполнении программ пользователя;
- определяемые пользователем константы и структуры данных (переменные);
- временная память для хранения промежуточных результатов при вычислении выражений;
- адреса возврата для подпрограмм;
- временная память при передаче параметров;
- буферы ввода/вывода, в которых хранятся данные между моментом их реальной передачи с периферийного устройства или на него и моментом инициации в программе операций ввода или вывода;
- различные системные данные (например, информация о статусе устройств ввода/вывода).

Таким образом, управление памятью касается широкого класса объектов. До сих пор мы пользовались простейшим способом распределения памяти — статическим.

Практика программирования часто выдвигает на передний план два конфликтующих фактора: время выполнения программы и объем занимаемой памяти. И иногда для экономии памяти приходится поступиться быстродействием. Статическое распределение памяти эффективно потому, что на управление памяти в этом случае не тратится ни время, ни память. Но рассмотрим пример. Пусть в некоторой задаче обрабатывается матрица размером 300×300 целых чисел. Тогда необходимо описание: var a1: array[1..300,1..300] of integer. Такие переменные, описанные в разделе var, Н. Вирт назвал *статическими* за то, что они могут обрабатываться компилятором без выполнения программы, на основа-

нии одного только статического (неизменного) текста программы. В нашем примере общее количество элементов 90000, необходимый объем памяти равен 90000×2 = 180000 байт. При этом маловероятно, что программе могут одновременно понадобиться все 90000 элементов. Кроме того, все переменные, объявленные в программе, располагаются в одной непрерывной области памяти «сегменте данных», длина которого (определяется архитектурой микропроцессора) равна 64 кбайта, что также вызывает затруднения при обработке больших массивов.

Какой может быть выход? Наиболее рациональное решение заключается в том, чтобы не резервировать заранее максимальный объем памяти для размещения данных, а, определив тип данных, создавать новый экземпляр данных всякий раз, когда в нем возникает необходимость.

▶ Переменные, которые создаются и уничтожаются во время выполнения программы, называются *динамическими*.

Помимо экономии памяти, динамические переменные позволяют решать задачи, для которых невозможно предсказать размер памяти в момент написания программы. Необходимость в динамическом распределении памяти возникает, например, при разработке систем автоматизированного проектирования (т. к. размеры различных матричных моделей могут существенно отличаться); при работе с графическими и звуковыми средствами ПК.

Итак, работа с переменными в различных языках программирования может быть организована различными способами:

I. *Статический*: место для хранения переменных отводится компилятором. В программе обращение к переменной происходит по имени, а компилятор связывает это имя с конкретным адресом в памяти. Эта связь постоянна в течение всего времени выполнения программы. В Паскале таким образом можно объявлять переменные размером до 64 кбайт.

II. *Динамический*: для хранения переменных выделяется общий участок памяти, а выделение памяти под конкретные переменные проводится на этапе выполнения программы. Участок оперативной памяти, в котором создаются динамические переменные, часто называют *кучей*.

III. *Совмещение статической и динамической памяти*: выделение места для локальных параметров подпрограмм.

▶ *Динамическая память* — это оперативная память ПК, предоставляемая программе при ее работе за вычетом сегмента данных (64 Кбайта), стека (также до 64 кбайт) и размера самой программы. Распределение памяти показано на рис. 6.4.

старшие адреса	системные программы	HEAPEND ←———
	еще не распределенная память --- динамическая память (расширяется в сторону увеличения адресов)	HEAPPTR ←——— HEAPORG ←———
до 64 кбайт	сегмент стека (расширяется в сторону уменьшения адресов) --- свободная память стека	
до 64 кбайт	сегмент данных основной программы (глобальные переменные, тип const)	
до 64 кбайт каждый	сегменты кодов включаемых модулей каждый	
до 64 кбайт	сегмент кода основной программы	
младшие адреса памяти	Паскаль системные программы	

Рис. 6.4

6.13.1. Указатели

Все рассмотренные ранее типы данных содержат непосредственные данные, размещенные в памяти ПК. Для организации динамической памяти применяются особые переменные, называемые указателями.

▶ *Указатель* — переменная, значением которой является адрес (первого байта памяти), по которому записаны данные.

Указатель занимает в памяти 4 байта. При этом данные, на которые он указывает, могут простираться на десятки килобайт.

К переменной, на которую указывает указатель, можно обращаться через имя этого указателя (то есть можно «ссылаться» на нее посредством указателя, отсюда и название — *ссылочная переменная*).

Операции с указателями

Тип — это всегда два множества: множество значений и множество операций. Значениями переменных типа указатель являются адреса памяти. К указателям применимы операции отношения, их можно сравнивать (равны или неравны, но не меньше или больше). Указатели равны только в случае, если они ссылаются на один и тот же объект, то есть содержат один и тот же адрес.

Существует операция, которая возвращает указатель — адрес первого байта памяти, по которому записана переменная (любого типа) — это:

@<Имя переменной> или функция ADDR (<имя переменной>).

Например, @x или ADDR(x).

Различают типизированные и нетипизированные указатели.

Нетипизированные указатели

Таким указателям можно присваивать адрес любой переменной независимо от типа. Нетипизированный указатель обозначается *pointer*. Используется для данных, структура и тип которых меняются в ходе выполнения программы.

Пример:

Пусть `var p,r:pointer; x,y:integer; st:string;`

```
а) p := @x; r := @st; тогда p <> r - true;
б) x := y; p := @x; r := @y; тогда p <> k - true;
в) p := @x; r := @x; тогда p = r - true.
```

Пример «указатель на указатель»:

```
Type ip:^integer;
Var a:ip; a1:^ip;
    x:integer;
Begin
  a:=@x; a1:=@a; a1^^:=10;
  Writeln(x) {10}
End.
```

Типизированные указатели

Типизированные указатели содержат адрес, по которому записана переменная заранее определенного типа.

Описываются они следующим образом: ^<идентификатор>;

Пример:
```
Type ip=^integer; rp=^real; и т.д.
     Var a,b:ip; c:rp; d:pointer;
         g:^byte; s:^string; a1,b1:^ip;
```

В Паскале все идентификаторы должны быть описаны перед использованием. Указатели являются единственным исключением. Базовый тип может быть объявлен и сразу после указателя.

Например:
```
Type RecP=^spis;
  spis=record
    fio:string[20];
    number:byte;
  End;
```

«Пустой» указатель

«*Пустой*» указатель — это постоянная указательного типа. Обозначается ***Nil***. Выделяется один адрес, в котором заведомо не может быть размещена никакая переменная. На это место и ссылается «нулевой» или «пустой» указатель.

Указатель, которому присвоено значение Nil, не содержит в себе никакого адреса. Указатель Nil считается постоянным, совместимым с любым ссылочным типом. Значит, его значение можно присваивать любому указателю. Nil используют для инициализации указателя «пустым» значением или когда его указание надо отменить. Это позволяет проверять значение указателя, прежде чем присвоить ему какое-либо значение.

6.13.2. Создание и уничтожение динамических переменных

Создание и удаление динамических переменных — это основные действия над динамическими переменными.

Типизированные объекты

NEW(x) — создает динамическую переменную, запрашивая в куче место для переменной соответствующего типа и возвращает адрес (x— переменная типа указатель).

Например:
```
Type vec = array[1..5] of integer;
     Var
       s:^vec;
```

```
    p,q:^real;
    x:real;
. . . . . .
 New(p);
 New(s);
```

Для задания значения переменной, на которую ссылается указатель, необходимо указать символ "^" справа от указателя:

<Имя переменной>^ := <Значение>;

Примеры:

```
1. p^:=1.125; g:=p; write(g^); — будет напечатано
число 1.125.
2. p:=@x; p^:= 10; — в результате в x окажется число 10.
3. s^[1]:= 3; for i:= 2 to 5 do s^[i]:=random;
```

Если переменная больше не нужна, то можем ее уничтожить, вернув память в кучу.

DISPOSE(x), x — типизированный указатель. Обычно используют для типизированных указателей.

Создание и уничтожение бестиповых объектов

В этом случае мы можем запросить у кучи любое заданное количество байт и адрес начала этой области присвоить переменной типа pointer (но не байт одного сегмента).

GETMEM (p: pointer; size: word) — создает новую динамическую переменную заданного размера size и переменную-указатель на нее.

FREEMEM (p: pointer; size: word) — уничтожает динамическую переменную данного размера.

6.13.3. Примеры использования указателей

Объявление

При объявлении переменной типа «указатель» выделяются лишь ячейки памяти для хранения собственно указателей. В начале работы программы эти ячейки пусты, конкретных адресов в них нет.

Примеры объявлений:

```
Type ip = ^integer;
```

```
A = array[1..5] of byte;
Var x,y:ip;
  B:^byte;
  R:^real;
  Pa:^A;
  p,q:pointer;
Type preco=^struct;
           Struct=record
           C:char; n:integer; end;
Var Pr:preco;
```

Выделение памяти

В результате выполнения процедур New и Getmem в ячейки переменных-указателей записываются адреса, по которым будут располагаться данные соответствующих типов, и выделяется память по указанным адресам для хранения объектов, на которые ссылаются указатели:

```
New(x); New(y); New(B); New(R); New(Pa); New(Pr);
Getmem (p, sizeof(real)); Getmem(q, sizeof(A));
```

Разыменование (занесение значений)

```
x^:=10; y^:=20; B^:=255; R^:=1.125;
for i:=1 to 5 do Pa^[i]:=sqr(i);
Pr^.c:='$'; Pr^.n:= 1000;
```

Копирование значений

```
x^:= y^;
R^:= B^;
```

Копирование адресов

```
x:=y;
```

Пусть в тексте программы имеется: var k:real; k := 625.5;

Тогда присвоение адреса переменной k указателям r и p будет выглядеть следующим образом: r:=@k; p:=@k;

Высвобождение памяти

```
Dispose(y); Dispose(x);
Freemem(p,sizeof(p));
Dispose(Pa); Dispose(Pr);
```

Задание «нулевого» значения

```
new(x);
x:=nil;
```

6.14. Использование указателей для организации связанных динамических структур

Чаще всего указатели используют для ссылки на записи, чем достигается значительная экономия памяти. В такие записи включаются информационные поля для хранения интересующих пользователя данных (любого типа, который может быть полем записи) и обязательно хотя бы одно поле-указатель на следующую запись. Наличие поля-адреса позволяет образовывать *связанные списки* — структуру, в которой отдельные записи связаны последовательно друг с другом посредством поля-указателя.

6.14.1. Списки

▶ *Линейный список* — связанный список, в котором имеется адрес точки входа в список (обозначим его HEAD), а последний элемент указывает на Nil. В любую точку списка разрешается добавлять элемент(ы) и исключать любой элемент. Обе операции производятся с помощью перенастройки указателей и, соответственно, выделения памяти и занесения в него значения (в случае вставки элемента в список) или высвобождения памяти (в случае удаления из списка). Пример линейного односвязного списка представлен на рис. 6.5.

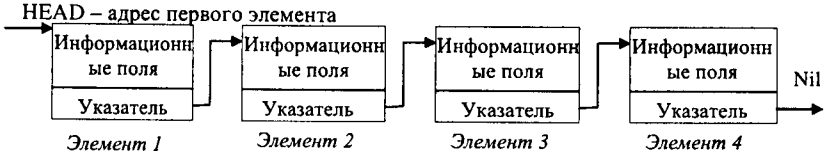


Рис. 6.5

Таким образом, все указатели списка, кроме первого (HEAD), размещаются в динамической памяти. Поскольку каждый элемент списка состоит из разнотипных частей: хранимой информации в виде информационных полей и указателя, то естественно представить в Паскале та-

кой элемент типом Record. Для указателя на вход в список необходимо описать переменную соответствующего типа.

Пример описания списка:

```
Type psp = ^tsp;
  tsp = Record
        ... {Поля любых типов}
        ...
        next : psp
      End;
```

Алгоритмы работы со списками

I. Построение списка сводится к последовательному добавлению элементов в первоначально пустой список. На пустой список указывает пустой список (head = nil). Пусть требуется создать список слов, вводимых с клавиатуры. Последнее слово в списке — «end».

```
Type psp = ^tsp;
  tsp = Record
       wd:string;
{В нашем примере информационное поле имеет
тип string}
       next:psp
       end;
Var    head,q,p:psp;
  sl:string;
Begin
  New(p); head:=p; p^.next:=nil;
       Readln(sl); p^.wd:=sl;
  While sl<>'end' do
    Begin
      Readln(sl); new(q); q^.wd:=sl; p^.next:=q;
      q^.next:=nil;  p:=q
    End;
End.
```

II.Просмотр и печать списка. Пусть требуется написать процедуру Print_Spisok печати элементов списка, созданного в алгоритме I.

```
Procedure Print_spisok;
begin
  p:= head;
```

```
    while p <> nil do
         begin
       writeln((p^.wd);  p := p^.next
          end
  end;
```

III. Поиск элемента в списке. Определить, есть ли в списке (алгоритм I) слово «begin» и если есть, то сколько раз встречается. К описанию алгоритма I добавим: var k:integer;

```
...
Begin
  p:=head;
  While p <> nil do
Begin
     If p^.wd = 'begin' then inc(k);
     p:=p^.next
   End;
  If k=0 then writeln('Такого элемента нет')
Else writeln('Слово',sl,'встречается ',k,' раз')
End;
```

IV. Удаление первого элемента списка. Необходимо 1) передвинуть указатель head на следующий элемент; 2) освободить память; 2) освободить память, занятую удаляемым элементом.

```
Begin
  p:=head; head:=p^.next;
  Dispose(p);
  Print_Spisok
End;
```

V. Удаление последнего элемента списка. К разделу описаний алгоритма I добавим вспомогательную переменную q типа psp.

```
Begin
  p := head;  q := p^.next;
  While q^.next <> nil do
   Begin
     p := q;  q := q^.next
   End;
  Dispose(p);  p^.next :=  nil;
  Print_Spisok
End;
```

VI. Вставка элемента в список. Пусть необходимо вставить в список (алгоритм I) новый элемент sl_new перед первым вхождением элемента sl_old, если sl_old входит в список. Добавим к разделу описаний алгоритма I описание переменных sl_new,sl_old : string; и fl:Boolean.

```
Begin
 Write('Введите искомое слово'); readln(sl_old);
 Write('Введите новое слово'); readln(sl_new);
 Fl:=false; p:=head;
 While (p<> nil) and not fl do
   If p^.wd=sl_old then fl:=true else p:=p^.next;
 If fl then {Вставка элемента}
   Begin
     New(q); q^.wd:=sl_old; q^.next:=p^.next;
     p^.wd:=sl_new; p^.next:=q
   End;
  Print_Spisok;
End;
```

6.14.2. Организация стека в динамической памяти

▶ *Стек* — линейный односвязный список, для которого разрешено добавлять или исключать элементы только с одного конца списка — «вершины» стека.

Принцип работы стека: «последним пришел — первым вышел».

Пример. Написать процедуры помещения элементов (целых чисел) в стек и извлечения их из стека.

```
Type pstack = ^element;
  element = record
       m: integer;
       next: pstack;
       End;
Var    p, st:pstack;
            {st – указатель на вершину стека}
  i:integer;
  k,n:integer;
Procedure put (n:integer);
                 {n– параметр-значение}
```

```
    Begin       {Помещает значение в стек}
    New(p);    {выделение памяти под новый элемент}
    p^.m := n;      {заносим в него информацию}
    p^.next := st; {указательная часть предшествующего
                элемента указывает на вершину стека}
  st:=p;
  End;
  Procedure get( var n: integer);
  {извлекает элемент из стека; n — параметр-перемен-
ная т. к. необходимо изменить значение n в основной
программе}
  Begin
    p:=st; {запишем в p адрес вершины стека}
    n:=p^.m;
         {в n — информационная часть верхнего
          элемента стека}
    st:=p^.next;
         {присваиваем вершине стека адрес
         следующего элемента}
    Dispose(p);
  end;

  Begin        {основная программа }
   st:=nil;    {вершина стека — «пустой» указатель,
               стек пуст}
   Readln(k); {количество чисел}
   For i:=1 to k do {вводим числа и помещаем в стек}
   Begin
   Readln(n); put(n)
   End;
  . . . . . .
   While st <> nil do {вывод элементов стека}
    Begin
    Get(n); writeln(n)
    End;
  End.
```

6.14.3. Очередь

▶ *Очередь* — линейный односвязный список, для которого разрешены только два действия: извлечение элемента из начала очереди «головы» и добавление элемента в конец («хвост») очереди.

Другими словами, в очереди элементы всегда добавляются в конец, а удаляются из начала. Принцип работы очереди: «первым пришел — первым вышел».

Пример. Создать очередь из произвольного числа компонентов. Извлечь и напечатать все элементы очереди.

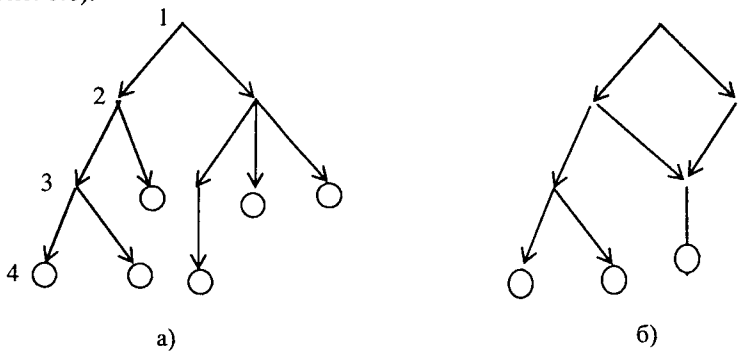
```
Type pocher=^element;
  element=record
  m: integer;
  next: pocher;
End;

Var no,ko,p:pocher;
  {no – начало очереди, ko – конец очереди}
n,k,i:integer;

Procedure putoch(n:integer);
  {помещает значение в очередь}
Begin
  New(p); {выделение памяти под новый элемент}
  p^.m:=n; {заносим в него информацию}
  p^.next:=ko;
  {указательная часть ссылается на начало очереди}
  ko := p
End;

Procedure getoch(var n:integer);
  {извлекает элемент из очереди}
Begin
  n:=p^.m;
  {в n – информационная часть первого элемента очереди}
  p:=no; {запишем в p адрес начала очереди}
  no:=p^.next;
  {присваиваем началу очереди адрес следующего элемента}
  Dispose(p);
End;
```

```
Begin       {основная программа}
  Readln(k);
{создадим первый (он же последний) элемент очереди}
  New(p); readln(n); p^.m:=n; p^.next:=nil;
     no:=p; ko:= p;
  For i:=2 to k do
    Begin
      Readln(n); Putoch(n);
  End;
  While ko <> nil do
    Begin
      Getoch(n);
      Writeln(n)
    End;
End.
```

6.14.4. Деревья

▶ *Деревья* — динамические данные иерархической структуры произвольной конфигурации, состоящие из графов. *Граф* — это структура, состоящая из узлов (или вершин) и соединяющих их дуг. Графы бывают двух видов: ориентированные и неориентированные (если дуги не имеют стрелок).

Специального вида граф, в котором в каждую вершину может входить только 1 стрелка, а выходить несколько, называется *деревом* (рис. 6.6).



▲

Рис. 6.6. а) изображение дерева; б) структура, не являющаяся деревом

Начальная вершина (в которую не входит стрелка) называется *корнем*. Вершины, из которых не выходят стрелки, — *терминальные* (*листья*) (помечены кружками). Структура, выходящая из одного узла — *поддерево*.

▶ Дерево, у которого из каждой вершины выходит не более 2-х стрелок, называется *бинарным* или *двоичным* деревом.

Уровень — количество стрелок, которые нужно пройти, чтобы добраться от нулевого уровня к данной вершине (см. рис. 6.6).

В дереве может быть только один путь от корня к узлу; дерево не содержит петель и циклов, что следует из определения.

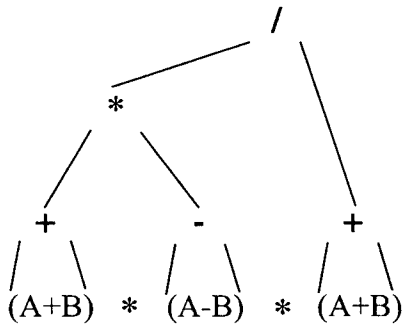
В виде деревьев можно изобразить отношения между субъектами из различных предметных областей.

Примеры деревьев:

Факультет: 5 курсов, на каждом курсе — определенное число групп; в каждой группе — *n* человек.

Программа на ПК. Ее блоки составляют дерево, дуги которого означают вложенность структур.

Арифметическое выражение можно представить в виде деревьев, в узлах которых находятся операции, за исключением терминальных вершин. В них находятся переменные или числа (рис. 6.7).



Терминальные вершины — цифры или переменные, остальные узлы – знаки операции

▲

Рис. 6.7

Для хранения и поиска информации наиболее часто применяют двоичные деревья.

Организация деревьев в динамической памяти

Определить (построить) дерево — значит определить узлы и связи между ними. Применительно к динамической памяти — это элементы комбинированного типа, в которых динамическая переменная может иметь одно и более информационных полей для хранения данных и столько указательных полей, сколько стрелок (максимально) может выходить из вершины.

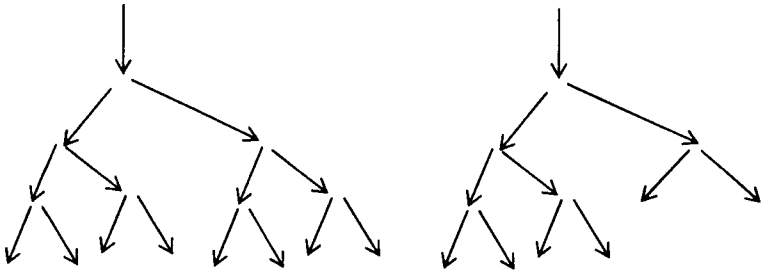
Для двоичного дерева достаточно двух указательных полей — на левую и правую «ветви»:

```
Type ptree=^ttree;
     ttree = record
       ... {инфомационные поля}
     l,r:ptree {указатели на левую и правую ветвь}
     End;
```

Построение полного двоичного дерева

▶ Дерево называется *полным*, если все вершины, уровень которых меньше некоторого заданного n, не являются терминальными, а все вершины, кроме терминальных, имеют одинаковое количество выходящих стрелок (рис. 6.8).

Построение полного двоичного дерева состоит из нескольких этапов. Проходя к какой-либо вершине, мы должны запоминать указатель пути. Для этой цели удобно использовать стек. Остановка алгоритма — создание самого правого пути.



а) полное бинарное дерево из 4-х уровней

б) неполное бинарное дерево из 4-х уровней

▲

Рис. 6.8

Построение полного двоичного дерева состоит из нескольких этапов. Проходя к какой-либо вершине, мы должны запоминать указатель пути. Для этой цели удобно использовать стек. Остановка алгоритма — создание самого правого пути.

Пример. Написать программу построения полного двоичного дерева в динамической памяти из 4-х уровней (рис. 6.9).

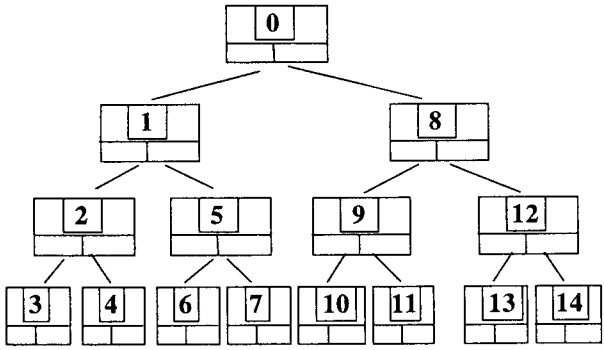


Рис. 6.9

```
{Построение полного двоичного дерева}
Const n=4; {сколько уровней}
Type ptree=^ttree;
    Ttree=record {опишем запись для создания дерева}
        dat:integer;
        l,r:ptree
        End;
Var
  kor,v,t:ptree;
  i,j:Integer;
  {kor –корень, v,t – вспомогательные, i – значение
информационного поля очередной вершины, j – № уровня}
Type pstack=^element;
  {Опишем запись для стека. Информационные поля имеют
тип ptree – дерева}
       element = record
       m:ptree;
       next:pstack
       End;
```

```
Var
  p,h:pstack;
Procedure putstack (n:ptree);
Begin
  New(p); p^.m:=n; p^.next:=h; h:=p
End;
Procedure getstack(var n:ptree);
Begin
  p:=h;  n:=p^.m;  h:=p^. next;  dispose (p)
End;
Begin
  h:=nil;  {стек в начале пуст}
  j:=0; i:=0; new(kor); {0 - уровень}
  t:=kor; t^.dat:=i; t^.r:=nil; t^.l:=nil; inc(i);
  Repeat
    While j<n do {спуск}
      Begin
         New(v); v^.dat:=i; v^.l:=nil; v^.l:=nil;
         v^.r:=nil;
      Inc(i);
      {создаем новый узел - вершину}
      If t^.l=nil then t^.l:=v else t^.r:=v;
{связывает его с левой  или правой ветвью узла
  предыдущего уровня}
        Putstack(t); {запомним: пока в стеке!}
        t:=v;  {write (t^.dat);} inc(j);
{переместим указатель t и переход на следующий
 уровень}
      End;
      Repeat  {подъем}
        Getstack(t);  dec(j) {взять из стека}
      Until (j=0) or (t^.r=nil);
    {либо дошли до корня, либо не заполнена
     правая ветвь}
    Until (t=kor) and (t^.r<>nil);
    {дошли до корня, и правая ветвь не пуста}
End.
```

Алгоритмы работы
с двоичными упорядоченными деревьями (деревьями поиска)

▶ Двоичное дерево упорядочено (является *деревом поиска*), если в нём все ключи левого поддерева каждого узла меньше, чем ключ узла, а ключи правого поддерева — больше.

▶ *Ключ* — признак, по которому ведётся поиск — значение одного из информационных полей дерева.

Пример. Необходимо создать базу данных «Рейтинг» для хранения сведений о студентах и их успеваемости. При этом программа должна обеспечивать: 1) добавление новых записей, 2) вывод отсортированной по какому-либо признаку информации (например, по алфавиту, по среднему баллу успеваемости), 3) быстрый поиск нужной записи (например, по фамилиям студентов, по номеру группы), 4) удаление записи.

```
Type preit = ^ treit;
Treit = record
  fam: string [20]; {фамилия};
  Ngr: integer; {№ группы};
  srb: real {ср. балл}
       ...
  l, r: preit;
End;
```

Для выполнения указанных задач удобно хранить информацию в виде упорядоченного двоичного дерева. При этом выбор ключа зависит от поставленной задачи. Если необходимо хранить и выводить информацию по алфавиту, а также вставлять элементы, не нарушая алфавитного порядка, то ключом будет поле fam типа string. Если информация дерева была отсортирована по среднему баллу — соответственно ключом будет поле srb типа real и т. д.

Почему в виде двоичного дерева удобнее хранить подобную информацию, чем в виде, например, связанных списков? Рассмотрим пример. Пусть необходимо построить связанный список, в котором фамилии распределены по алфавиту, при этом на вход данные поступают случайным образом. В этом случае иногда запись будет вставлена в начало списка, иногда — в конец, иногда — в середину. В среднем, до определения положения вставки новой записи придётся просматривать половину списка. Процесс поиска можно было бы ускорить, если бы вместо такого последовательного просмотра можно было сравнить новую за-

пись с записью из середины списка. В результате такого сравнения можно решить: помещать новую запись после средней или до неё. И далее половину списка игнорировать. Этот процесс повторять до тех пор, пока не найдём искомую запись (при поиске) или подходящее место (при вставке).

Например, если в списке 64 записи, то такой метод потребует максимум 7 сравнений, а обычный поиск, в среднем, 32 сравнения. В идеальном случае описанную структуру можно представить в виде двоичного дерева.

Таким образом, в упорядоченных массивах можно быстро найти нужную информацию. Динамические списки можно быстро пополнять новыми элементами, используя память из кучи. Двоичные упорядоченные деревья соединяют оба этих качества и позволяют быстро осуществлять двоичный поиск информации.

Пусть записи поступали в следующем порядке: L, E, R, A, W, H, P, а необходимо запомнить их в алфавитном порядке. Для простоты положим, что каждая запись содержит поле-букву, которое и является ключом. Тогда: отсортированный список и упорядоченное двоичное дерево будут выглядеть соответственно рис. 6.10 и рис. 6.11.

Рис. 6.10

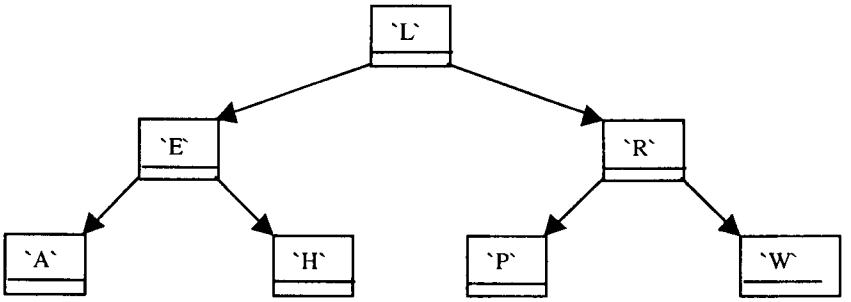


Рис. 6.11

Алгоритм построения двоичного дерева, а также включение новых элементов (процедура bild) будет выглядеть следующим образом:

1. Первая поступившая запись выбирается «корнем» дерева.

2. При поступлении каждая следующая запись сравнивается с корневой.

2.1. Если поступившая запись предшествует корневой, идти в левое поддерево.

2.2. Иначе — в правое поддерево.

3. Если того поддерева, в которое можно вставить новую запись не существует (на что указывает значение nil левой или правой связи), то новую запись надо вставить в этом месте (тем самым формируя новое поддерево, состоящее из единственной записи).

При этом для вставки новой записи нет необходимости изменять связи, указывающие на другие записи как для связывания списка.

Алгоритм просмотра входящих в упорядоченное дерево записей и изображения записей, отсортированных по выбранному ключу (prozm (kor)):

1. Изобразить записи левого поддерева и т. д., пока не будет встречено пустое поддерево — тогда никаких действий не предпринимать.

2. Изобразить корневую запись.

3. Напечатать записи правого поддерева, пока не будет встречено пустое поддерево.

Алгоритм двоичного поиска в упорядоченном дереве:

1. Если дерево не пусто, сравнить искомый ключ с тем, что в корне дерева.

2. Если ключи совпадают, поиск завершён.

3. Если ключ в корне больше искомого, выполнить поиск в левом поддереве.

4. Если ключ в корне меньше искомого, выполнить поиск в правом поддереве.

5. Если дерево пусто (пройдены все элементы), поиск неудачен.

Рекурсивные алгоритмы работы с двоичными деревьями

Дадим рекурсивное определение дерева. Представим, что отдельные дуги дерева ведут к частям дерева, которые сами являются деревьями. Такая точка зрения приводит к рекурсивному определению дерева. *Дерево* — это пустая структура или узел, связанный дугами с конечным числом деревьев. Прохождение дерева заключается в обходе всех его узлов. **Рекурсивный алгоритм прохождения двоичного** дерева будет таким:

1. Посетить корень.

2. Посетить левое поддерево.

3. Посетить правое поддерево.

Обходя дерево по этому алгоритму, мы посетим узлы в следующем порядке (по алфавиту) (рис. 6.12):

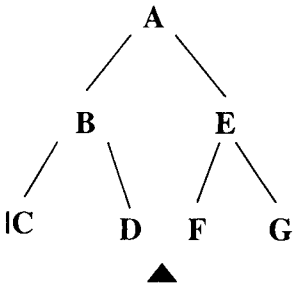


Рис. 6.12

Проходя дерево по данному алгоритму, мы посетим вершины дерева в следующем порядке: A, B, C, D, E, F, G. Как изменить порядок пунктов алгоритма, чтобы пройти узлы в последовательности:

а) A, E, G, F, B, D, C б) C, D, B, F, G, E, A в) C, B, D, A, F, E, G?

Примеры использования рекурсивных алгоритмов для работы с двоичными деревьями.

```
Const k = 13;
Const a:array[1..k] of integer=
                (2,4,6,1,3,5,7,10,12,14,11,13,15);
Type ptree=^ttree;
     ttree = record
        dat:integer;
         l,r:ptree
       End;
Var kor,t:ptree;
    i,x:integer;
    fl:boolean; {v – новый элемент}

Procedure bild (v: ptree; var kor:ptree);
                                        {построение}
  Begin
    If kor = nil
     Then begin kor := v; v^.l:=nil; v^.r:=nil end
      Else if v^.dat < kor^.dat
        Then  bild(v,kor^.l)
```

```
                    {заполнение левой ветви}
        Else bild(v,kor^.r)
                    {заполнение правой ветви}
  End;

Procedure prosm(t:ptree);
  {просмотр в порядке возрастания}
  Begin
    If t^.l <> nil then prosm(t^.l);
    Writeln(t^.dat);
    If t^.r<> nil then prosm(t^.r);
  End;

Procedure poisk(t:ptree);
  {линейный поиск  в неупорядоченном дереве}
  Begin
    Writeln(' ',t^.dat);
    If t^.dat=x
     Then writeln ('Нужный элемент найден !',t^.dat)
      Else
        Begin
          If t^.l <> nil then poisk (t^.l);
          If t^.r <> nil then  poisk (t^.r);
        End;
  End;

Procedure dv_poisk(t:ptree);
  {двоичный поиск в упорядоченном дереве}
  Begin
    Write(' ',t^.dat);
    If t^.dat = x
      Then
        Begin
          fl := true; writeln;
         Writeln ('Нужный элемент найден!',t^.dat)
        End
      Else
   If (x<t^.dat) and (t^.l<>nil) then dv_poisk(t^.l)
    Else if t^.r<>nil then dv_poisk(t^.r)
  End;
```

```
{******  ОСНОВНАЯ ПРОГРАММА  ******}
Begin
  kor:=nil;
  For i:=1 to k do {строим упорядоченное дерево}
    Begin
      New(t); t^.dat:=a[i]; bild(t,kor);
    End;
  Writeln('Просмотр дерева!'); prosm(kor);
{или writeln('Обход дерева!'); obch(kor);}
  Readln;
  Write('Линейный поиск, введите x ');
  Readln(x); poisk(kor);
  Writeln('Двоичный поиск: ');
  fl:=false; dv_poisk(kor);
  If not fl then writeln('Элемент не найден!');
  Readln
End.
```

6.15. Введение в объектно-ориентированное программирование

Начиная с первого мнемокода, все средства разработки программ изобретались с единственной целью: чтобы проще было думать. Объектно-ориентированное программирование (ООП) является одним из способов достижения этой цели. Почему проще думать в ООП?

Программа, решающая некоторую задачу, заключает в себе описание части мира, относящегося к этой задаче. Описание действительности в форме системы взаимодействующих объектов естественнее, чем в форме иерархии подпрограмм. Предпосылки возникновения ООП: модульное программирование, абстрактные типы данных, ситуационное моделирование, фреймы. Первый алгоязык, где были классы и объекты, — Simula-67. Окончательно принципы ООП оформились в языке Smalltalk-80. Среди современных систем программирования, в основу которых положены принципы ООП, можно назвать *Delphi*, *Visual* C, *Visual Basic*.

6.15.1. Тип Объекты

▶ *Объект* представляет собой *единство состояния и методов*. Программируя объект, его состояние можно хранить в наборе переменных, а методы реализовать в форме процедур и функций.

В Паскале объект — это особый тип данных, а экземпляры объекта — переменные этого типа.

Состояние характеризуется значениями полей объекта.

Методами объекта являются ассоциированные с ним функции и процедуры, которым доступны поля.

Передача сообщений объекту происходит в виде вызовов его методов с заданными параметрами.

Тип объекта можно рассмотреть как усовершенствование типа запись; только поля (описывающие свойства) дополняются методами (описаниями действий с объектом).

В описании объектного типа дают только заголовки соответствующих блоков, а сами блоки приводят ниже. За счёт методов описания сущности как бы «оживают». В ООП оперируют цельными представлениями сущностей.

Описание типа «объект»:

Object
{*описание свойств*}
<идентификатор1, ..., идентификаторN>:<тип>;
...
{*описание методов*}
Procedure <идентификатор>[<параметры>];
...
end;

6.15.2. Основные понятия ООП. Инкапсуляция. аследование

▶ *Инкапсуляция* — объединение информационных полей и методов, которые выполняют над ними действия.

Пример. Простейший объект — позиция на экране. Пусть в программе необходимо случайным образом выводить точки и получать их координаты.

```
Uses graph;
Type pozicia = object {начало описания типа pozicia}
     x,y:integer; {координаты позиции}
Procedure Init (XN, YN: integer);
{заголовки блока порождения объекта}
Procedure Locate (Var XL, YL: integer);
{заголовок блока получения координат позиции}
End;
```

```
Procedure Pozicia.Init; {можно не писать параметры}
  Begin x:=XN; y:=YN end;
  {поля x, y объекта глобальны для метода}
Procedure Pozicia.Locate;
  Begin xL:=X; yL:=Y end;
Var i, d, m, t: integer;
       p: pozicia; {описание переменной-объекта}
       xx, yy: array [1..1000] of integer;
Begin {основная программа}
  d: =detect; Randomize;
{инициируем графический режим и генератор
 случайных чисел}
  InitGraph (d, m, "c:\bp\bgi");
  Clear device; {очистка экрана}
  For i :=1 to 1000 do
    Begin
      P.Init (Random (GetMaxX), Random (GetMaxY));
      {порождение объекта}
      P.Locate(xx[i],yy[i]);
      {запоминаем её координаты в массиве}
    End;
  Cleardevice;
End.
```

Блоки-методы размещаются в произвольном порядке. Совпадение их имён, если методы принадлежат разным типам, не запрещено, следовательно, в заголовках методов их имена снабжаются префиксом — именем типа. Например: pozicia.init, krug.init.

Обращение к методу нужно начинать с *префикса* — имени действующего объекта. Например: P.Locate (параметры).

В обоих случаях разделителем служит (.).

Допустимо обращение с полями объекта как с полями записи: P.z, P.y, но *нежелательно*.

Объекты не запрещается использовать в блоках, не являющимися методами. Но такое применение должно быть исключением, так как противоречит идее объектов.

Доступ к полям объектов можно принудительно ограничивать. Чтобы поля были доступны лишь методам данного модуля, в объектном типе до описания группы полей записать «**Private**», а после неё — «**Public**». Это делают, чтобы предупредить ошибочные действия с объектными данными пользователя библиотеки объектов.

▶ *Наследование* — позволяет создавать новые объекты, изменяя или дополняя свойства прежних.

Такие простые программы, как рассмотренный выше пример, мало значимы, но они позволяют избежать дублирования элементов описания, их называют родительскими.

Родительские (предки) — тип объектов, которые используются как носители общих свойств и методов для семейства типов — потомков, позволяя избежать дублирования элементов описания.

Объект может не использоваться самостоятельно, а служить основой для начинающегося с него дерева объектов. Такие объекты называются *абстрактными*. Они объединяют в себе некоторые общие для всех своих потомков поля и методы.

Абстрактные методы — методы, которые не содержат исполняемых операторов и служат заготовками для аналогичных методов в объектах — потомках. Абстрактные методы должны перекрываться в потомках.

Правила наследования.

1. *Объект-наследник (потомок)* получает все поля и методы предка.

2. Потомок может (хотя и не обязательно) добавить: собственные поля, методы или перекрыть своими методами одноимённые унаследованные методы.

3. Любой тип-потомок может быть родительским для других типов. В этом случае типы наследуют поля ***всех*** своих предков .

4. Доступ к полям и методам родительских типов в описании любых типов-потомков выполняется так, как будто они описаны в самом типе-потомке.

5. Ни в одном из типов-потомков не должны использоваться имена полей и формальных параметров, совпадающих с указанными в типах-предках. Идентификаторы методов могут совпадать.

6. Имена методов в родительских типах-потомках могут совпадать. В этом случае метод дочернего типа «перекрывает» метод родительского. При этом обращение к родительскому методу остаётся возможным с помощью префикса — имени родителя (Position.Init, если у Position есть потомок, у которого также имеется метод Init).

Замечание. Значения полей не наследуются. Наследование — это *отношение объектных типов*, а не объектов.

Совместимость объектных типов по присваиванию

Пусть объекты A и B однотипны, либо B — потомок A. Присваивание A:=B означает передачу (копирование) всех значений объекта B в

поля объекта A. Если A и B не однотипны, либо A — потомок B, присваивание A:=B недопустимо. Это объясняется тем, что в первом случае для каждого поля в A есть соответствующее поле в B, а во втором случае это требование не выполняется.

6.15.3. Полиморфизм

▶ *Полиморфизм* или «многообразие» — выражается в том, что под одним и тем же именем скрываются различные действия, содержание которых зависит от типа объекта.

Полиморфизм предполагает определение *класса* или нескольких *классов методов* для родственных объектных типов так, что каждому классу отводится своя функциональная роль.

Например, метод «отобразить геометрическую фигуру на экране», на самом деле не метод, а класс, т. к. характер фигуры определяет конкретный способ отображения (конкретный метод).

Методы одного класса обычно наделяются общим именем. Например, **Show, Init**.

Проведем аналогию с процедурными типами. Сложному стандартному блоку (например, вычисляющему интеграл) передают тот или иной подчиненный блок (подынтегральную функцию). Без процедурного типа пришлось бы запасать копии стандартного блока, различающиеся в местах вызова подчиненного блока.

При работе с объектами может возникнуть сходная ситуация: сложный метод приходится давать заново для каждого типа объекта, хотя различия в поведении объектов могут быть небольшими.

Изменяя алгоритм того или иного метода в потомках объекта, программист может придавать этим потомкам отсутствующие у родителя специфические свойства.

Для изменения метода необходимо перекрыть его в потомке, то есть объявить в потомке одноименный метод и реализовать в нем нужные действия. В результате в объекте-родителе и объекте-потомке будут действовать два одноименных метода, имеющих разную алгоритмическую основу, а следовательно, придающие объектам разные свойства. Это свойство и называется *полиморфизмом* объектов.

В Паскале полиморфизм достигается также *виртуализацией* методов, позволяющей родительским методам обращаться к методам потомков.

6.16. Основы программирования в среде Visual Basic for Application (VBA)

В предыдущих разделах данной главы были достаточно подробно описаны основные понятия и базовые структуры, используемые при построении алгоритмов и написании программ для компьютеров, которые являются общими для всех языков программирования. Освоив искусство программирования на одном каком-либо алгоритмическом языке, современному пользователю компьютера не сложно перейти к изучению других языков, отличающихся лишь нюансами синтаксиса основных конструкций, а также освоить какую-либо среду быстрой разработки приложений (или визуальную среду). Современные системы программирования, такие как Delphi, Visual Basic, Visual Basic for Application, Visual C++, Visual Studio.Net и им подобные, располагают интерактивной справочной системой, в которой содержится исчерпывающая информация обо всех объектах среды, их свойствах и методах, а также о самой интегрированной среде разработки программ. В этом разделе остановимся лишь на некоторых основных моментах, позволяющих познакомить читателя с основами программирования в одной из таких систем, ставшей популярной среди пользователей компьютеров ввиду широкого распространения пакета Microsoft Office, а также простоты ее изучения. Примеры выполнения программ на VBA, а также практические задания для самостоятельного изучения приведены, например, в [23]. Запуск VBA осуществляется из любого приложения Microsoft Office нажатием клавиш Alt-F11.

6.16.1. Типы данных VBA

Перечислим некоторые из основных типов данных в VBA.

Целые

Byte (1 байт), *Integer* (2 байта), *Long* (4 байта) — аналогичны целочисленным типам Паскаля.

Вещественные

Single — число с плавающей запятой (4 байта), *Double* — число с плавающей запятой двойной точности (8 байт).

Логические

Boolean — аналогичен логическому типу Паскаля.

Строковый

String — строка, может быть постоянной или переменной длины.

Объект

Object — любой указатель объекта (4 байта).

Подтипы

Variant — числовые (любое числовое значение вплоть до границ диапазона для типа *Double*, 16 байт); строковые (от 0 до приблизительно 2 млрд, 22 байта плюс длина строки).

Дата и время

Date (дата от 1 января 100 г. до 31 декабря 9999 г.)

Тип данных, определяемый пользователем

с помощью ключевого слова *Type*, (объём определяется элементами).

6.16.2. Описание переменных

Описание типа каждой переменной делает программу надежнее и, кроме того, убыстряет ее работу, т. к. VBA не требуется тратить время на распознавание типа неописанной переменной при каждом обращении к ней.

Формат описания:

Dim [WithEvents] <ИмяПеременной> [As [New] <Тип>],

где аргументы

WithEvents — это ключевое слово, указывающее, что аргумент *ИмяПеременной* является именем объектной переменной, которая используется при отклике на события, генерируемые объектом ActiveX (т. е. объектом, который может быть открыт для других приложений и средств программирования);

ИмяПеременной — имя переменной, удовлетворяющее стандартным правилам именования переменных;

New — ключёвое слово, включающее возможность неявного создания объекта;

Тип — тип данных переменной.

Инструкция Dim предназначена для описания типа данных переменной на уровне модуля или процедуры. Переменные, описанные с помощью ключевого слова Dim на уровне модуля, доступны для всех проце-

дур в данном модуле. Переменные, описанные на уровне процедуры, доступны только в данной процедуре. Например, следующая инструкция описывает переменную с типом Integer.

```
Dim X As Integer
```

Инструкция Dim предназначена также для описания объектного типа переменных. Опишем переменную для нового экземпляра рабочего листа MS Excel:

```
Dim Y As  New Worksheet
```

Если при описании объектной переменной не используется ключевое слово *New* то для использования объекта, на который ссылается переменная, существующий объект должен быть присвоен переменной с помощью инструкции Set.

Если тип данных или тип объекта не задан, и в модуле отсутствует его явное описание, по умолчанию переменная получает тип *Variant*. Для обязательного описания всех переменных надо поместить в начале модуля инструкцию Option Explicit. Использование этой инструкции полезно при отладке программ, т. к. она позволяет легче отслеживать возможную путаницу в именах при наборе программы.

Допустимые имена

Правила задания имен в VBA аналогичны правилам в Паскале. Как и в любом языке программирования, вводимые пользователем имена должны отражать суть обозначаемого объекта так, чтобы делать программу легко читаемой. В русифицированных приложениях MS Office допускается использование русских символов в именах.

6.16.3. Константы

Формат описания:

[Public|Private] CONST <ИмяКонстанты> [As <Тип>]= <Выражение>

Здесь:

Public — ключевое слово, используемое на уровне модуля для описания констант, доступных всем процедурам во всех модулях. Не допускается в процедурах;

Private — ключевое слово, используемое на уровне модуля для описания констант, доступных только внутри модуля, в котором выполняется описание. Не допускается в процедурах;

ИмяКонстанты — имя константы, удовлетворяющее стандартным правилам именования переменных;

Тип — один из поддерживаемых типов данных;

Выражение — значение соответствующего типа, другая константа или любое выражение, которое может включать арифметические и логические операторы.

Пример:

```
Const Z As Single = 0.2
Const Name = «Имя»
```

Комментарии

Работая с программой, удобно использовать комментарии, т. е. фрагменты текста программы, не являющиеся программными кодами и игнорируемые VBA. Комментарии выполняют две важные функции:

1. Делают программу легко читаемой, поясняя смысл программных кодов и алгоритма.

2. Временно отключают фрагменты программы при ее отладке.

В языке VBA существуют два способа ввода комментариев:

Применение апострофа (‘). Его можно ставить в любом месте строки. При этом все символы, начиная от апострофа до конца строки, будут восприниматься средой разработки как комментарий.

Применение зарезервированного слова Rem вместо апострофа.

Ниже приведен пример использования комментариев в тексте программы:

```
Dim x1 As Integer
'x1 – целая переменная
Dim st As String' st – строковая переменная
```

6.16.4. Операции, операторы и встроенные функции VBA

В программах на VBA можно использовать стандартный набор операций над данными. Как и в других языках, в нем имеются три основных типа операций: математические, операции отношения и логические, запись которых аналогичная описанным п 6.4. Приведем лишь отличные от Паскаля:

\ — целочисленное деление;

^ — возведение в степень.

В VBA имеется большой набор встроенных функций и процедур, перечислять которые в рамках учебника не представляется возможным,

для этого существует специальная справочная литература, а также, как уже упоминалось выше, встроенная справочная система VBA. Основные категории функций: математические функции, функции проверки типов, функции преобразования форматов, функции обработки строк, функции времени и даты.

Оператор VBA представляет собой полную команду языка VBA. Он может содержать ключевые слова, операции, переменные, константы и выражения. Как и в Паскале, основными являются операторы присваивания, ввода и вывода данных, перехода и выбора, циклов.

Оператор присваивания

Оператор присваивания позволяет присвоить значение выражения переменной, константе или свойству объекта. Оператор присваивания всегда включает знак равенства (=). Формат оператора:

<Переменная>(<Константа>,<Свойство Объекта>) = <Выражение>

Оператор присваивания предписывает выполнить выражение, заданное в его правой части, и присвоить левой части. В результате, например, действия следующей пары операторов

```
x = 2
x = x + 2
```

переменной *x* будет присвоено 4.

Для присваивания переменной ссылки на объект применяется инструкция Set. В следующем примере инструкция Set присваивает переменной Область Диапазон A1:B3 (ячеек листа Excel):

```
Set Область = Range(«A1:B3»)
```

Расположение нескольких операторов на одной строке

Использование знака двоеточия позволяет разместить несколько операторов на одной строке. Таким образом, следующие две конструкции эквивалентны:

```
x = x+1
y = x+2
```

и

```
x = x+1: y = x+2
```

6.16.5. Ввод и вывод информации

Как правило, в визуальных средах осуществляется с помощью диалоговых окон. В проектах VBA часто встречаются две разновидности диалоговых окон: окна сообщений и окна ввода. Они встроены в VBA, и если их возможностей достаточно, то можно обойтись без проектирования диалоговых окон. Окно сообщений (*MsgBox*) выводит простейшие сообщения для пользователя, а окно ввода (*InputBox*) обеспечивает ввод информации.

InputBox

Выводит на экран диалоговое окно, содержащее сообщение и поле ввода, устанавливает режим ожидания ввода текста пользователем или нажатия кнопки, а затем возвращает значение типа string, содержащее текст, введенный в поле. Синтаксис:

InputBox (prompt [, title] [, default] [, xpos] [, ypos] [, helpfile, context])

Здесь аргументы означают:

prompt — строковое выражение, отображаемое как сообщение в диалоговом окне;

title — строковое выражение, отображаемое в строке заголовка диалогового окна. Если этот аргумент опущен, в строку заголовка помещается имя приложения;

default — строковое выражение, отображаемое в поле ввода как используемое по умолчанию, если пользователь не введет другую строку. Если этот аргумент опущен, поле ввода изображается пустым;

xpos — числовое выражение, задающее расстояние по горизонтали между левой границей диалогового окна и левым краем экрана. Если этот аргумент опущен, диалоговое окно выравнивается по центру экрана по горизонтали;

ypos — числовое выражение, задающее расстояние по вертикали между верхней границей диалогового окна и верхним краем экрана. Если этот аргумент опущен, диалоговое окно помещается по вертикали примерно на одну треть высоты экрана;

helpfile — строковое выражение, определяющее имя файла справки, содержащего справочные сведения о данном диалоговом окне. Если этот аргумент указан, необходимо наличие также аргумента context;

context — числовое выражение, определяющее номер соответствующего раздела справочной системы. Если этот аргумент указан, необходимо наличие также аргумента helpfile.

MsgBox

Выводит на экран диалоговое окно, содержащее сообщение, устанавливает режим ожидания нажатия кнопки пользователем, а затем возвращает значение типа integer, указывающее, какая кнопка была нажата. Синтаксис:

MsgBox(prompt[, buttons] [, title] [, helpfile, context])

Аргументы:

prompt — строковое выражение, отображаемое как сообщение в диалоговом окне;

buttons — числовое выражение, представляющее сумму значений, которые указывают число и тип отображаемых кнопок, тип используемого значка, основную кнопку и вид окна сообщения. Значение по умолчанию этого аргумента равняется 0.

Рассмотрим пример использования окон сообщений. В результате действия приведенной ниже процедуры CommandButton1_Click() появится диалоговое окно *Пример окна ввода* с полем ввода (рис. 6.13). Следуя приглашению в этом диалоговом окне, введем в поле ввода имя, например, *Николай Маркович*. Нажмем кнопку ОК. На экране отобразится диалоговое окно пример окна сообщения с текстом приветствия (рис. 6.14). Если пользователь не введет имя в поле ввода диалогового окна *Пример окна ввода* или нажмет кнопку Отмена, то компьютер выразит свое неудовлетворение действиями пользователя отображением диалогового окна *Еще один пример* окна сообщения (рис. 6.15).

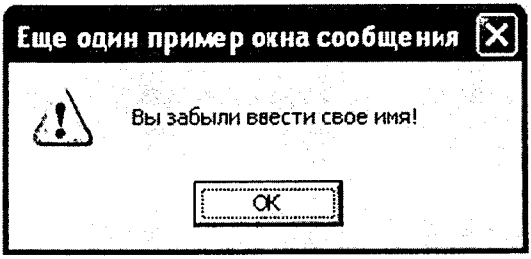


Рис. 6.13

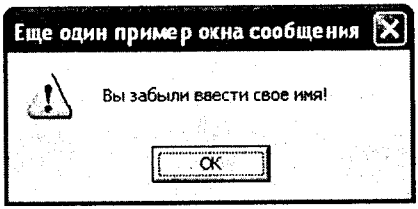


Рис. 6.14

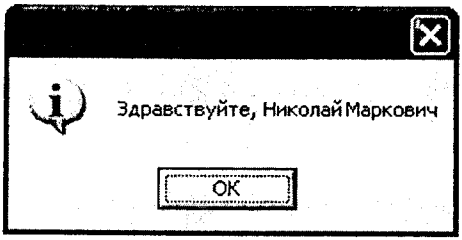


Рис. 6.15

```
Private Sub CommandButton1_Click()
' Описание переменной
Dim Name As String
' Ввод имени пользователя
Name = InputBox(«Введите свое имя», «Пример окна ввода»)
' Реакция программы на ввод имени пользователя
If Name <> «» Then
Response = MsgBox('Здравствуйте, ' + Name, vbInformation, ' ')
Else
Response = MsgBox('Вы забыли ввести свое имя!', vbExclamation, 'Еще один пример окна сообщения')
End If
End Sub
```

6.16.6. Реализация разветвляющихся алгоритмов в VBA

Операторы перехода и выбора

Позволяют организовать ветвление в программе посредством выполнения тех или иных действий в зависимости от некоторого условия. Оператор GoTo осуществляет безусловный переход, но в настоящее время используется в программах все реже.

Перечислим операторы перехода и выбора VBA.

Оператор безусловного перехода. Формат:

GoTo Строка

Задает безусловный переход на указанную строку внутри процедуры. Обязательный аргумент Строка может быть любой меткой строки или номером строки

Оператор условного перехода (аналогичен соответствующему оператору Паскаля). Формат:

If <Условие> Then <Операторы 1> [Else Операторы 2]

Если *Условие* принимает значение *True* (истина), то выполняются операторы после Then, иначе выполняются операторы после Else. Ветвь Else может отсутствовать.

Допускается также использование формы синтаксиса в виде блока:

```
If <Условие> Then
[Операторы 1]
...
[Elself <Условие-n> Then
[Операторы n]
...
[Else
[Операторы]] End If
```

Оператор выбора (аналогичен оператору множественного выбора в Паскале). Формат:

```
Select Case <выражение>
[Case списокВыражений-1
[операторы-1]]
...
[Case списокВыражений-n
[операторы-n]]
[Case Else
[операторы_else]]
End Select
```

Операторы-n (необязательная часть) — один или несколько операторов, выполняемых в том случае, если выражение совпадает с любым компонентом списка список-Выражений-n.

Операторы_else (необязательная часть) — один или несколько операторов, выполняемых в том случае, если выражение не совпадает ни с одним из предложений Case.

Активизация подпрограммы обработки ошибок (в Паскале такую подпрограмму программист вынужден разрабатывать «вручную»).

Оператор

On Error GoTo строка

активизирует подпрограмму обработки ошибок, начало которой определяется обязательным аргументом *строка*, значением которого может быть любая метка строки или номер строки. Для того чтобы предотвратить выполнение программы обработки ошибок в тех случаях, когда ошибка не возникла, необходимо помещать соответствующую инструкцию Exit Sub, Exit Function или Exit Property сразу после подпрограммы обработки ошибки, как в следующем примере:

```
Sub InitializeMatrix(Varl, Var2, Var3, Var4)
On Error GoTo M1
.....
Exit Sub
M1:
.....
Resume Next
End  Sub
```

В этом примере программа обработки ошибок помещена между операторами Exit Sub и End Sub, что позволяет отделить ее от части программы, соответствующей нормальному ходу выполнения.

On Error Resume Next указывает, что при возникновении ошибки происходит передача управления на инструкцию, непосредственно следующую за инструкцией, вызвавшей ошибку.

On Error GoTo 0 отключает любой активизированный обработчик ошибок в текущей процедуре

6.16.7. Операторы цикла

Позволяют организовать в программе повторение некоторых действий (как правило, с различными значениями параметров цикла).

В VBA имеются следующие операторы цикла

For — Next. Формат:

For <Счетчик> = <Начало> To <Конец> [Step <Шаг>] [Операторы 1]
[Exit For]
[Операторы 2] Next [<Счетчик>]

Цикл со счетчиком повторяет выполнение группы Операторов 1, пока Счетчик изменяется от Начального значения до Конечного с указанным Шагом. Если шаг не указан, то он полагается равным 1. Досрочный способ выхода из цикла предоставляет оператор Exit For.

For Each — Next. Формат:

For Each <Элемент> In <Группа>
[Операторы 1]
[Exit For]
[Операторы 2]
Next [<Элемент>]

Цикл повторяет выполнение группы Операторов 1 для каждого элемента массива или семейства.

Do Until — Loop. Формат:

Do [Until <Условие>]
[Операторы 1]
[Exit Do]
[Операторы 2] Loop

Тело цикла выполняется, пока Условие имеет значение *False*. Операторы 1 выполняются по крайней мере один раз, а затем проверяется условие. Досрочный способ выхода из цикла —Exit Do.

Do — Loop While. Формат:

Do
[Операторы 1]
[Exit Do]
[Операторы] Loop [While <Условие>]

Повторяет выполнение набора операторов, пока Условие имеет значение *True*. Сначала выполняются Операторы 1, а потом проверяется условие.

Do While — Loop. Формат:

Do [While <Условие>]
[Операторы 1]
[Exit Do]
[Операторы 2]
Loop

Цикл с «предусловием» проверяет Условие перед выполнением Операторов 1. Когда Условие становится ложным, цикл прекращает свое выполнение.

Do — Loop Until. Формат:

Do
[Операторы 1]
[Exit Do]
[Операторы 2]
Loop [Until <Условие>]

Повторяет выполнение набора инструкций, пока условие не примет значение *True*. Сначала выполняется инструкция, а потом проверяется условие.

While — Wend. Формат:

While <Условие>
[Операторы]
Wend

Выполняет последовательность инструкций, пока заданное условие имеет значение *True*.

6.16.8. Массивы

Как и в других языках программирования, в VBA можно использовать массивы. Примеры объявления массивов:

```
Dim M(3,3) As Single
Dim V(12) As Integer
```

Первая строка объявляет двумерный массив 3г3 (матрицу), состоящий из действительных чисел. Вторая строка объявляет одномерный массив (вектор) из 12 целых чисел, причем по умолчанию первый элемент массива будет V(0), а последний V(11). В этом случае говорят, что 0 — базовый индекс. Можно изменить базовый индекс, написав в начале листа модуля инструкцию Option Base 1. После этого индексы массивов M и V будут начинаться с единицы. Другим способом изменения базового индекса является использование ключевого слова To при объявлении массива:

```
Dim M(1 To 3, 1 To 3) As Single
Dim V(1 To 12) As Integer
```

Значения элементов массива задаются поэлементно.

Например,

```
Dim M(1 To 2,1 To 2) As Integer
M(1,1)=2: M(1,2)=4: M(2,1)=1: B(1,2)=6
```

Удобным способом задания одномерных массивов является функция Array, преобразующая список элементов, разделенных запятыми, в вектор из этих значений, и присваивающая их переменной типа Variant.

Например,

```
Dim A As Variant
A = Array(10,20,30)
B = A(2)
```

Динамические массивы

Иногда в процессе выполнения программы требуется изменять размер массива. В этом случае первоначально массив объявляют как динамический. Для этого при объявлении массива не указывают размерность.

Например:

```
Dim R() As Single
```

В программе следует вычислить необходимый размер массива и связать его с некоторой переменной, например *n,* затем изменить размер динамического массива с помощью оператора ReDim, формат которого:

ReDim [Preserve] ИмяПеременной(Индексы) [As Тип]_
[. ИмяПеременной(Индексы)[As Тип]] ...

Preserve — ключевое слово, используемое для сохранения данных в существующем массиве при изменении значения последней размерности.

ИмяПеременной — имя переменной, удовлетворяющее стандартным правилам именования переменных.

Индексы — размерности переменной массива; допускается описание до 60 размерностей. Аргумент Индексы использует следующий синтаксис:

[Нижний To] Верхний [, [Нижний To] Верхний] ...

Если нижний индекс не задан явно, нижняя граница массива определяется инструкцией Option Base. Если отсутствует инструкция Option Base, нижняя граница массива равняется нулю.

Тип — тип данных массива.

Например, установим границы массива R:

```
ReDim R(1 To  10)
```

Допустимо повторное использование инструкции ReDim для изменения числа элементов и размерностей массива.

6.17. Структурный подход к программированию

С возрастанием количества и размеров программ процесс их разработки стал практически не управляемым. Возникла проблема коренного изменения подходов к созданию больших программных комплексов. Были разработаны строгие правила ведения проектов, которые получили название *структурной методологии.*

Впервые термин «структурное программирование» ввел Эдсгер Дейкстра. Он рассматривал программу как совокупность иерархических абстрактных уровней, которые позволяли четко структурировать программу, выполнять доказательства при ее корректировке, а значит, и повышать надежность функционирования программы, и сокращать сроки ее разработки.

Э. Дейкстра указывал: «Оператор GOTO нужно считать вредным». Его поддержали ученые Н. Вирт (Цюрих) и Хоар (Оксфорд).

Было доказано, что любая программа может быть написана с использованием только простой последовательности операторов, циклической конструкции типа for или while и выбора if или case.

Цели структурного программирования

- Обеспечение *дисциплины* программирования («Структурное программирование — это дисциплина, которую программист навязывает сам себе», — Э. Дейкстра).
- Повышение *эффективности* (например, разбиение на относительно независимые модули).
- Повышение *надежности* (облегчение тестирования и отладки).
- Уменьшение *времени и стоимости* (повышение производительности программистов).
- Улучшение *читабельности* программы.

6.17.1. Основные принципы структурного подхода

1. Принцип абстракции позволяет рассматривать программу по уровням. Верхний уровень показывает наибольшую степень абстракции, упрощает восприятие программы; нижний уровень показывает детали реализации (например, восходящие и нисходящие стратегии программирования).

2. *Разделение программы* на отдельные фрагменты (методы), которые просты по управлению и допускают независимую отладку и тестирование.

3. Строгий *методический подход* (принцип формальности) позволяет изучать программы (алгоритмы) как математические объекты, ускорить принятие решений, избежать ошибок.

4. Возможность участия в результате большого количества людей.

5. Программы должны быть построены так, чтобы можно было вносить в них изменения без участия автора.

6. Возможность демонстрирования работы программы на всех этапах ее создания.

7. Возможность *планирования* работы, уверенность в реальных сроках завершения.

8. Принцип *иерархического упорядочения.*

Структурный подход включает в себя 3 основные составные части:

- нисходящее проектирование;
- структурное программирование;
- сквозной структурный контроль.

Рассмотрим подробнее часть первую. При возрастании сложности и объема программ они становятся слишком большими для анализа и понимания даже при условии правильной организации их структуры. С целью управления процессом создания программ прибегают к делению большой программы на модули. Этот процесс называется *нисходящим проектированием.*

Делить программу на модули необходимо, учитывая следующие *свойства* (или требования) *модулей:*

1. Возможность обособления модуля в исходном и объектном кодах.

2. Наличие имени у модуля и возможность вызова по имени.

3. Модуль должен иметь один вход и один выход.

4. Модуль должен возвращать управление в ту точку, откуда был вызван.

5. Модуль должен иметь возможность вызывать другие модули.

6. Внутренний текст модуля должен быть структурирован (представлять комбинацию простых структур).

7. Модуль должен быть обозрим в исходном тексте (желательно помещаться на одной странице).

8. Модуль не использует историю своих вызовов для управления своим функционированием (сколько бы мы ни вызывали модуль, он делает одно и тоже).

9. Функциональная замкнутость модуля. Идеальный модуль выполняет только одну функцию, но выполняет ее целиком.

10. Модуль должен иметь минимальную возможность обмена информацией с другими модулями.

11. Модуль может использовать глобальные объекты, но не должен их менять (без необходимости).

Итак, *нисходящее проектирование* — это процесс проектирования программной системы как совокупности модулей, основанный на уровнях абстракции, которые сначала представляют собой уровни функций, а затем — уровни модулей. На самой вершине этого процесса находится единственный модуль, который является образом всей системы. Его функция — основная функция всей системы. Мы рассматриваем его функцию и выясняем, чем он должен обладать, чтобы соответствовать свойствам модуля.

Завершение разработки схемы иерархии должно сопровождаться согласованием ее с заказчиком и написанием ее объяснения.

6.17.2. Спецификация программ

Спецификация модуля включает в себя:

1. *Проектирование:*

1.1) описание функции модуля;

1.2) описание интерфейса модуля с другими модулями, т. е. описание тех данных, которыми модуль обменивается с другими модулями при вызовах;

1.3) разработка руководства по модулю для пользователя и утверждение с пользователем.

2. *Планирование:*

2.1) распределение работы между группой исполнителей;

2.2) определение очередности реализации;

2.3) определение способа отладки и способа проверки достоверности результатов.

2.4) проработка тестовых заданий, которые проверяют работу программы. Они должны быть продуманы и разработаны до начала написания программы.

3. *Этап реализации:*

3.1) написание модуля;

3.2) отладка;

3.3) документирование (в соответствии с ЕСПД — единой системой проектной документации);

3.4) сдача.

6.17.3. Метод пошаговой детализации

Метод пошаговой детализации — еще одно средство создания структурированных алгоритмов. Сущность метода — проектирование алгоритма разбивается на ряд последовательных шагов. На *первом шаге* задача рассматривается укрупненно, обобщенно, и описывается полный алгоритм ее решения таким образом, чтобы он представлял простую структуру или комбинацию простых структур. На *втором шаге* мы просматриваем наш алгоритм и те действия, которые недостаточно детализированы, уточняем таким образом, чтобы эти уточнения также представляли собой простую структуру или элементарную комбинацию простых структур. При этом могут быть две формы уточнения: в рамках того же самого алгоритма или отдельно от алгоритма в качестве заголовка выносится действие, которое надо уточнить, и уточняется.

На *третьем* и *последующих шагах* мы повторяем второй шаг для тех действий, которые содержатся в предыдущих уточнениях.

Процесс заканчивается тогда, когда все действия оказываются *элементарными* для программирования.

Правила пошаговой детализации:

1. Не следует спешить сразу дробить задачу на мелкие детали. На каждом шаге нужно делать небольшие уточнения.

2. По мере детализации нужно внимательно следить за данными (можно составить таблицу идентификаторов). Следует избегать использования одинаковых имен в разных смыслах.

3. Надо быть готовым вернуться назад и отказаться от какой-то детализации, которая ведет в тупик.

ГЛАВА 7

КОМПЬЮТЕРНЫЕ СЕТИ

7.1. Общие принципы построения компьютерных сетей

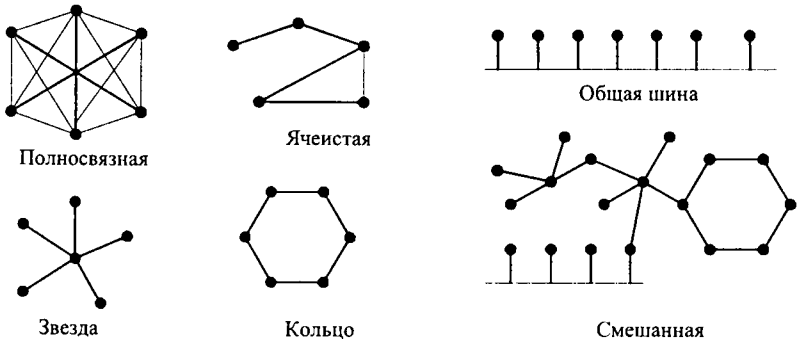
▶ *Сеть ЭВМ* (*компьютерная или вычислительная сеть*) — это совокупность компьютеров, соединенных с помощью каналов связи в единую систему для совместного использования общих информационных и вычислительных ресурсов.

По территориальному признаку компьютерные сети можно разделить на локальные сети (в пределах одной организации), корпоративные сети (при большой территориальной распределенности одной организации), региональные (охватывают территорию одного района или города) и глобальные сети. В глобальных сетях выделяют магистральные сети в пределах стран и объединяющую их вместе сеть Internet.

Любая компьютерная сеть включает в себя систему каналов передачи данных, телекоммуникационное, серверное и клиентское оборудование, называемое узлами сети, сетевые операционные системы, сетевые приложения.

▶ Принцип построения физической структуры компьютерной сети часто называют *топологией сети.* Иными словами, топология это конфигурация графа, узлами которого служат компьютеры и сетевое оборудование, а ребрами — физические связи между ними.

На рис. 7.1 приведены варианты сетевых топологий.



▲

Рис. 7.1

В случае, если один из узлов связан с несколькими другими узлами, это означает, что он должен иметь несколько сетевых интерфейсов, то есть сеть значительно усложняется. В настоящее время полносвязная топология никогда не применяется, а также редко используется кольцевая топология.

Наиболее распространенными топологиями локальных сетей является общая шина и звезда. При объединении нескольких сетей образуется топология, называемая смешанной.

При построении компьютерной сети необходимо решить следующие задачи: выбора кодирования данных, протокола передачи данных, разделения канала передачи между абонентами и маршрутизации передаваемых сообщений.

Кодирование представляет собой изменение формы представления информации с целью ее передачи, хранения или обработки.

▶ *Протокол* — это набор правил взаимодействия, определяющих способ кодирования информации и передачи служебных данных между сетевыми интерфейсами.

▶ *Сетевой интерфейс* — это устройство, обеспечивающее передачу данных, в том числе между программными компонентами.

Разделение канала передачи может производиться при помощи мультиплексирования каналов, либо при помощи мультиплексирования пакетов данных.

▶ *Маршрутизация* представляет собой определение маршрута передачи сообщения в сложных сетях.

Средства сетевого взаимодействия представляются в виде многоуровневого множества модулей, при этом нижние модули отвечают за физическое кодирование, модули более высокого уровня — за адресацию и маршрутизацию, и на самом высоком уровне — за предоставление прикладных услуг пользователю компьютера. При этом модули низкого уровня прозрачны для модулей более высокого уровня, у которых создается иллюзия «непосредственного взаимодействия» (рис. 7.2).

Для каждого уровня существует свой протокол и интерфейсы взаимодействия с нижним и верхним уровнями. Совокупность таких протоколов называется *стеком протоколов*, так как передаваемая информация последовательно преобразуется на всех уровнях и восстанавливается на принимающей стороне в обратном порядке.

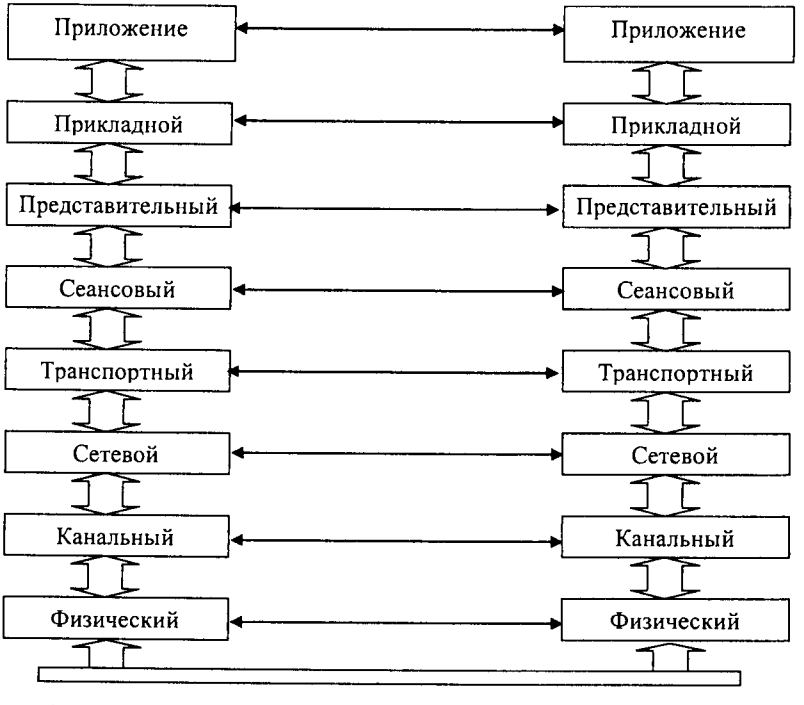


Рис. 7.2

7.1.1. Концепция открытых систем OSI

При построении многоуровневой сетевой системы необходимо обеспечить «взаимопонимание» на каждом из уровней, то есть совместимость протоколов, а также «взаимопонимание» при передаче информации с одного уровня на другой, то есть совместимость интерфейсов. Для этого должно существовать описание уровней, их задач, протоколов и интерфейсов. Фирма-разработчик системы взаимодействия может разработать свой собственный протокол или придерживаться стандарта. Использование собственных протоколов имеет некоторые преимущества, так как дает возможность применить передовые технологии и патентованные разработки, и привязывает потребителя к продукции конкретной фирмы. Но в связи с последним обстоятельством потребители обычно предпочитают систему на основе открытых стандартов.

▶ Системы, обеспечивающие возможность взаимодействия по опубликованным протоколам, называются *открытыми системами*.

К началу 1980-х годов было разработано достаточно много корпоративных и отраслевых стандартов на стеки протоколов (*TCP/IP, IPX/SPX*), и возникла потребность в разработке единой терминологии для их описания. Международная организация стандартов *ISO* (*International Standard Organization*) разработала модель взаимодействия открытых систем *OSI* (*Open System Interconnection*). Такую модель взаимодействия часто называют также моделью *ISO/OSI*. В настоящее время модель *OSI* используется в качестве основы при описании средств сетевого взаимодействия. Хотя ни один из реальных стеков протоколов не соответствует ей полностью, их принято рассматривать в проекции на модель *OSI*.

В данной модели средства сетевого взаимодействия разделены на 7 уровней: *прикладной*, *представительный*, *сеансовый*, *транспортный*, *сетевой*, *канальный*, *физический*. Согласно модели для передачи информации прикладная программа обращается к протоколу прикладного уровня, который на основании этого формирует сообщение, состоящее из заголовка и поля данных. В заголовке обычно содержится служебная информация, например, адрес назначения. После формирования сообщение направляется «вниз» по стеку, и на каждом из уровней к нему добавляется заголовок, а иногда также и завершающая фраза. В стандартах *ISO* для обозначения сообщений используется термин «протокольный блок данных» (*Protocol Data Unit, PDU*), и существуют специальные названия для *PDU* каждого уровня.

Ниже описано назначение каждого из уровней.

Физический уровень (*Physical layer*)

На этом уровне выполняется согласование электрических сигналов между сетевыми интерфейсами и способы их синхронизации при передаче на расстояния. Протоколы физического уровня определяют и набор правил представления информации в виде изменения той или иной физической величины, например, электрического напряжения в кабеле. Также физический уровень определяет требования к характеристикам физической среды передачи данных (кабелей, разъемов) и передающего оборудования.

Канальный уровень (*Data-link layer*)

На канальном уровне организуется совместный доступ к среде передачи данных, проверяется её доступность. В связи с возможностью на-

чала одновременной передачи данных несколькими независимыми источниками по единой среде возможно столкновение данных (*коллизия*). Сетевые устройства, подключаемые к сегменту одной физической среды, называют *доменом коллизий*. Задача предотвращения коллизий в технологии *Ethernet* решается с помощью протокола *CSMA/CD* (*коллективного доступа к среде с обнаружением коллизий*). Выполняются также задачи обнаружения и коррекции ошибок и адресации (передачи информации между двумя заданными компьютерами), но не маршрутизации. Это означает, что доставка информации происходит только в пределах одного сегмента сети, имеющего строго определенную топологию и построенного на основе конкретной технологии. Для возможности решения задач коррекции ошибок и разделения среды передачи данных, передаваемые на канальном уровне данные делятся на блоки, называемые *кадрами* (либо *фреймами*). Этот уровень состоит из двух подуровней: *LLC* (*Logical Link Control*, *управление логическим соединением*) и *MAC* (*Media Access Control*, *управление доступом к физической среде*).

Функции канального уровня реализуются средствами телекоммуникаций: *коммутаторами*, *концентраторами*, *мостами*, *сетевыми адаптерами* и их драйверами.

Сетевой уровень (*Network layer*)

На сетевом уровне решаются задачи маршрутизации и объединения разнородных по топологии и оборудованию сетей, то есть передачи блоков информации между компьютерами, находящимися в различных сегментах сети, которые могут работать на основе различных сетевых технологий.

При обсуждении проблем сетевого уровня под термином *сеть* понимают совокупность компьютеров, объединенных друг с другом на основе одной из стандартных технологий канального уровня, поэтому задачей сетевого уровня является организация *межсетевого взаимодействия* (*internetworking*).

Сети соединяются между собой специальными устройствами — *роутерами* (*route*, маршрут) или *маршрутизаторами*. Чтобы передать блок информации от отправителя к получателю, находящемуся в другой сети, необходимо передать его через несколько маршрутизаторов, то есть совершить несколько *транзитных передач* (*hop*, *прыжок*). Передача от отправителя к маршрутизатору, между маршрутизаторами и от последнего маршрутизатора к получателю осуществляется средствами ка-

нального уровня, а на самих маршрутизаторах с использованием данных протокола сетевого уровня осуществляется выбор дальнейшего направления движения блока информации, то есть маршрутизация. В качестве маршрутизаторов могут выступать компьютеры (маршрутизирующие серверы) и специальные телекоммуникационные устройства.

▶ Сообщения сетевого уровня принято называть *пакетами*.

На сетевом уровне выделяют *сетевые протоколы* (*IP, IPX*), *протоколы поиска маршрутов* (*RIP*) и *протоколы разрешения адресов* (*ARP*). Сетевые протоколы обеспечивают адресацию пакетов и их продвижение по сети. В настоящее время главным сетевым протоколом является *IP* (*Internet Protocol*). Протоколы маршрутизации выполняют обмен служебной информацией между маршрутизаторами. Протоколы разрешения адресов служат для определения адреса компьютера в локальной сети (для доставки на канальном уровне) по его сетевому адресу.

Транспортный уровень (*Transport layer*)

На транспортном уровне решается задача обеспечения надежной и безошибочной связи, совместного использования одного сетевого соединения различными программами и протоколами сетевого уровня и коррекции ошибок.

При передаче пакетов через сложную сеть возможны потери, дублирования или подмены пакетов, которые обнаруживаются средствами протоколов сетевого уровня. Также этот протокол позволяет распределять принимаемые данные между программами, совместно работающими с сетью. Это достигается путем вложения в пакет сетевого уровня дополнительной информации, в частности, о номере данного пакета в цепочке взаимодействия между приложениями и номере этого виртуального соединения.

Стек протоколов обычно содержит несколько протоколов транспортного уровня, имеющих различные функции.

Самые распространенные протоколы транспортного уровня это *TCP* (*Transmission Control Protocol, протокол управления передачей*) и *UDP* (*User Datagram Protocol, протокол передачи пользовательских дейтаграмм*).

Протоколы нижних четырех уровней, как правило, называют *транспортной подсистемой*, так как они полностью решают задачу передачи блоков информации между узлами сети, а протоколы верхнего уровня решают прикладные задачи.

Сеансовый уровень (*Session layer*)

На сеансовом уровне организуется синхронизация взаимодействия приемника и передатчика, которая позволяет разбить сложный процесс взаимодействия на несколько сеансов. Примером может являться процесс возобновления приема информации с восстановлением с прежнего места в случае разрыва. На практике функции сеансового уровня требуются достаточно редко и, как правило, не выделяются в отдельный протокол, а реализуются в протоколе прикладного уровня.

Представительный уровень (*Presentation layer*)

На представительном уровне проводится изменение формы представляемой информации, например, шифрование или приведение к единой кодировке символов. Примером этого протокола является *SSL* (*Secure Socket Layer*, *уровень безопасных соединений*).

Прикладной уровень (*Application layer*)

Протоколы прикладного уровня обеспечивают приложениям доступ к разделяемым ресурсам. Прикладной протокол представляет собой набор команд, которые клиент передает серверу для получения определенной информации. Примерами протоколов сетевого уровня являются *HTTP* (*Hyper Text Transfer Protocol*, *протокол пересылки гипертекста*), *FTP* (*File Transfer Protocol*, *протокол пересылки файлов*) и др.

Основные характеристики компьютерных сетей

Для того чтобы сеть успешно справлялась с этой задачей, она должна отвечать требованиям по производительности, надежности и др.

Производительность сети определяет объем передаваемых данных и время, требуемое на их передачу. Для оценки производительности используются числовые характеристики — время реакции сети, средняя пропускная способность, максимально возможная пропускная способность, задержка передачи.

Надежность означает вероятность того, что сеть выполняет свои функции. Надежности технических устройств обычно характеризуется *временем наработки на отказ* и *коэффициентом готовности* (процент времени, в течение которого система может быть использована).

Надежность сложных систем также характеризуется вероятностью доставки сообщения адресату.

Безопасность означает невозможность несанкционированного доступа к данным и обеспечение надежности и устойчивости к преднамеренным разрушающим действиям.

Расширяемость это возможность сравнительно легкого добавления новых элементов сети.

Масштабируемость это возможность значительного наращивания размеров сети, в том числе путем увеличения числа сегментов.

Прозрачность означает возможность использования ресурсов сети одним и тем же способом, независимо от их фактического размещения — на локальном компьютере или в сети. При этом пользователь как бы «не замечает» сети, работая непосредственно с ресурсами.

Поддержка разных видов трафика — возможность совмещения функций различных сетей, например, телефонной, телевизионной и компьютерной.

Управляемость — возможность централизованного обнаружения и устранения неполадок, распределения ресурсов и полномочий между пользователями.

Совместимость — возможность взаимодействия с различным оборудованием и программным обеспечением.

7.1.2. Аппаратное обеспечение компьютерных сетей

Типы линий связи

▶ *Линия связи* состоит в общем случае из физической среды, по которой передаются электрические информационные сигналы, аппаратуры передачи данных и промежуточной аппаратуры. Синонимом термина линия связи (*line*) является термин *канал связи* (*channel*).

▶ *Физическая среда передачи данных* (*physical medium*) может представлять собой кабель, то есть набор проводов, изоляционных и защитных оболочек и соединительных разъемов, а также земную атмосферу или космическое пространство, через которые распространяются электромагнитные волны.

В зависимости от среды передачи данных линии связи разделяются на:

1. *Проводные кабельные* (медные и волоконно-оптические);
2. *Беспроводные* радиоканалы наземной и спутниковой связи.

Проводные линии связи на основе кабелей представляют собой несколько проводников, заключенных в экранирующую и изолирующую оплетки. В компьютерных сетях применяются три типа кабелей: кабели на основе скрученных пар медных проводов (*витая пара*); *коаксиальные* кабели, и *волоконно-оптические* кабели.

Радиоканалы образуются с помощью передатчика и приемника радиоволн. Разные типы каналов отличаются друг от друга частотным

диапазоном, который определяет дальность распространения радиоволн. Для компьютерной связи используется диапазон ультракоротких волн или волн сверхвысокой частоты. В этих диапазонах сигналы не отражаются от ионосферы, поэтому для организации канала необходима прямая видимость между передатчиком и приемником, или ретрансляция.

Формирование электрических (или иных) сигналов для передачи по линии связи осуществляется *аппаратурой передачи данных (Data Circuit terminating Equipment)*. Примерами *DCE* являются модемы, и различные устройства подключения.

Подготовка данных для передачи осуществляется *оконечным оборудованием данных (Data Terminal Equipment)*, которое обычно не включают в состав линий связи.

Промежуточное оборудование линий связи служит для усиления сигнала, проходящего через линию, а также для организации совместного использования линий связи (мультиплексирования и коммутации).

В зависимости от способа формирования и приема сигналов линии связи делятся на *цифровые* и *аналоговые*. В цифровых линиях данные представляются дискретными сигналами, имеющими конечное число состояний. Это означает, что вся переносимая сигналом информация заключена в значениях этого сигнала в заданные равномерно распределенные моменты времени, причем сигнал может принимать конечное число значений. В аналоговых линиях используются аналоговые сигналы, имеющие непрерывный диапазон значений и непрерывные во времени.

При передаче данных в аналоговой форме сигналы имеют более узкий спектр, поэтому их используют в линиях связи с узкой полосой пропускания, например, в телефонных сетях. Цифровые сигналы позволяют достичь более высокой скорости передачи данных, но имеют более широкий спектр.

Типы кабелей

Самыми распространенными кабелями для построения локальных компьютерных сетей можно считать кабели *UTP (Unshielded Twisted Pair, неэкранированная витая пара)*.

Медный неэкранированный кабель *UTP*, наряду с тонким коаксиальным кабелем, является наиболее употребительным в современных локальных сетях. В стандартах описаны пять категорий данного типа кабеля. Кабели первых двух категорий были описаны в ранних стандартах, но в последней редакции признаны устаревшими.

Кабели *категории 3* имеют рабочий диапазон до 16 МГц и предназначены для передачи данных и голоса. Шаг скрутки проводников установлен 3 витка на 1 фут. В настоящее время этот кабель используется для передачи голоса и данных, в том числе в сетях Ethernet со скоростью передачи 10 Мбит/с.

Кабели *категории 4* являются некоторым улучшением категории 3 и имеют рабочий диапазон 20 МГц.

Кабели *категории 5* — наиболее распространенный в настоящее время вид кабелей типа витая пара, ориентированный на высокоскоростные протоколы и имеющий рабочий диапазон 100 МГц.

Кабели *категорий 6 и 7* имеют рабочий диапазон 200 и 600 МГц и лучшие характеристики затухания и помехозащищенности, но используются редко из-за своей дороговизны.

Волновое сопротивление кабелей витой пары составляет 100 Ом (стандарт *ISO* допускает также сопротивление 120 Ом). Все кабели выпускаются в 4-парном исполнении, каждая из пар имеет свой цвет. Для соединения кабелей с оборудованием используются вилки и розетки *RJ*-45.

Кабели *STP* (*Shielded Twisted Pair*, *экранированная витая пара*) имеют лучшие, по сравнению с неэкранированной витой парой, характеристики. Основным стандартом, определяющим параметры кабелей данного типа, является стандарт фирмы *IBM*, в котором кабели разделены на девять типов.

Экранированные кабели Type 1 стандарта *IBM* имеют волновое сопротивление 150 *Ом*, поэтому для работы с ними нужно соответствующее оборудование. Данные кабели поддерживаются некоторыми сетевыми технологиями (*Fast Ethernet*, *TokenRing* и др.), наряду с кабелями *UTP*.

Коаксиальные кабели широко используются не только в компьютерных сетях, но и для передачи высокочастотных телевизионных сигналов.

Кабель *RG*-8 и *RG*-11 — «толстый» коаксиальный кабель для сетей *Ethernet* 10*Base*-5. Имеет волновое сопротивление 50 Ом и внешний диаметр 0.5 дюйма. Это достаточно дорогой кабель с высокими характеристиками.

Кабели *RG*-58/*U* (сплошной тонкий проводник), *RG*-58 *A*/*U* (многожильный проводник) и RG-58 C/U — тонкий коаксиальный кабель для сетей *Ethernet* 10*Base*-2 с волновым сопротивлением 50 Ом. Для соединения кабеля с оборудованием используется разъем типа *BNC*. Кабель *RG*-59 с волновым сопротивлением 50 Ом для передачи телевизионных сигналов.

Волоконно-оптические кабели состоят из центрального проводника света (волокна), окруженного другим проводником — оболочкой. Оболочка обладает меньшим показателем преломления, чем сердцевина, поэтому излучение не выходит за пределы волокна.

Различают *одномодовое* волокно (дорогое, очень тонкого диаметра, с полосой пропускания сотни гигагерц); *многомодовое* волокно (с более широким сердечником и меньшей полосой пропускания 500-800 МГц.

В *многомодовом* волокне из-за относительно больших размеров электромагнитная волна высокой частоты может распространяться в нескольких режимах (модах), с разными скоростями, что приводит к невозможности передачи информации. Поэтому верхняя граничная частота такого волокна ограничена нижней частотой возникновения высших мод.

В качестве источников света в таких кабелях используют светодиоды и полупроводниковые лазеры. Для передачи информации используется свет с длиной волны 1150 нм (лазеры), 1300 нм и 850 нм. Для присоединения кабеля к оборудованию используются разъемы *MIC*, *ST*, и *SC*.

Типы сетевых адаптеров

Сетевой адаптер (сетевая плата, *Network Interface Card*, *NIC*) выполняет функции физического и канального уровня, то есть осуществляет формирование электрических сигналов и контроль доступа к среде передачи данных.

Управление работой сетевого адаптера осуществляется соответствующим драйвером. Взаимодействие программ и ОС с драйвером сетевого адаптера осуществляется программным модулем *NDIS* в ОС Windows.

В функции сетевого адаптера и его драйвера входит: прием данных от *LLC*-подуровня, оформление кадра *MAC*-подуровня, логическое кодирование, физическое кодирование и передача сигнала в кабель, прием сигнала, логическое декодирование, проверка контрольной суммы, передача принятых кадров *LLC*-подуровню.

Распределение обязанностей между аппаратурой и драйвером адаптера может проводиться по-разному. Чем больше функций выполняется аппаратно, тем производительнее (и дороже) адаптер. Дорогие серверные адаптеры содержат встроенный процессор и выполняют почти все функции, не используя ресурсы центрального процессора компьютера.

Для различных технологий используются различные сетевые адаптеры.

Как и любые устройства, подключаемые к компьютеру, сетевые адаптеры используют ресурсы — *IRQ*, *DMA* и порты ввода-вывода.

7.1.3. Основы технологий локальных сетей

7.1.3.1. Локальные сети Ethernet

В зависимости от направления возможной передачи данных способы передачи данных по линии связи делятся на следующие типы:

- *симплексный* — передача осуществляется по линии связи только в одном направлении;
- *полудуплексный* — передача ведется в обоих направлениях, но попеременно во времени. Примером такой передачи служит технология *Ethernet*;
- *дуплексный* — передача ведется одновременно в двух направлениях.

Дуплексный режим — наиболее универсальный и производительный способ работы канала. Самым простым вариантом организации дуплексного режима является использование двух независимых физических каналов (двух пар проводников или двух световодов) в кабеле, каждый из которых работает в симплексном режиме, то есть передает данные в одном направлении. Именно такая идея лежит в основе реализации дуплексного режима работы во многих сетевых технологиях, например *Fast Ethernet* или *ATM*.

Иногда такое простое решение оказывается недоступным или неэффективным. Чаще всего это происходит в тех случаях, когда для дуплексного обмена данными имеется всего один физический канал, а организация второго связана с большими затратами. Например, при обмене данными с помощью модемов через телефонную сеть у пользователя имеется только один физический канал связи с автоматической телефонной станцией — двухпроводная линия, и приобретать второй вряд ли целесообразно. В таких случаях дуплексный режим работы организуется на основе разделения канала на два логических подканала.

Рассмотрим подробнее самую распространенную технологию построения локальных компьютерных сетей *Ethernet*.

Когда говорят *Ethernet*, то под этим обычно понимают любой из вариантов этой технологии. В более узком смысле *Ethernet* — это сетевой стандарт, основанный на экспериментальной сети *Ethernet Network*, которую фирма *Xerox* разработала и реализовала в 1975 г.

В 1980 году фирмы *DEC*, *Intel* и *Xerox* совместно разработали и опубликовали стандарт *Ethernet* версии II для сети, построенной на основе коаксиального кабеля, который стал последней версией фирменного стандарта *Ethernet*. Поэтому фирменную версию стандарта *Ethernet* называют стандартом *Ethernet DIX* или *Ethernet* II.

На основе стандарта *Ethernet DIX* был разработан стандарт *IEEE* 802.3, который во многом совпадает со своим предшественником, но некоторые различия все же имеются. В зависимости от типа физической среды стандарт *IEEE* 802.3 имеет различные модификации — 10*Base*-5, 10*Base*-2, 10*Base-T*, 10*Base-FL*, 10*Base-FB*. Цифра 10 в названии обозначает максимальную пропускную способность 10 Мбит/с.

В 1995 г. был принят стандарт *Fast Ethernet*, с пропускной способностью сети до 100 Мбит/с. Принятый в 1998 г. стандарт *Gigabit Ethernet* позволяет осуществлять передачу данных со скоростью до 1000 Мбит/с.

Физические спецификации технологии *Ethernet* на сегодняшний день включают следующие среды передачи данных:

10Base-5 — коаксиальный кабель диаметром 0.5 дюйма, называемый «толстым» коаксиалом. Имеет волновое сопротивление 50 Ом. Максимальная длина сегмента — 500 м (без повторителей).

10Base-2 — коаксиальный кабель диаметром 0,25 дюйма, называемый «тонким» коаксиалом. Имеет волновое сопротивление 50 Ом. Максимальная длина сегмента — 185 м (без повторителей).

10Base-T — кабель на основе неэкранированной витой пары (*Unshielded Twisted Pair, UTP*). Образует звездообразную топологию на основе концентратора. Расстояние между концентратором и конечным узлом — не более 100 м.

10Base-F — волоконно-оптический кабель. Топология аналогична топологии стандарта 10Base-*T*. Имеется несколько вариантов этой спецификации — *FOIRL* (расстояние до 1000 м), *10Base-FL* (расстояние до 2000 м), *10Base-FB* (расстояние до 2000 м).

Стандарт 10Base-5 в основном соответствует экспериментальной сети *Ethernet* фирмы *Xerox* и может считаться классическим *Ethernet*. Он использует в качестве среды передачи данных коаксиальный кабель с волновым сопротивлением 50 Ом, диаметром центрального медного провода 2, 17 мм и внешним диаметром около 10 мм («толстый» *Ethernet*). Такими характеристиками обладают кабели марок *RG*-8 и *RG*-11.

Слово *Base* обозначает метод передачи на одной базовой частоте 10 МГц в отличие от методов, использующих несколько несущих частот, которые называются *Broadband, широкополосными*. Последний символ в названии стандарта физического уровня обозначает тип кабеля.

Для построения сетей *Fast Ethernet* используются следующие технологии.

100Base-FX — многомодовое оптоволокно. Каждый узел соединяется с сетью двумя оптическими волокнами, передача по которым осу-

ществляется в двух различных направлениях. В стандарте предусматривается использование метода кодирования 4B/5B, в котором каждые 4 бита заменяются на 5 бит. Физическое кодирование проводится по методу *NRZI*.

Оборудование 100Base-*FX* не совместимо с 10 Мбит *Ethernet* на оптоволокне.

100Base-TX — витая пара *UTP* категории 5 или *STP* типа 1.

Для соединения используются две витые пары, логическое кодирование 4B/5B и физическое кодирование *MLT*-3. Оборудование 100Base-FX может работать совместно с оборудованием 10 Мбит Ethernet. Для выбора режима работы предусмотрена функция автопереговоров. Переговорный процесс заключается в том, что одно из устройств посылает определенный сигнал, кодирующий предлагаемый режим работы, начиная с самого приоритетного (высокоскоростного). Если этот режим не поддерживается, то происходит попытка установить связь в менее приоритетном режиме.

100Base-T4 — витая пара категории 3 (или 5).

Характеристики *Gigabit Ethernet* следующие.

1000Base-SX — многомодовое оптоволокно, длина волны передатчика 850 нм. Предусматривается использование двух различных типов кабеля: с диаметром внутреннего проводника 62.5 и 50 мм и предельной длиной оптоволоконного сегмента 220 и 500 м соответственно (это ограничение менее жесткое, чем ограничение на диаметр сети, но ограничение на диаметр сети может быть преодолено при использовании коммутаторов).

1000Base-LX — многомодовое или одномодовое оптоволокно, длина волны передатчика 1300 нм. Ограничение на максимальную длину отрезка кабеля — 550 м для многомодового оптоволокна 50 мм, 400 м для многомодового оптоволокна 52.5 мм и 5000 м для одномодового.

1000Base-CX — экранированная витая пара *STP* с волновым сопротивлением 150 Ом. Максимальная длина отрезка кабеля 25 м.

1000Base-TX — неэкранированная витая пара, максимальная длина отрезка кабеля 100 м.

Рассмотрим общие для всех указанных технологий принципы передачи данных в сетях *Ethernet*. В таких сетях используется метод доступа к среде передачи данных, называемый *методом коллективного доступа с опознаванием несущей и обнаружением коллизий* (*CSMA/CD, Carrier Sense Multiply Access / Collision Detection*).

Этот метод применяется исключительно в сетях с логической общей шиной, к которым относятся также и радиосети, породившие этот ме-

тод. Все компьютеры такой сети имеют непосредственный доступ к общей шине, поэтому она может быть использована для передачи данных между любыми двумя узлами сети. Одновременно все компьютеры сети имеют возможность немедленно (с учетом задержки распространения сигнала по физической среде) получить данные, которые любой из компьютеров начал передавать на общую шину. Все данные, передаваемые по сети, помещаются в кадры определенной структуры и снабжаются уникальным адресом станции назначения. Чтобы получить возможность передавать кадр, станция должна убедиться, что разделяемая среда свободна. Это достигается прослушиванием основной гармоники сигнала, которая также называется *несущей частотой* (*CS, Carrier Sense*). Признаком незанятости среды является отсутствие на ней несущей частоты, которая при манчестерском способе кодирования равна 5 - 10 МГц, в зависимости от последовательности единиц и нулей, передаваемых в данный момент. Если среда свободна, то узел имеет право начать передачу кадра. В классической сети *Ethernet* на коаксиальном кабеле сигналы передатчика распространяются в обе стороны, так что все узлы сети их получают. После окончания передачи кадра все узлы сети обязаны выдержать технологическую паузу в 9,6 мкс. Эта пауза, называемая также *межкадровым интервалом*, нужна для приведения сетевых адаптеров в исходное состояние, а также для предотвращения монопольного захвата среды одной станцией. После окончания технологической паузы узлы имеют право начать передачу своего кадра, так как среда свободна.

При этом возможна ситуация, когда две станции одновременно пытаются передать кадр данных по общей среде. Механизм прослушивания среды и пауза между кадрами не гарантируют от возникновения такой ситуации, когда две или более станции одновременно решают, что среда свободна, и начинают передавать свои кадры. Говорят, что при этом происходит коллизия (*collision*), так как содержимое обоих кадров сталкивается на общем кабеле, и происходит искажение информации.

Чтобы корректно обработать коллизию, все станции одновременно наблюдают за возникающими на кабеле сигналами. Если передаваемые и наблюдаемые сигналы отличаются, то фиксируется обнаружение коллизии (*CD, Collision Detection*). Для увеличения вероятности скорейшего обнаружения коллизии всеми станциями сети станция, которая обнаружила коллизию, прерывает передачу своего кадра (в произвольном месте, возможно, и не на границе байта) и усиливает ситуацию коллизии посылкой в сеть специальной последовательности из 32 бит, называе-

мой *jam* - последовательностью. Если 16 последовательных попыток передачи кадра вызывают коллизию, то передатчик должен прекратить попытки и отбросить этот кадр.

Четкое распознавание коллизий всеми станциями сети является необходимым условием корректной работы сети *Ethernet*. Если передающая станция не распознает коллизию и решит, что кадр данных ею передан верно, то этот кадр данных будет утерян. Скорее всего, искаженная информация будет повторно передана каким-либо протоколом верхнего уровня, например, транспортным или прикладным, работающим с установлением соединения, но через значительно более длительный интервал времени по сравнению с микросекундными интервалами, которыми оперирует протокол *Ethernet*. Поэтому если коллизии не будут надежно распознаваться узлами сети *Ethernet*, то это приведет к заметному снижению полезной пропускной способности данной сети.

7.1.3.2. Беспроводные локальные сети

Существует несколько стандартов на технологии беспроводных локальных сетей, наиболее распространенными из которых являются *Bluetooth* и *IEEE* 802.11.

Bluetooth — технология, разработанная консорциумом фирм-производителей устройств мобильной связи *Bluetooth Special Interest Group*. Передача данных осуществляется на частоте 2.4 ГГц на расстояния до 10 м (существуют модификации стандарта, увеличивающие расстояние до 100 м.) Скорость передачи данных до 432.6 Кбит/с (до 721 Мбит/с в асимметричном режиме).

Изначально данная технология была ориентирована на подключение к компьютеру мобильных устройств — карманных компьютеров и телефонов, но в настоящее время разработчики активно продвигают ее в качестве основы построения сети для нескольких офисов. Сейчас уже производятся принтеры с беспроводным доступом и мосты для подключения Bluetooth-адаптеров к Ethernet.

В основу стандарта 802.11 положена сотовая архитектура, причем сеть может состоять как из одной, так и нескольких ячеек. Каждая сота управляется базовой станцией, называемой *точкой доступа* (*AP*, *Access Point*), которая вместе с находящимися в пределах радиуса ее действия рабочими станциями пользователей образует *базовую зону обслуживания* (*BSS*, *Basic Service Set*). Точки доступа многосотовой сети взаимодействуют между собой через *распределительную систему* (*DS*,

Distribution System), представляющую собой эквивалент магистрального сегмента кабельных компьютерных сетей. Вся инфраструктура, включающая точки доступа и распределительную систему, образует расширенную зону обслуживания (*Extended Service Set*).

Стандартом предусмотрен также односотовый вариант беспроводной сети, который может быть реализован и без точки доступа, при этом часть ее функций выполняется непосредственно рабочими станциями.

Для обеспечения перехода мобильных рабочих станций из зоны действия одной точки доступа к другой в многосотовых системах предусмотрены специальные процедуры сканирования, активного и пассивного прослушивания эфира, присоединения, однако строгих спецификаций по их реализации стандарт 802.11 не предусматривает.

Стандартом *IEEE* 802.11 предусмотрен целый комплекс мер безопасности передачи данных под общим названием *Wired Equivalent Privacy* (*WEP*). Он включает средства противодействия несанкционированному доступу к сети, механизмы и процедуры аутентификации, а также шифрование информации при её передаче.

Принятые впоследствии (а также разрабатываемые в настоящее время) дополнительные спецификации 802.11a — 802.11h предусматривают использование других частотных диапазонов и увеличение максимальной пропускной способности (до 54 МБит/с и более) за счет расширения спектра и использования других способов кодирования.

7.2. Основы Internet-технологии

7.2.1. История возникновения и развития Internet

В 1969 г. Агентство перспективных исследований (*ARPA*) Министерства обороны США разработало и построило новую сеть *ARPANet*, ставшую прообразом Internet. Сеть имела следующие особенности:

- децентрализованное управление и распределенность информационных ресурсов;
- использование телефонных каналов связи;
- пакетная коммутация;
- специально разработанный протокол *IP* (*Internet Working protocol*).

В середине 1970-х г.г. было выпущено описание в открытой технической печати сети *ARPANet*, протокола *IP* и выполнено подключение к сети *ARPANet* гражданских государственных и академических организаций.

В начале 1980-х г.г. на технологической базе *APRANet* создана общая сеть, объединившая множество локальных и корпоративных сетей США и получившая название *Internet* (*межсетевое пространство*). В конце 1980-х гг. подключилось к Internet большинство стран мира.

В 1989 г. произошло особо знаменательное событие — разработка Тимом Бернерс-Ли *WWW*-технологии (*World Wide Web, всемирная паутина*). До этого времени пользователями Internet могли быть только специалисты, знакомые с командами операционной системы Unix. После практической реализации в 1993 г. WWW-технологии началось массовое освоение Internet непрофессионалами в области компьютерных технологий.

В дальнейшем был снят запрет на подключение к Internet пользователей, не имеющих специального разрешения, после чего в Internet стал активно вовлекаться бизнес (в настоящее время около 80 % ресурсов Internet финансируются коммерческими организациями).

После 2000 г. количество пользователей Internet превысило 300 млн. чел., и число подключившихся пользователей на сегодняшний день составляет более 400 млн. чел.

Структурно Internet представляет собой совокупность множества локальных и региональных сетей, принадлежащих к различным организациям и государствам, связанных между собой различными каналами связи на основе протокола *IP*; совокупность хостов (от англ. *host* — хозяин) — серверов, предоставляющих информационные и коммуникационные услуги и ресурсы по запросу клиента. Программное обеспечение Internet делится на клиентское, которое обеспечивает передачу запросов к серверу и получение ответов от него, и серверное, обеспечивающее предоставление сетевых услуг сервером по запросу клиента.

7.2.2. Базовые протоколы и адресация в Internet

Передача данных по сети Internet осуществляется на основе семейства протоколов *TCP/IP*. *IP*-протокол– межсетевой протокол, отвечающий за передачу данных. *TCP*-протокол– протокол управления передачей данных.

Данные в Internet передаются в виде коротких пакетов (до 1,4 кбайт). В заголовке каждого пакета вместе со служебной информацией (контрольная сумма) указывается *IP*-адрес получателя. Каждый клиент Internet имеет уникальный *IP*-адрес. *IP*-адрес представляет собой 4-х байтовое число, которое представляется чаще всего в виде 4-х чисел от

0 до 255 (например, 10.83.114.10), конкретизация адреса осуществляется слева направо.

Наряду с *IP*-адресами используются смысловые (семантические), или доменные, адреса (адреса-имена).

▶ *Домен* — совокупность компьютеров, принадлежащих одному уровню.

DNS (Domain Name System) — доменная система имен — система преобразования доменных адресов в *IP*-адреса. *DNS* представляет собой распределенную базу данных, в которой каждому *IP*-адресу ставится в соответствие один или несколько доменных адресов. Рассмотрим доменный адрес www.yandex.ru. В доменной адресации уточнение адреса сервера происходит справа налево. Иерархическая структура доменов для данного примера:

ru — домен верхнего уровня,

yandex — домен второго уровня,

www — имя протокола передачи данных по Internet.

Домены подразделяются на организационные и территориальные.

Организационные домены верхнего уровня:

.com (*commercial*) — коммерческие организации;

.edu (*educational*) — образовательные организации;

.gov (*govemental*) — правительственные организации;

.mil (*military*) — военные организации;

.net (*network*) — организации, предоставляющие сетевые услуги;

.org (*organization*) — организации, не относящаяся к вышеназванным видам.

Территориальные домены верхнего уровня:

.ru (*Russia*) — Россия;

.su (*Soviet Union*) — страны бывшего СССР, ныне ряд государств СНГ;

.uk (*United Kingdom*) — Великобритания;

.ua (*Ukraine*) — Украина;

.bg (*Bulgaria*) — Болгария;

.hu (*Hungary*) — Венгрия;

.uz (*Uzbekistan*) — Узбекистан;

.de (*Deutchland*) — Германия.

Всего зарезервировано около 300 обозначений, из которых используется около 150.

Как правило, пакеты данных от одного компьютера к другому проходят в Internet через цепочку промежуточных компьютеров (до 25) —

маршрутизаторов, или роутеров (*route* — маршрут). Каждый из роутеров анализирует *IP*-адрес в заголовке пакета и с помощью таблицы маршрутизации выбирает один из подключенных к нему компьютеров, которому и пересылает пакет.

Значительную часть информационного пространства занимает гипертекстовая мультимедийная технология для работы с информационными ресурсами Internet *WWW*.

Главным понятием этой технологии является *гипертекст* — документ, связанный с фрагментами этого же документа или другими документами через систему выделенных слов (гиперссылок). Документ, включающий не только текст, но и другие данные, часто называют гипермедиа — гипертекстовый мультимедийный документ (*web*-документ, *web*-страница). Гипермедиа формируется на специально разработанном языке *HTML* (*Hyper Text Markup Language, язык разметки гипертекстов*) и имеет одно из расширений: .htm, .html, .shtml.

Документ, подготовленный на *HTML*, представляет собой текстовый файл, содержащий собственно текст, несущий информацию читателю, и теги (флаги разметки) — команды-инструкции по представлению (отображению) текста, а также ссылки на другие документы (файлы).

Для просмотра и работы с web-документами используются специальные программы — браузеры (*browse, осматривать*). Браузер работает с сервером по протоколу *HTTP* (*Hyper Text Transfer Protocol, протокол для передачи гипертекстовых документов*). К наиболее популярным браузерам относятся *Internet Explorer*, *Netscape Navigator*, *Opera*, *Mozilla* и др. Работа браузера заключается в посылке запроса к серверу, заданному пользователем, и интерпретации получаемых *HTML*-файлов. Также браузер выполняет следующие функции кэширования, т.е. хранения уже загруженных документов на жестком диске компьютера, формирования списка уже просмотренных и часто посещаемых документов, для обеспечения к ним быстрого возврата (инструменты: «журнал», «адрес», «переход», «избранное»), представления документа в виде исходного *HTML*-кода и его редактирования; автоматизированный или ручной подбор кодировки отображения страницы. Если документ подготовлен в одной кодировке, а представляется в другой, то текст будет выглядеть как бессмысленный набор знаков. Браузер позволяет подбирать кодировку документа автоматически или вручную. В строке адреса браузера вводится *URL* (*Universal Resource Locator* — универсальный определитель ресурсов), представляющий собой способ адресации ресурсов в сети Internet и однозначно определяющий их мес-

тонахождение. Структура *URL*: *протокол* : // *адрес сервера* / *локальный путь к файлу*.

7.2.3. Организация подключения и работы с Internet

Для подключения к Internet необходимы следующие элементы.

1. Internet-провайдер *(Internet Service Provider, ISP)* — компания, предоставляющая услуги по подключению к Internet.

2. Физический канал связи между организацией и Internet-провайдером.

3. Устройство, обеспечивающее связь по каналу.

Наиболее распространены следующие каналы связи и способы подключения к оборудованию провайдера:

- коммутируемая линия и режим *dail-up*, «дозвониться», средняя скорость передачи/приема данных — около 33–52 Кбит/сек;
- по выделенной линии, скорость передачи/приема данных зависит от типа канала.

Использование цифрового канала *ISDN* обеспечивает скорость 64/128 Кбит/сек, а при технологии *ADSL* скорость передачи достигает до 8 Мбит/с. Вне зависимости от способа доступа к Internet пользователю требуется *IP*-адрес, который, в зависимости от способа получения, может быть статистическим, и фиксируется за клиентом, или динамическим, и меняется каждый раз при подключении к Internet. В функции провайдера входят: предоставление доступа к Internet; информационная поддержка существующих и потенциальных клиентов; программная и техническая поддержка и настройка оборудования; дополнительный сервис и обеспечение работы почтового ящика клиента, предоставление дополнительного дискового пространства на сервере (до 10–20 Мбайт) и др.; учетно-финансовые функции и учет времени работы в Internet, трафика и др.

Для подключения корпоративной или локальной сети к Internet используется *proxy*-сервер, осуществляющий фильтрацию пакетов, передаваемых между Internet и локальной сетью (*intranet*). Основное назначение *proxy-сервера* — кэширование, т.е. сохранение наиболее часто запрашиваемых документов, что существенно снижает сетевой трафик от провайдера к локальной сети; также он может использоваться для установки настроек и ограничений на работу каждого пользователя Internet из локальной сети; отчасти *proxy-сервер* повышает безопасность локальных сетей, т.к. затрудняет несанкционированное проникновение в сеть извне.

Ценовая политика Internet-провайдеров предполагает установление базисных тарифов по различным тарифным планам и системы льгот и скидок с целью стимулирования спроса и привлечения новых клиентов. Основные виды тарифных планов: повременная оплата — оплачивается время работы в Internet (время использования канала), руб/час; оплата по трафику — оплачивается количество переданной и полученной информации, руб/Мбайт; абонентская плата — фиксированная плата за работу в Internet в течение определенного времени, руб/мес. Часто провайдером устанавливаются дополнительные ограничения на трафик. Существует также смешанная оплата — комбинирование нескольких различных форм оплаты, например, может учитываться время работы и трафик. Предоставление пользователю возможности работы в Internet предусматривает также регистрационный взнос.

7.2.4. Информационные и коммуникационные сервисы Internet

▶ *E-mail* (*Electronic mail*) — система для обмена электронными сообщениями (файлами), снабженными стандартным заголовком.

Для работы с *E-mail* необходимо зарегистрироваться то есть «завести почтовый ящик» на почтовом сервере провайдера или на одном из серверов, предоставляющих бесплатные почтовые услуги (*www.rambler.ru*, *www.mail.ru*, *www.yahoo.com* и др.). На компьютере пользователя устанавливается клиентская почтовая программа — *мэйлер* (*mail*, *почта*). При запуске эта программа устанавливает связь с почтовым сервером, а после прохождения авторизации и проверки учетной записи клиента осуществляется обмен сообщениями по протоколам *pop3* / *smtp* или *http*. Клиент может иметь несколько учетных записей, т.е. несколько *e-mail* адресов.

В функции *мэйлера* входят: подготовка и отправка сообщений (*New*, *Create*); ответ на входящие сообщения (*Reply To*); редактирование сообщений; автоматическая сортировка прочитанных и непрочитанных сообщений по разным критериям; пересылка сообщений (*Forward*); вложение и извлечение файлов (*Attach*, *Extract*); настройка начальных установок; блокировка нежелательных сообщений; использование адресной книги; автоматическое добавление подписи, визитной карточки и бланков; шифрование сообщений; добавление электронно-цифровой подписи.

Преимущества E-mail: удобство подготовки и работы с сообщениями; высокая скорость доставки сообщения от нескольких секунд до нескольких минут; низкая стоимость; надежность доставки.

Дополнительные возможности: работа с группами новостей и службами рассылок, системы подписки и отправки сообщений по определенной тематике, получение и отправка объявлений по определенной тематике, участие в телеконференциях (*Usenet*), работа со службами *Fax-mail,* возможность отправки по *E-mail* факсимильных сообщений.

Структура электронных сообщений:

- заголовок;
- текст;
- вложения;
- электронно-цифровая подпись.

Основные *поля заголовка*: *From* — адрес отправителя; *To* — адрес получателя; *CC* (*Copies*) — кому еще адресовано это письмо (копии); *Subj* (*Subject*) — тема сообщения; *Date* — дата и время отправки сообщения.

Служебные поля: *Message ID* — уникальный идентификатор сообщения; *Received* — отметка о прохождении письма через маршрутизатор (аналог почтового штемпеля); *Status* — прочитано/не прочитано, уровень значимости, секретность и др.

Структура электронного адреса: идентификатор абонента @ адрес почтового сервера (@-коммерческое *at*, означает «на, при»). Пример *E-mail* адреса: *user_2005@mail.ru* .

При отправлении по *E-mail* сведений конфиденциального характера необходимо обеспечение подлинности сообщений и безопасности их передачи. Для подтверждения авторства отправителя используется механизм цифровой подписи. Для шифрования сообщений и формирования цифровой подписи используются алгоритмы шифрования с синхронным и асинхронным ключом. Шифрование с синхронным ключом предполагает, что ключ известен как отправителю, так и получателю. При шифровании с асинхронным ключом используется личный ключ, который известен только одному человеку, и открытый общедоступный ключ. Отправляемые сообщения шифруются с помощью открытого ключа получателя, но дешифруются только с помощью его личного ключа.

Электронная почта и сервисы, функционирующие на ее базе (телеконференции, списки рассылок, доски объявлений и др.) представляют собой *off-line* коммуникационные сервисы.

Коммуникационные сервисы, работающие в режиме реального времени, — *on-line* сервисы, или сервисы и программы для прямого межпользовательского общения:

- *Talk*,
- *IRC* (*Internet Relay Chat*),

■ *Chat-rooms* (виртуальные гостиные),
■ *ICQ* (*I seek you*),
■ *Net Meeting* (обмен файлами, сообщениями в Intranet и Internet),
■ службы *Internet-Phone* (обмен голосовыми сообщениями через Internet).

Для передачи файлов с удаленного компьютера на локальный в Internet используется протокол передачи файлов *FTP* (*File Transfer Protocol*), с помощью которого можно обмениваться файлами с любым клиентом сети.

FTP-сервис используется:

- для загрузки файлов (архивов) с удаленного компьютера на свой (*download*),
- для загрузки собственных страниц на удаленный компьютер (*upload*).

Например, *FTP* часто используется для «закачивания» «домашних страничек» на серверы, предоставляющие место под них, и «выкачивания» файлов из *FTP*-архивов. Существуют тысячи серверов, предоставляющих гигабайты самой разнообразной информации: от фотографий из космоса, видео и музыкальных коллекций до дистрибутивов общедоступных программ. Файловые архивы в *FTP* чаще всего имеют формат *.zip*.

Любой *FTP*-сервер требует авторизации пользователя, то есть ввода его имени и пароля, в зависимости от которых пользователю будет предоставлен доступ к определенным каталогам и возможность осуществлять разрешенные действия над содержимым *FTP*-архива.

Структура *FTP*-запроса: *ftp://<user>:<password>@<host>/<url-path>*, где user — имя пользователя, password — его пароль, host — доменное имя или *IP*-адрес сервера, url-path — путь к файлу.

Практически каждый *FTP*-сервер предоставляет анонимный вход (другое название этого сервиса — анонимный *FTP*). Для анонимного *FTP* существует упрощенный синтаксис: *ftp://<host>/<url-path>*. Имя для входа — *anonymous*. При таком входе предоставляется доступ к общедоступным каталогам и данным. Обычно в таком режиме доступа к серверу пользователь может только просматривать каталоги и «выкачивать» файлы к себе на диск. Некоторые серверы создают специальные каталоги, куда каждый желающий также может «закачать» свои собственные файлы.

Когда пользователь открывает *FTP*-соединение, то оказывается в root-каталоге (самый верхний каталог в иерархии файлов).

Каталоги */bin*, */usr*, */etc*, */dev* содержат системные файлы. Пользователь может перемещаться по дереву каталогов файлового архива, просматривать список файлов и при желании «скачивать» необходимые файлы к себе на компьютер. Скачивание файлов осуществляется обычно из общедоступного каталога */pub FTP*-сервера. Работа в *FTP*-каталогах аналогична работе с любым файл-менеджером, например, *Windows Commander*, *FAR* или *Windows Explorer* (Проводник). Щелкая клавишей мыши на именах папок, можно переходить из каталога в каталог, а щелкая на именах файлов — копировать файл.

7.2.5. Поиск информации в Internet

Для поиска нужной информации в Internet можно воспользоваться специальными поисковыми машинами, например: *www.rambler.ru*, *www.yandex.ru*, *www.yahoo.com*, *www.altavista.com*, *www.google.com*, *www.alltheweb.com* и многими другими. Из-за большого количества информации в Internet на каждой поисковой машине заранее создается специальная база данных (база индексов), которая представляет собой список слов, встречающихся в web-документах, и их сетевые координаты (*URL*).

Пользователь составляет так называемый поисковый *тезаурус,* то есть набор ключевых слов для формирования поисковых запросов, по возможности наиболее точно характеризующий тематическую область поиска, и направляет запрос на поисковую машину. После обработки запроса и сопоставления с базой индексов, сервер выдает ранжированный список ссылок на документы, в которых встречаются слова, указанные в запросе пользователя. Таким образом, работа поисковых систем включает в себя:

- индексирование, которое осуществляется при помощи *спайдеров* (*spider, паук*) — программ, посещающих ресурсы Internet и полностью или частично индексирующих их содержание);
- занесение индексов в базу данных поисковой системы;
- сопоставление поискового запроса с индексами базы данных.

Основные характеристики поисковых систем и результатов поиска:

- *релевантность* — соответствие результатов поиска первоначальному поисковому запросу;
- *охват* — объем базы индексов поисковой системы; измеряется количеством уникальных серверов (хостов) и уникальных документов;

- *скорость обхода* — скорость формирования индексов;
- *скорость поиска* — скорость выполнения поисковой системой запроса пользователя;
- *точность поиска* — параметр, показывающий, какова доля релевантных документов в общем количестве найденных;
- *полнота поиска* — параметр, показывающий, какова доля найденных релевантных документов в общем количестве релевантных документов.

Каждая конкретная поисковая машина содержит на своем WWW-сервере подробное описание языка запросов поиска. Конечно, языки запросов поисковых машин отличаются друг от друга, но среди них есть и общие черты, например, знак плюса перед словом обозначает обязательный поиск данного слова, знак минуса обозначает исключение слова из поиска, фраза, взятая в кавычки, означает поиск её точного совпадения.

7.3. Моделирование компьютерных сетей средствам теории очередей

7.3.1. Общие сведения о моделировании информационных систем

Моделирование информационных систем (ИС), построенных, как правило, на основе локальных и/или глобальных сетей передачи данных, включает задачи синтеза и задачи анализа системы и ее частей. Задача *синтеза* состоит в том, чтобы из заданного множества элементов построить некоторую систему, обладающую заранее заданными свойствами. Задача *анализа* состоит в выделении отдельных элементов заданной системы и установлении их параметров, а также соответствия ИС заданным свойствам. Основными такими свойствами являются работоспособность и эффективность. *Работоспособность* ИС состоит в правильном выполнении заданных функций, то есть в правильной реализации заданного множества алгоритмов обработки информации. *Эффективность* ИС заключается в ограниченности или минимальности различного рода затрат, связанных с созданием и эксплуатацией ИС. Показателями эффективности могут служить, например, быстродействие устройств, вероятность получения ошибочного результата. Показатели, характеризующие затраты времени на получение системой ожидаемых результатов, называют показателями *производительности*.

В общем виде задача анализа производительности ИС состоит в том, чтобы оценить показатели производительности при заданных параметрах технического, программного обеспечения ИС и внешней среды. К таким параметрам могут относиться быстродействие устройств, уровень сложности программ, интенсивность потоков требований на выполнение программ и многие другие. Анализируя производительность ИС, необходимо учитывать случайную природу многих факторов, от которых она зависит. Чаще всего случайными являются моменты поступления в ИС требований, объемы подлежащей обработке информации. По мере совершенствования средств вычислительной техники усложняется и задача анализа производительности ИС. Поэтому все более широкое распространение получают методы математического моделирования ИС средствами теории очередей с использованием моделей в виде сетей массового обслуживания (СеМО). Это объясняется тем, что в терминах этой теории описываются многие реальные ИС и подсистемы, из которых они состоят: вычислительные системы и сети, узлы сетей связи, системы продажи билетов пассажирам и многие другие, в которых возможны очереди и отказы в обслуживании.

СеМО представляет собой совокупность конечного числа обслуживающих узлов (в свою очередь, являющихся системами массового обслуживания, СМО), в которой циркулируют заявки, переходящие в соответствие с маршрутом из одного узла сети в другой. СеМО используют для определения важнейших системных характеристик ИС: производительности; времени доставки пакетов; вероятности потери сообщений и блокировки в узлах; максимальной пропускной способности, допустимых пиковых значений нагрузки, при которых обеспечивается требуемое качество обслуживания.

В распределенной информационной вычислительной сети в качестве обслуживающих могут выступать устройства, осуществляющие доставку данных по компьютерной сети (маршрутизаторы, пакетные коммутаторы), а также серверы сетей. Заявками на обслуживание являются пакеты данных различного назначения, передаваемые по сети, запросы к базам данных, Internet-серверам, решаемые задачи. Буфером для хранения заявок является внутренняя память обслуживающих устройств.

В теории СеМО фундаментальным является понятие состояния сети. Для определения вероятностей состояния СеМО исследуют протекающий в ней случайный процесс. При моделировании процессов в СеМО наиболее часто исходят из предположения, что этапы обслуживания независимы между собой и не зависят ни от параметров входящего потока,

ни от состояния сети, ни от маршрутов следования требований. Поскольку входящие потоки требований и времена каждого этапа обслуживания описывают экспоненциальными законами распределения, такие СеМО называют экспоненциальными, а входные потоки в них — пуассоновскими.

Пуассоновский или простейший — это поток, удовлетворяющий требованиям:

• стационарности — когда интенсивность (число заявок в единицу времени) поступления заявок постоянна;

• ординарности — означающей, что вероятность поступления на элементарный промежуток времени двух и более заявок пренебрежимо мала по сравнению с вероятностью поступления одной заявки. Другими словами, требования на обслуживание поступают «по одному»;

• отсутствия последействия — вероятность поступления k требований в течение некоторого промежутка времени не зависит от того, сколько требований и как поступали до этого промежутка.

7.3.2. Пример практического применения аппарата сетей массового обслуживания для моделирования компьютерной сети

Постановка задачи

Необходимо обосновать выбор аппаратуры передачи данных (АПД) и системного блока сервера из ряда наиболее распространенных для комплектования системы удаленной обработки заданий. Система должна обеспечивать получение ответа на запрос за время, среднее значение которого не превышало бы заданное значение. Другим естественным ограничением будем считать обеспечение наименьшей стоимости реализации системы.

Задания поступают от отдельных рабочих станций на концентратор в виде потоков запросов (требований). Концентратор обеспечивает суммирование поступающих от рабочих станций потоков заданий на выборку данных из базы и распределение ответов с решениями по источникам запросов.

Примем допущение, что запросы и ответы с решениями имеют фиксированный объем, и передача запросов и ответов по каналу связи осуществляется без ошибок.

Исходные данные

– структура системы удаленной обработки заданий (рис. 7.3):

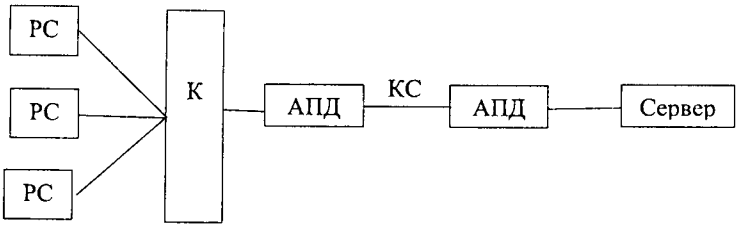


Рис. 7.3. Система удаленной обработки данных
РС – рабочие станции; К – концентратор; КС – канал связи

— суммарный поток требований (запросов), поступающих от терминалов через концентратор на вход АПД – пуассоновский с интенсивностью $л_{вх}$ запросов/с;
— объем запроса $V_з$ (байт);
— объем результата решения $V_р$ (байт);
— скорость АПД B (бит/с);
— структура сервера (рис. 7.4):

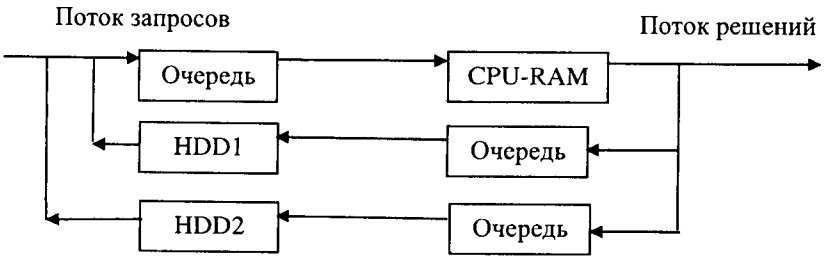


Рис. 7.4. Структура сервера
CPU-RAM – подсистема процессор – оперативная память; HDD1 – внешняя память для хранения данных; HDD2 – внешняя память для хранения задач

Характеристики решаемых задач:
– трудоемкость алгоритма (число машинных операций) Q_m, $m = 1,...,M$;
– число обращений за данными D_l, $l = 1,...,L$;
– вероятность прохождения k-го алгоритма P_k, $k = 1,...,K$;

– интенсивность обслуживания $м_1$ – для HDD1, $м_2$ – для HDD2 (операций чтения-записи /с);

– быстродействие процессора w_i, i=1,…,I (операций/с);

– скорость передачи данных на АПД V_j, j=1,…,J (бит/с);

– допустимое время получения ответа $T_{доп}$(с).

Математическая постановка задачи

Выбрать процессор из ряда w_i, i=1,…,I и АПД из ряда V_j, j=1,…,J так, чтобы время получения ответа с решением удовлетворяло условию

$$\tau_{отв}^{(i,j)} \le T_{доп}.$$

Методика выполнения задания

Время получения ответа складывается из времени задержки задания/ответа с решением на концентраторе $\tau_K^{(j)}$, времени передачи задания $\tau_{зпрд}^{(j)}$ и решения задания $\tau_{рпрд}^{(j)}$ и времени, затрачиваемого на решение задания на сервере $\tau_{серв}^{(i)}$:

$$\tau_{отв}^{(i,j)} = \tau_{зпрд}^{(j)} + \tau_{рпрд}^{(j)} + \tau_{серв}^{(i)} + \tau_K^{(j)},$$

где j – номинал скорости передачи АПД (бит/с),

i – номинал быстродействия подсистемы CPU-RAM (операций/с).

Поскольку АПД обеспечивает дуплексный канал (см. п. 7.1.3)

$$\tau_{зпрд}^{(j)} = \frac{V_з \times 8}{V_j};\ \tau_{рпрд}^{(j)} = \frac{V_р \times 8}{V_j}.$$

Задержка на концентраторе может возникнуть за счет ожидания передачи задания либо ответа с решением к источнику за счет занятости канала передачи данных. Среднее время обслуживания на концентраторе определяется средним временем занятости АПД, которое определим в виде

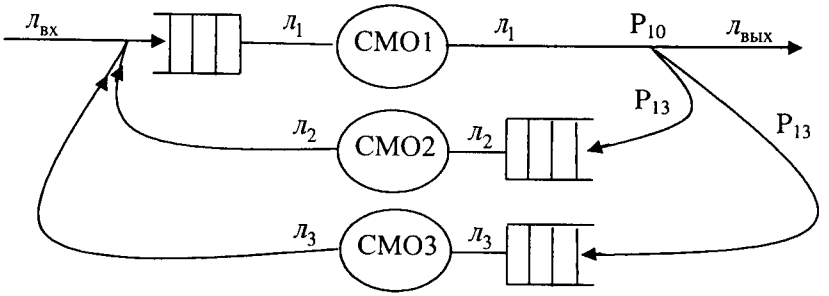
$$\overline{T_k} = \frac{(V_з + V_р)\times 8}{V_j},$$

и среднее время задержки на концентраторе определим как среднее время ожидания в очереди

$$\overline{\tau_k}^{(j)} = \frac{\overline{T}_k \cdot л_{вх}/м_j}{1 - л_{вх}/м_j}.$$

Для формализации расчета сервера используем схему разомкнутой экспоненциальной СеМО (рис. 7.5), которая задается следующими параметрами:

1) числом СМО N, N=3;



▲

Рис. 7.5. Модель сервера в виде СеМО
СМО1 – CPU-RAM; МО2 – HDD1; СМО3 – HDD2

2) числом каналов обслуживания в каждой СМО, q_i, i=1, 2, 3;

3) матрицей вероятностей передачи, или маршрутной матрицей $P = \|P_{ij}\|$, i,j=0, N; 0 – внешняя среда; P_{ij} – вероятность того, что заявка после обслуживания в узле i перейдет в узел j. Если узлы непосредственно не связаны между собой, то P_{ij} =0. Если из i-го узла может быть переход только в один какой-либо узел j, то P_{ij} =1;

4) интенсивностями $л_1, \ldots, л_n$ входных потоков, $л_1 = л_{вх}$;

5) средними временами обслуживания $T^1_{обс}, \ldots, T^N_{обс}$ заявок в СМО. $T^1_{обс} = T_{проц}$, $T_{проц}$ – время обслуживания отдельного требования процессором, зависит от производительности w_i. $T^2_{обс} = T_{HDD1} = 1/м_1$, $T^3_{обс} = T_{HDD2} = 1/м_2$.

1. Среднее время пребывания заявки в СеМО, представленной на рис. 7.3, рассчитывается по формуле $Ф_{серв} = \frac{1}{л}\sum_{j=1}^{N} л_j T^j_{преб}$, где $л = л_1 + л_2 + \ldots + л_n$, $л_1 = л_{вх}$, $T^j_{преб}$ – время пребывания заявки в j-ой СМО, j=1, 2, 3.

Необходимо найти интенсивности $л = л_1, л_2, л_3$ и $T^1_{преб}$, $T^2_{преб}$, $T^3_{преб}$.

2. Нахождение интенсивностей $л = л_1, л_2, л_3$ осуществляется на основе уравнений баланса сети с учетом свойств слияния и разветвления потоков. Слияние и разветвление задается матрицей P переходов.

2.1. Согласно рис. 7.5,

$$P = \begin{pmatrix} 0 & 1 & 0 & 0 \\ P_{10} & 0 & P_{12} & P_{13} \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}. \qquad (7.1)$$

2.2 Уравнение баланса. Для сети без потерь $л_{вх} = л_{вых}$, $л_{jвх} = л_{jвых}$, j=1, 2, 3.

Система уравнений для системы удаленной обработки заданий:

$$\begin{aligned} л_1 &= л_{вх} + л_2 + л_3 \\ л_2 &= л_1 \times P_{12} \\ л_3 &= л_1 \times P_{13} \\ л_{вх} &= л_{вых} = л_1 P_{10}. \end{aligned} \qquad (7.2)$$

2.3. Маршрут заявки в сети случаен, поэтому случайно и число обслуживаний заявки в j-ой сети. Среднее значение α_j этого числа обслуживаний называется передаточным коэффициентом. Решение системы:

$$\begin{aligned} л_1 &= л_{вх} \times \frac{1}{P_{10}} = б_1 л_{вх}; \\ л_2 &= л_1 \times P_{12} = л_{вх} \times \frac{P_{12}}{P_{10}} = б_2 л_{вх}; \\ л_3 &= л_1 \times P_{13} = л_{вх} \times \frac{P_{13}}{P_{10}} = б_3 л_{вх}. \end{aligned} \qquad (7.3)$$

2.4 Среднее число обращений к данным на HDD1

$$\overline{D} = \sum_{i=1}^{L} P_i D_i . \qquad (7.4)$$

2.5 Как видно из рис. 7.4, в процессе решения задача «проходит» через CPU-RAM $\left(\overline{D} + 2\right)$ раз: 1 раз идентифицируется и обращается к HDD2 (за программой); $\overline{D}$ раз прерывается для обращения за данными к HDD1; 1 раз завершается обработка задания, и готовое решение выходит из СМО1 во внешнюю среду.

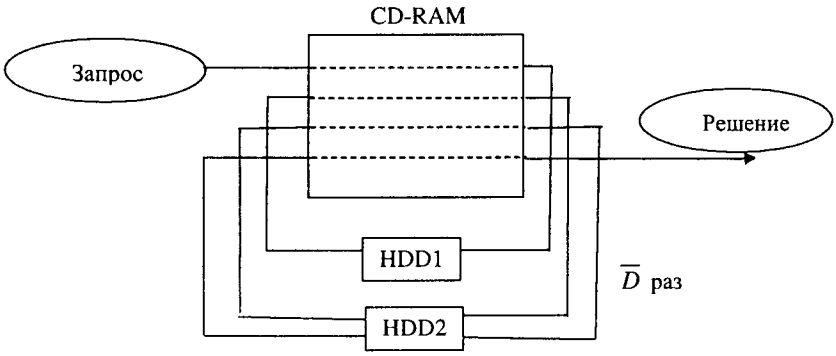


Рис. 7.6. Процесс решения задачи

Следовательно, переходные вероятности можно теперь определить в виде

$$P_{10} = \frac{1}{\overline{D}+2};\quad P_{12} = \frac{\overline{D}}{\overline{D}+2};\quad P_{13} = \frac{1}{\overline{D}+2}. \tag{7.5}$$

2.6 Интенсивность входных потоков отдельных СМО.

Согласно (7.3) и (7.5) имеем выражения для коэффициентов: $б_1 = \overline{D} + 2$; $б_2 = \overline{D}$; $б_3 = 1$. Соответственно,

$$л_1 = б_1 л_{вх} = \left(\overline{D}+2\right) л_{вх};\quad л_2 = б_2 л_{вх} = \overline{D} \times л_{вх};\quad л_3 = б_3 л_{вх}. \tag{7.6}$$

2.7 Тогда

$$\overline{ф}_{сри} = \frac{1}{л_{вх}} \sum_{i=1}^{3} л_i T_c^i \sum_{i=1}^{3} б_i T_c^{(i)}, \tag{7.7}$$

$T_c^{(i)}$ – время пребывания заявки в СМО.

3. Определение $T_c^{(i)}$

3.1 Для экспоненциальной СМО

$$T_c = \frac{1}{м(1-с)}, \tag{7.8}$$

$с = л/м$, $м$ — интенсивность обслуживания заявки, ρ — коэффициент загрузки.

3.2 $с = л \times \overline{T}_{обс}$. (7.9)

3.3 Определим время обслуживания отдельного запроса на CPU-RAM (СМО1).

3.3.1 Средняя трудоемкость решения задачи

$$\overline{Q} = \sum_{i=1}^{M} P_i Q_i. \tag{7.10}$$

3.3.2 Трудоемкость обслуживания отдельного обращения

$$\overline{Q}_1 = \overline{Q} \Big/ б_1 = \frac{\overline{Q}}{\overline{D}+2}. \tag{7.11}$$

3.3.3 Время обслуживания отдельного обращения при быстродействии процессора w_i, i=1,…,I

$$\overline{T}_{обс}^{(i)} = \overline{Q}_1 / w_i. \tag{7.12}$$

3.3.4 Рассчитаем коэффициенты загрузки.

$$с_1^{(i)} = л_1 \times \overline{T}_{обс}^{(i)} \tag{7.13}$$

$$с_2 = л_2 / м_2 \tag{7.14}$$

$$с_3 = л_3 / м_3 \tag{7.15}$$

3.3.5 Время пребывания отдельного обращения (требования) в соответствующих СМО

$$T_c^1 = \frac{1}{м_1\left(1-с_1\right)} = \frac{\overline{T}_{обс}^{(i)}}{1-с_i^1} = \frac{\overline{Q}_1}{w_1 - л_1 Q_1}; \quad (7.16)$$

$$T_c^2 = \frac{1}{м_{HDD1}\left(1-с_2\right)} = \frac{1}{м_2 - л_2}; \quad (7.17)$$

$$T_c^3 = \frac{1}{м_{HDD2}\left(1-с_3\right)} = \frac{1}{м_3 - л_3}. \quad (7.18)$$

Таким образом, определены передаточные коэффициенты $б_i$, i=1, 2, 3 и времена пребывания отдельных требований T_c^1, T_c^2, T_c^3 в соответствующих СМО. Подставим выражения этих величин в (7.7) и определим $\tau_{серв}$.

4. Время решения задачи на сервере

$$ф_{отв}^{(j,i)} = \sum_{j=1}^{3} б_j T_{прj}^{(i)} = \left(4 \times T_{пр}^{(i)} + T_{пр}^{(2)} + T_{пр}^{(3)}\right) =$$

$$= \left(\overline{D} + 2\right) \times \frac{\overline{Q}_1}{w_1 - л_1 Q_1} + \overline{D} \times \frac{1}{м_2 - л_2} + \frac{1}{м_3 - л_3}. \quad (7.19)$$

5. Общее время ответа складывается из времен передачи запроса и решения и времени получения решения на сервере.

$$ф_{отв}^{(j,i)} = \frac{8}{V_j}\left(V_з + V_p\right) + ф_{серв}^{(i)} + ф_к^{(i)} \quad (7.20)$$

6. Это общее время получения ответа на запрос должно удовлетворять сформулированной выше постановке задачи

$$ф_{отв}^{(j,i)} \le T_{доп} \quad (7.21)$$

и задавать допустимый вариант комплектования АПД и сервера.

Алгоритм расчета

1. Среднее время пребывания заявки на сервере
2. Нахождение интенсивностей входных потоков отдельных СМО $л_1$, $л_2$, $л_3$.

2.1. Матрица переходных вероятностей (формула 7.1).

2.2. Уравнение баланса (7.2).

2.3. Решение системы уравнений баланса (7.3).

2.4. Среднее число обращений к данным на HDD1 (7.4).

2.5. Вычисление переходных вероятностей (7.5).

2.6. Определение интенсивностей входных потоков отдельных СМО (7.6)

2.7. Выражение для определения времени обработки запроса на сервере (7.7).

3. Определение времени пребывания заявки в каждой СМО.

3.1. Время пребывания заявки в отдельной СМО (7.8).

3.2. Коэффициент загрузки (7.9).

3.3. Определение времени обслуживания отдельного запроса.

3.3.1. Средняя трудоемкость решения задачи (7.10).

3.3.2. Средняя трудоемкость обслуживания отдельного обращения (7.11)

3.3.3. Время обслуживания отдельного запроса (7.12).

3.3.4. Определение коэффициентов загрузки (7.13), (7.14), (7.15).

3.3.5. Определение времени пребывания отдельного обращения (требования) в соответствующей СМО (7.16), (7.17), (7.18).

4. Вычисление времени, затрачиваемого на получение решения на сервере по запросу (7.19).

5. Вычисление общего времени ответа на запрос (7.20).

6. Условие для комплектования вариантов АПД и процессоров в систему (7.21)

Разработка компьютерной программы по приведенному алгоритму не требует специализированного программного обеспечения и может быть реализована, например, на языке Паскаль или в среде VBA, описанных в главе 6.

Более подробно с теорией моделирования компьютерных сетей средствами теории очередей можно познакомиться в [27].

ГЛАВА 8

ИНФОРМАЦИОННАЯ БЕЗОПАСНОСТЬ И ЗАЩИТА ДАННЫХ

8.1. Задача обеспечения информационной безопасности в современных условиях

8.1.1. Информация как объект защиты

Современный уровень информационных технологий позволяет рассматривать информацию, как общественный ресурс развития, наравне с традиционными (сырье, электроэнергия и т.д.) ресурсами. За последнее десятилетие резко увеличился уровень производства и потребления обществом информационных продуктов и услуг. В нашей стране ускоренными темпами идет развитие систем телекоммуникаций, увеличиваются парки компьютерной техники, непрерывно растет число пользователей сотовой связи и сети Интернет.

Информация, являясь по своей физической природе нематериальным объектом, давно стала предметом торговли и обмена, а с появлением ряда правовых и регулирующих положений, приобретает черты материального объекта. Федеральным Законом «Об информации, информатизации и защите информации» определено, что информационные ресурсы, то есть отдельные документы или массивы документов, в том числе и в информационных системах, являясь объектом отношений физических, юридических лиц и государства, подлежат обязательному учету и защите, как всякое материальное имущество собственника. При этом собственнику предоставляется право самостоятельно в пределах своей компетенции устанавливать режим защиты информационных ресурсов и доступа к ним. Ответственность за выполнение мер защиты информации возлагается также и на пользователя информации. При этом следует особо подчеркнуть, что защищается только информация документированная, зафиксированная на материальном носителе информации с реквизитами, позволяющими её идентифицировать. Так, нематериаль-

ный объект, например, электронный текст, может обрести юридическую силу, если у него есть, кроме прочих реквизитов, заверенная электронная подпись и дата.

По степени важности и ценности защищаемой информации выделяется три категории:

- информация, составляющая государственную тайну;
- персональная информация;
- информация, составляющая коммерческую тайну.

Возможности доступа к первой категории информации, как и владение ей, определяется государством. Выбор степени защиты второй и третьей категорий информации возлагается на собственников информации, поэтому, например, право отнесения той или иной информации, циркулирующей на предприятии, к категории коммерческой тайны предоставлено руководителю данного предприятия. И, наконец, прежде чем определить комплекс мер по защите своих информационных ресурсов, важно напомнить основные цели защиты информации, согласно законодательным актам:

- предотвращение утечки, хищения, искажения, подделки информации;
- обеспечение безопасности личности, общества, государства;
- предотвращение несанкционированных действий по уничтожению, искажению, блокированию информации;
- защита конституционных прав граждан на сохранение личной тайны и конфиденциальности персональных данных;
- сохранение государственной тайны, конфиденциальности документированной информации.

Безопасность информационных систем и технологий обеспечивается реализацией следующих основных функций безопасности.

Защита конфиденциальности — достигается путем применения сертифицированных средств защиты информации от несанкционированного доступа, реализации правил разграничения доступа к информации, применения алгоритмов специального преобразования данных при передаче информации в сети передачи данных, проведения специальных работ по предотвращению утечки конфиденциальной информации по техническим каналам, а также организационных мер по предотвращению разглашения конфиденциальной информации и неправомерных действий со стороны лиц, имеющих право доступа к конфиденциальной информации.

Защита целостности — достигается путем разработки и внедрения технологий резервирования и восстановления ресурсов, применения тех-

нологий электронной цифровой подписи, физической охраной средств вычислительной техники и носителей информации, другими организационными мерами.

Защита доступности — достигается путем резервирования технических средств, дублирования каналов передачи данных в сети, применением активного анализа трафика сети, а также организационными мерами.

Защита живучести системы — достигается распределенным резервированием баз данных, реализацией технологий восстановления ресурсов, а также организационными мерами, направленными на поддержание механизма реагирования на потенциальные угрозы при возникновении чрезвычайных ситуаций.

И, наконец, любая система обеспечения информационной безопасности должна выполнять следующий ряд задач:

- своевременное выявление и устранение угроз информационным и техническим ресурсам;
- выявление причин и условий, способствующих нанесению материального ущерба организации, предприятию и их клиентам при нарушении функционирования системы и несанкционированного использования информации;
- создание механизма и условий оперативного реагирования на угрозы безопасности и проявления негативных тенденций в функционировании;
- эффективное пресечение посягательств на ресурсы и информацию, циркулирующую в системе, на основе комплексного подхода к обеспечению безопасности;
- контроль эффективности мер защиты.

8.1.2. Источники угроз и способы нарушения информационной безопасности

Порядок исследований в сфере нарушений информационной безопасности можно представить в виде трехэтапной схемы:

1) исследование действий, попадающих под понятие нарушение (преступление) в информационной сфере;

2) исследование объектов обмена информацией и опасностей, угроз нанесения ущербов субъектам информационных отношений;

3) исследование факторов, уязвимостей конкретных объектов информатизации и действий, атак, реализующихся через имеющиеся уязвимости.

Дадим некоторые определения.

▶ *Информационная безопасность* — состояние защищенности важных интересов личности, общества и государства от внутренних и внешних угроз.

Угроза (вообще) — потенциально возможное событие, действие (воздействие), процесс или явление, которое может привести к нанесению ущерба чьим-либо интересам.

▶ *Угроза интересам субъектов информационных отношений* — потенциально возможное событие, процесс или явление, которое посредством воздействия на информацию или другие компоненты информационных систем может прямо или косвенно привести к нанесению ущерба интересам данных субъектов.

▶ *Уязвимость* — причина нестабильности режима информационной безопасности, обусловленная просчетами и ошибками, допущенными в процессе проектирования и эксплуатации объекта информатизации.

▶ *Атака* — действия, проявляющиеся при реализации угроз информационной безопасности через имеющиеся уязвимости.

Результатами реализации угроз информационной безопасности и осуществления способов воздействия на информационные ресурсы, информационные системы и процессы в общем случае являются:

- нарушение секретности (конфиденциальности) информации (разглашение, утрата, хищение, утечка и перехват и т.д.);
- нарушение целостности информации (уничтожение, искажение, подделка и т.д.);
- нарушение доступности информации и работоспособности информационных систем (блокирование данных и информационных систем, разрушение элементов информационных систем, компрометация системы защиты информации и т.д.).

Одним из первых в области информационной безопасности задается следующий вопрос: какие действия попадают под понятие преступление в информационной сфере? Ответ на это вопрос дает международная классификация способов совершения компьютерных преступлений, которая ведется Интерполом в классификаторе информационно-поисковой системы, начиная с 1990-х годов.

Все коды, характеризующие компьютерные преступления, имеют идентификатор, начинающийся с буквы **Q**. Для характеристики преступления могут использоваться до пяти кодов, расположенных в порядке убывания значимости совершенного.

1. QA — Несанкционированный доступ и перехват

QAH — компьютерный абордаж (хакинг) — доступ в компьютер или сеть без права на то. Этот вид компьютерных преступлений обычно используется хакерами для проникновения в чужие информационные сети.

QAI — перехват при помощи технических средств без права на то, осуществляемый либо прямо через внешние коммуникационные каналы системы, либо путем непосредственного подключения к линиям периферийных устройств.

QAT — кража времени, с целью неуплаты за пользование компьютерной системой или сетью.

QAZ — прочие виды несанкционированного доступа и перехвата

2. QD — Изменение компьютерных данных

QUL/QDT — логическая бомба, троянский конь, действия, позволяющие осуществлять функции, не планировавшиеся владельцем или пользователем программы.

QDV — компьютерный вирус, вредоносная самопорождающаяся программа, порождающая нежелательные программные действия и/или приводящая к утрате и искажению хранимых данных.

QDW — компьютерный червь, искажение или хищение данных посредством компьютерной сети.

QDZ — прочие виды изменения данных, взлом (крэкинг).

3. QF — Компьютерное мошенничество

QFC — мошенничество с банкоматами.

QFF — компьютерная подделка.

QFG — мошенничество с игровыми автоматами.

QFM — манипуляции с программами ввода-вывода.

QFP — мошенничества с платежными средствами.

QFT — телефонное мошенничество.

QFZ — прочие компьютерные мошенничества.

4. QR — Незаконное копирование

QRG — компьютерные игры.

QRS — прочее программное обеспечение.

QRT — топография полупроводниковых изделий.

QRZ — прочее незаконное копирование.

5. QS — Компьютерный саботаж

QSH — с аппаратным обеспечением.

QSS — с программным обеспечением.

QSZ — прочие виды саботажа.

6. QZ — Прочие компьютерные преступления

QZB — с использованием компьютерных досок объявлений.

QZE — хищение информации, составляющей коммерческую тайну.

QZS — передача информации конфиденциального характера.

QZZ — прочие компьютерные преступления.

Из приведенной классификации видно, что наибольшее значение имеют преступления, связанные с несанкционированным доступом и неправомерным завладением информации.

Однако, в нашей стране, в отличие от приведенного классификатора, наиболее распространенные (практически повсеместно совершающиеся), преступления имеют иную цель — пользование нужными свойствами неправомерно полученной информации (программы, базы данных), как то: выполнения расчетов с использованием программы, получения справок или отчетов из базы данных и связанное с этим незаконное копирование программ. Вторым по распространенности злом следует признать разрушение информации, связанное с ущербом, наносимым вирусными программами, а затем идут действия по модификации информационных продуктов как товара (лишение их свойств защищенности), т.е. различные виды взлома.

Перейдем к рассмотрению следующего вопроса: каковы угрозы, присущие объектам информатизации?

Современные объекты информационных систем состоят из следующих подсистем:

- рабочих станций или удаленных терминалов сети;
- серверов (служб Интернета, файлов, печати, баз данных и т.п.);
- межсетевых телекоммуникационных средств (коммутаторов, маршрутизаторов, серверов удаленного доступа и т.д.);
- каналов связи (проводных, беспроводных, оптических).

В связи с этим существует значительное число угроз информационной безопасности и принципов их классификации. Однако, очевидно, что в любой классификации необходимо рассмотреть стороны и мотивы происхождения угроз. По отношению к рассматриваемым информационным системам и сетям угрозы делятся на внутренние и внешние.

Источниками внутренних угроз являются:

- лица, прямо или косвенно участвующие в процессах информационных отношений своих организаций (основные сотрудники организаций, другой технический и вспомогательный персонал и др.);
- аппаратные средства и программное обеспечение информационных систем (как объекты, подверженные дефектам, сбоям, отказам и авариям).

Источниками внешних угроз являются:

- лица и организации, прямо заинтересованные или косвенно вовлекаемые в процессы информационных отношений других организаций (преступники, недобросовестные партнеры, поставщики телекоммуникационных услуг и др.);

- сети инженерных, транспортных и телематических коммуникаций.
- природные явления и стихийные бедствия.

Все перечисленные угрозы по мотивации действий делятся на:

- непреднамеренные (неумышленные, случайные) угрозы;
- преднамеренные (умышленные, спланированные) угрозы.

По способам воздействия на объекты информационных отношений угрозы делятся на:

- информационные (несанкционированный доступ к информационным ресурсам; незаконное копирование данных в информационных системах; хищение информации из библиотек, архивов, банков и баз данных; нарушение технологии обработки информации; противозаконный сбор и использование информации; использование информационного оружия);
- программные (дефекты и уязвимости в программном обеспечении; компьютерные вирусы; программные закладки);
- физические (разрушение компьютерных средств и сетей; хищение носителей информации, ключей защиты; воздействие на лиц, имеющих доступ к информации);
- радиоэлектронные (внедрение средств съема информации в компьютерные средства, сети и помещения; перехват, расшифровка, подмена и уничтожение информации в каналах связи);
- организационно-правовые (навязывание или закупки не сертифицированных информационных средств или систем; нарушение требований законодательства в отношении режимов информационной безопасности).

Дальнейшая детализация классификационной схемы угроз весьма объемна, поэтому приведем наиболее распространенные, на наш взгляд, типовые угрозы информационной безопасности.

Основные непреднамеренные угрозы:

1) нелегальная установка и использование неучтенных программ (игровых, обучающих, технологических и др., не являющихся необходимыми для выполнения служебных обязанностей);

2) заражение компьютера вирусами;

3) разглашение, передача или утрата атрибутов разграничения доступа (паролей, ключей шифрования, идентификационных карточек, пропусков и т.п.);

4) неправомерное включение оборудования или изменение режимов работы устройств и программ;

5) неумышленная порча носителей информации;

6) игнорирование режимов соблюдения информационной безопасности;

7) вход в систему в обход средств защиты (загрузка посторонней операционной системы со сменных магнитных носителей и т.п.);

8) некомпетентное использование, настройка или неправомерное отключение средств защиты администраторами сетей и персоналом службы безопасности;

9) пересылка данных по ошибочному адресу абонента (устройства);

10) ввод ошибочных данных.

Основные преднамеренные угрозы:

1) несанкционированное копирование носителей информации;

2) хищение носителей информации (магнитных дисков, лент, микросхем памяти, запоминающих устройств и целых ПК);

3) незаконное получение паролей и других реквизитов разграничения доступа с последующей маскировкой под зарегистрированного пользователя;

4) внедрение аппаратных и программных закладок и вирусов («троянских коней» и «жучков»), позволяющих преодолеть систему защиты, скрытно и незаконно осуществлять доступ к системным ресурсам с целью регистрации и передачи критической информации или дезорганизации функционирования системы;

5) перехват данных, передаваемых по каналам связи, и их анализ с целью выяснения протоколов обмена, правил вхождения в связь и авторизации пользователя и последующих попыток их имитации для проникновения в систему;

6) незаконное подключение к линиям связи с целью прямой подмены законного пользователя путем его физического отключения после входа в систему и успешной аутентификации с последующим вводом дезинформации и навязыванием ложных сообщений;

7) несанкционированное использование терминалов пользователей, имеющих уникальные физические характеристики, такие как номер рабочей станции в сети, физический адрес, адрес в системе связи, аппаратный блок кодирования и т.п.;

8) внедрение агентов в число персонала системы (в том числе, возможно, и в административную группу, отвечающую за безопасность) или вербовка (путем подкупа, шантажа и т.п.) персонала и отдельных пользователей, имеющих определенные полномочия;

9) вскрытие шифров криптозащиты информации;

10) физическое разрушение системы (путем взрыва, поджога и т.п.) или вывод из строя всех или отдельных наиболее важных компонентов компьютерной системы (устройств, носителей важной системной информации, лиц из числа персонала и т.п.).

Ввиду неуклонного роста числа пользователей сети Интернет особую значимость приобретают угрозы, связанные с удаленным сетевым доступом — сетевые компьютерные атаки. Существует ряд организаций (*SANS Institute*, *CERT Coordination Center* и др.), которые занимаются экспертизой уязвимостей сетевых компьютерных средств, программного обеспечения, сообщают о найденных уязвимостях и публикуют предлагаемые меры по их устранению. Приведем распространенные службы сетевого программного обеспечения, представляющие наибольшую уязвимость в отношении сетевых компьютерных атак.

На платформе Windows

1. Служба web-сервиса Интернет *Internet Information Services* (*IIS*).
2. Компоненты доступа к данным *Microsoft Data Access Components* (*MDAC*).
3. Сервер баз данных *SQL Server*.
4. Сетевой протокол *NetBIOS*, разделяемые сетевые ресурсы.
5. Анонимный вход в операционную систему («*null*»-сессии).
6. Менеджер аутентификации локальной сети (*LANMAN hash*).
7. Аутентификация входа в Windows.
8. Броузер Интернет *Internet Explorer* (*IE*).
9. Удаленный доступ к реестру (*Remote Registry Access*).
10. Windows-сценарии (скрипты).

На платформе UNIX:

1. Удаленный вызов процедур (*RPC*).
2. Веб-сервер *Apache*.
3. *SSH* (удаленный терминал с шифрованием трафика).
4. Протокол управления сетью *SNMP*.
5. Протокол передачи файлов *FTP*.
6. *RSH*-сервисы (возможность входа на доверенные системы).
7. Служба печати (*LPD — Line printer daemon*).
8. Почтовый сервер *Sendmail*.
9. Службы разрешения доменных имен *BIND/DNS*.
10. Аутентификация входа в систему.

8.2. Аспекты практической компьютерной безопасности

8.2.1. Средства анализа защищенности систем и сетей

Первой и важнейшей задачей обеспечения информационной безопасности является анализ состояния защищенности компьютерных систем и сетей. Выполнение анализа защищенности программного обеспечения систем и сетей достигается применением дополнительных инструментальных средств. К их числу относятся:

- средства анализа защищенности операционных систем (сканеры уязвимостей);
- средства анализа защищенности сетевых служб (сетевые сканеры).

Средства анализа защищенности операционных систем позволяют осуществлять ревизию механизмов разграничения доступа, идентификации и аутентификации, средств мониторинга, аудита и других компонентов операционных систем. Кроме этого, средствами данного класса проводится контроль целостности и неизменности программных средств и системных установок и проверка наличия уязвимостей системных и прикладных служб. Как правило, такие проверки проводятся с использованием базы данных уязвимостей операционных систем и сервисных служб, которые могут обновляться по мере выявления новых уязвимостей.

Средства анализа защищенности сетевых сервисов применяются для оценки защищенности компьютерных сетей по отношению к внутренним и внешним атакам. По результатам анализа защищенности сетевых сервисов средствами генерируются отчеты, включающие в себя список обнаруженных уязвимостей, описание возможных угроз и рекомендации по их устранению. Поиск уязвимостей основывается на использовании базы данных, которая содержит широко известные уязвимости сетевых сервисных программ и может обновляться путем добавления новых.

В настоящее время существует значительное число сканеров угроз безопасности, как коммерческих, так и некоммерческих для любой аппаратной платформы. Рассмотрим некоторые из них.

Средство сетевого анализа безопасности *Cisco Secure Scanner*

Cisco Secure Scanner (производитель *Cisco Systems*) позволяет проводить активный аудит ресурсов сети на наличие ошибок в области обеспечения защиты от несанкционированного доступа и на соответствие сетевых ресурсов политике безопасности. *Cisco Secure Scanner* предоставляет следующие возможности по анализу сетевой безопасности: инвентаризация систем и сервисов в корпоративной сети; определение воз-

можных ошибок в системах безопасности путем сканирования и протоколирования; эффективное управление уязвимостью данных; принятие решений в области защиты данных на основе отчетов и диаграмм; определение и реализация политики безопасности для новых устройств сети. Другими возможностями сканера являются: автоматическое определение узлов и сервисов, предоставление пользователю детальной информации обо всех устройствах, подключенных к наблюдаемой сети, построение карты доступных сервисов сети, используя порты *TCP/UDP* и запросы *SNMP*.

Средства активного аудита безопасности *Internet Security Systems*

Технология *Internet Security Systems* (производитель *ISS*) включает:

- технологию анализа защищенности или поиска уязвимостей;
- технологию обнаружения атак.

Реализует эту технологию семейство продуктов *SAFEsuite*, которое включает в себя:

1. Систему анализа защищенности на уровне сети *Internet Scanner*.
2. Системы анализа защищенности на уровне операционной системы и прикладного программного обеспечения *System Scanner* и *Online Scanner*.
3. Систему анализа защищенности на уровне систем управления базами данных *Database Scanner*.
4. Системы обнаружения атак на уровнях сети, операционных систем, баз данных и прикладного программного обеспечения *RealSecure*.

Программный комплекс для распределенного анализа защищенности сетей *Distributed CyberCop Scanner*

Distributed CyberCop Scanner (производитель *Network Associates*) позволяет выполнять сканирования на географически распределенных сетях, накапливая результаты в общей базе данных. В дополнение к большому количеству проверок конфигураций межсетевых экранов *Distributed CyberCop Scanner* тестирует web-сервера, файловые сервера, рабочие станции, маршрутизаторы и другие сетевые устройства на предмет обнаружения слабых мест в защите с выдачей подробных графических отчетов и рекомендаций по усилению безопасности. Графические отчеты значительно упрощают процесс определения приоритетов, визуально отображая уровень риска обнаруженных уязвимостей и предоставляя возможность администраторам безопасности быстро определить наиболее слабые места сети. Пользователи имеют возможность осуществлять поиск уязвимостей по уникальному номеру *CVE* (*Common Vulnerabilities and Exposures*), первому публичному списку, обеспечи-

вающему стандартизированные имена и описания более чем 500 известных уязвимостей.

Среди некоммерческих средств анализа защищенности можно выделить программы *Nessus* и *Nmap*, относящиеся к классу сетевых сканеров безопасности. Они используют много различных методов сканирования, основанных на особенностях протоколов стека *TCP/IP*. Другими возможностями этих программ являются: определение операционной системы удаленного компьютера; «невидимое» сканирование; динамическое вычисление времени задержки и повтор передачи пакетов; параллельное сканирование; определение неактивных узлов сети методом параллельного опроса; сканирование с использованием ложных узлов сети; определение наличия пакетных фильтров; сканирование с использованием *IP*-фрагментации, а также произвольное указание *IP*-адресов и номеров портов сканируемых сетей.

8.2.2. Защита компьютеров и сетей

Антивирусное программное обеспечение

К основным видам антивирусного программного обеспечения относятся: антивирусные сканеры, антивирусные мониторы, ревизоры изменений, и эвристические анализаторы. Некоторые из них практически вышли из употребления в связи с низкой эффективностью, и, наоборот, другие еще не используются достаточно широко.

Первые из упомянутых, *антивирусные сканеры* производят поиск в файлах, памяти, и загрузочных секторах вирусных масок, т.е. уникального программного кода вируса. Вирусные маски, описания известных вирусов содержатся в антивирусной базе данных, и если сканер встречает программный код, совпадающий с одним из этих описаний, то он выдает сообщение об обнаружении соответствующего вируса.

Антивирусные мониторы, являясь резидентной разновидностью антивирусных сканеров («резидентный» означает «постоянно находящийся в памяти»), позволяют осуществлять проверку файлов или сообщений электронной почты в реальном масштабе времени.

Принцип работы *ревизоров изменений* основан на снятии оригинальных «отпечатков» (например, контрольных *CRC* сумм) с файлов, системных секторов и системного реестра. Эти «отпечатки» сохраняются в базе данных, а при следующем запуске ревизор сверяет «отпечатки» с их оригиналами и сообщает пользователю о произошедших изменениях, отдельно выделяя вирусоподобные и другие, не подозрительные, изменения.

Эвристический анализатор представляет собой программу, отслеживающую различные события и в случае «подозрительных» действий (действий, которые может производить вирус или другая вредоносная программа), запрещает это действие или запрашивает разрешение у пользователя. Иными словами, эвристический анализатор совершает не поиск уникального программного кода вируса (как это делают сканеры и мониторы), не сравнивает файлы с их оригиналами (наподобие ревизоров изменений), а отслеживает и нейтрализует вредоносные программы по их характерным действиям.

Среди антивирусного программного обеспечения явно выделяются несколько лидеров: *McAfee VirusScan*, *Symantec Norton Antivirus*, *AVP* Лаборатории Касперского.

McAfee VirusScan обеспечивает антивирусную защиту компьютера, включая обнаружение вирусов при работе с файлами и групповыми приложениями, блокирование враждебных объектов *Java/ActiveX*, зараженных сообщений электронной почты и опасных Интернет-ресурсов. Он способен обезвреживать даже некоторые удаленные сетевые атаки, например, типа *Distributed Denial of Service (DDoS)*.

В *Symantec Norton Antivirus* включены новые технологии распознавания вирусных атак:

- технология *Striker32*, которая обнаруживает вирусы и восстанавливает данные, поврежденные наиболее сложными полиморфными вирусами, что часто невозможно сделать с помощью других антивирусных средств;
- эвристическая технология *Bloodhound*, которая отыскивает новые и неизвестные макровирусы (макросы, внедряющиеся в документы), используя передовую эвристическую технологию, позволяющую автоматически пропускать «чистые» файлы и задерживать инфицированные прежде, чем они смогут причинить вред.

В антивирусный пакет Лаборатории Касперского *AVP* входят все передовые способы борьбы с вирусами: антивирусный сканер для проверки мест хранения данных; антивирусный монитор для проверки всех используемых файлов в масштабе реального времени; ревизор изменений для контроля целостности данных на компьютерах; уникальный модуль фонового перехвата скрипт-вирусов; поведенческий блокиратор, обеспечивающий 100 % защиту от макровирусов.

Средства защиты сетевых операционных систем

Средства защиты операционных систем реализуют все современные принципы информационной безопасности. Рассмотрим их на примере серверной сетевой операционной системы Microsoft Windows 2003 Server.

В системах на базе этой операционной системы возможна идентификация пользователей сети и управление их доступом к ресурсам. В модели безопасности этой операционной системы используются доверительная проверка подлинности контроллерами доменов, делегирование полномочий между службами, а также объектно-ориентированное управление доступом.

В состав операционной системы Windows 2003 Server входят следующие службы обеспечения безопасности:

1. *Групповые политики безопасности.* Для управления рабочими средами пользователей и обеспечения безопасности сетевых ресурсов Windows 2003 Server поддерживает групповые политики. Они хранятся в доменах Windows 2003 Server (*Active Directory*) или в локальных базах данных на компьютерах.

2. *Инфраструктура открытого ключа.* Инфраструктура открытого ключа (*PKI — Public Key Infrastructure*) используется для проверки и подтверждения подлинности компьютеров и пользователей, вовлеченных в сетевое взаимодействие. Windows 2003 Server использует для обеспечения работы *PKI* службы сертификации (*Certification Services*).

3. *Шифрующая файловая система.* Windows 2003 Server поддерживает шифрованную файловую систему (*EFS — Encrypting File System*) для обеспечения безопасности данных на локальных жестких дисках. *EFS* зашифровывает файлы и папки, так что неавторизированный пользователь не сможет прочесть данные файлы. В корпоративных средах *EFS* обычно используется на исполнительских переносных компьютерах для защиты бизнес-информации в случаях, если компьютер будет физически украден.

4. *Протокол Kerberos v5.* Windows 2003 Server проверяет его подлинность пользователя, входящего в сеть при помощи протокола *Kerberos*, прежде чем позволить пользователю получить доступ к ресурсам сети. Kerberos является стандартизованным протоколом аутентификации.

5. *Протокол IPSecurity.* Для того чтобы обеспечить взаимодействие между двумя компьютерами в сети, Windows 2003 Server поддерживает протокол *Internet Protocol Security* (*IPSec*). *IPSec* шифрует данные на передающем компьютере и обеспечивает механизм дешифрования данных только на принимающем компьютере для обеспечения целостности данных.

6. *Делегирование администрирования безопасности.* Для управления сетями Windows 2003 Server системные администраторы могут разрешать пользователям определенных сетевых ресурсов выполнение специальных операций безопасности. Например, право осуществлять изменение паролей для отдела учета может быть делегировано определенному пользователю этого отдела компании.

Эффективное функционирование ни одной многопользовательской операционной системы невозможно без четкого разграничения доступа к ресурсам. Групповые политики безопасности являются одним из средств, позволяющих настраивать параметры безопасной работы пользователей в сети в операционных системах Windows 2003 Server. Политики являются правилами, которые определяют опции для настройки установок безопасности файлов, установки программного обеспечения, сценариев, запуска и выключения компьютеров, входа и выхода пользователя из системы, установок регистра и перенаправления папок.

Инфраструктура открытого ключа в Windows 2003 Server создает архитектуру для сетевой безопасности, которая закладывает фундамент для текущих и будущих потребностей бизнеса и предоставляют платформу для безопасных сетевых транзакций, подтверждаемых с помощью сертификатов. Математическим аппаратом этого информационного обмена служит криптосистема с открытым ключом. Сертификат — это средство, позволяющее гарантированно установить связь между переданным открытым ключом и передавшей его стороной, владеющей соответствующим личным ключом. Он представляет собой набор данных, зашифрованных с помощью электронной подписи, по стандарту является *ITU-T X*.509. Информация сертификата подтверждает истинность открытого ключа и владельца соответствующего личного ключа. Сертификаты также используются для обеспечения гарантии того, что содержание послания не будет подделано за время прохождения его от автора до того момента, когда его прочтет потребитель. Участниками обмена открытыми ключами являются центры сертификации («электронные нотариусы»), которым доверяют создавать, заверять и отзывать сертификаты; центры регистрации, уполномоченные выполнять идентификацию и регистрацию пользователей сертификатов и, наконец, сами пользователи, которые могут передавать запросы на получение сертификатов через web-интерфейс.

Поддержка прикладных программ шифрования информации с открытым ключом осуществляется с помощью интерфейса прикладного программирования *CryptoAPI*. Он содержит ряд функций, позволяющих приложениям шифровать данные и ставить цифровую подпись различными способами, обеспечивая защиту личных ключей. Независимые модули — *CSP* (*Cryptographic Service Provider* — поставщик службы шифрования) — фактически выполняют все процедуры шифрования. В том числе шифрующая файловая система обрабатывает файлы, используя открытый ключ и алгоритм симметричного шифрования *Data Encryption Standard X* (*DESX*) с длиной ключа 56 бит, функции которого доступны с помощью интерфейса *CryptoAPI*.

IP Security представляет собой два протокола: *Authentication Header* (*AH*) и *Encapsulated Security Payload* (*ESP*). Протокол *AH* обеспечивает взаимную аутентификацию узлов сети и обеспечивает целостность передаваемых пакетов. Протокол *ESP* выполняет шифрование содержимого пакетов, и также гарантирует взаимную аутентификацию и целостность. Когда защищенные данные аутентифицированы, получатель знает, что сообщение поступило именно от того отправителя, с которым установлено защищенное соединение, и во время передачи не подвергалось изменениям.

Защита компьютерных сетей

При информационном обмене между организациями часто возникает задача безопасного информационного взаимодействия локальных сетей и отдельных компьютеров через открытые сети, например, через сеть Интернет. При её решении требуется обеспечивать:

1) защиту подключенных к открытой сети Интернет локальных сетей организаций и отдельных компьютеров от несанкционированных действий со стороны внешней среды;

2) защиту информации в процессе передачи по общедоступным каналам связи.

Решение первой задачи основано на использовании межсетевых экранов (*brandmauer*, *firewall*). *Межсетевые экраны* — программные или программно-аппаратные средства, предназначенные для защищенного подключения корпоративной сети к сетям общего пользования на базе протоколов *TCP/IP*, *X*.25; разграничения ресурсов внутри корпоративной сети; а также для обеспечения контроля информационных потоков между различными сегментами корпоративной сети и внешними сетями. Они поддерживают безопасность информационного взаимодействия путем фильтрации двустороннего потока сообщений, а также выполнения функций посредничества при обмене информацией. Для защиты локальных сетей межсетевой экран располагают на стыке между локальной сетью и Интернет. Для защиты отдельного удаленного компьютера, подключенного к Интернету, программное обеспечение межсетевого экрана устанавливается на этом же компьютере, а сам он в этом случае называется персональным.

Большинство компонент межсетевых экранов относятся либо к фильтрующим пакетным брандмауэрам, либо к брандмауэрам уровня приложений (прокси-серверам). Первый из них выполняет отсев нежелательных сетевых пакетов на основании полей *IP*-дейтаграмм (адресов и портов отправителей и получателей). Второй выполняет регистрации собы-

тий и аудита каждого из приложений, пользующихся сетевыми сервисами, выполняя также преобразование сетевых адресов. Программных коммерческих и некоммерческих реализаций межсетевых экранов насчитывается довольно много. Хорошим решением является применение для приложений на платформе Windows 2003 Server корпоративного брандмауэра *Microsoft Internet Security and Acceleration Server*, а для персональных применений *Symantec Personal Firewall.*

Виртуальные частные сети

Для защиты информации в процессе передачи её по общедоступным каналам связи разработана концепция виртуальных частных сетей (*VPN*, *Virtual Private Network*). Виртуальная сеть формируется на основе каналов связи открытой сети. Сам термин «виртуальная» подчеркивает, что каналы связи виртуальной сети моделируются с помощью каналов связи реальной. Открытая сеть, например, Интернет, может служить основой для одновременного сосуществования множества виртуальных сетей, количество которых определяется пропускной способностью каналов доступа к Интернету. Защита *VPN* может строиться на уровне сокетов (*сетевой сокет* –совокупность *IP*-адреса и номера порта сетевого приложения, определяющего тип приложения), на основе протоколов *SSL* (*Securiry Socket Low*) и *SSH* (*Securiry Shell*), на транспортном (*TCP*) уровне и на сетевом (*IP*) уровне. Для решения задач *VPN* защита на последнем уровне наиболее предпочтительна ввиду следующих условий:

- модуль системы защиты реализуется в виде драйвера операционной системы, располагающегося между *IP*-стеком и драйвером сетевого адаптера, что позволяет контролировать весь проходящий трафик;
- защита работает на уровне драйверов, что затрудняет атаку на средство защиты, используя сервисы операционной системы;
- защита прозрачна для сервисов более высокого уровня и пользовательских приложений;
- защита организуется между любыми двумя точками сети, используя топологию сети;
- обеспечивается маскирование внутренней топологии сети.

Защита информации в *VPN* строится с использованием следующих технических приемов:

1. Шифрование исходного *IP*-пакета, что обеспечивает секретность содержащихся в пакете данных, таких как поля *IP*-заголовка и поле данных.

2. Цифровая подпись *IP*-пакетов, что обеспечивает аутентификацию пакета и источника-отправителя пакета.

3. Инкапсуляция *IP*-пакета в новый защищенный *IP*-пакет с новым заголовком, содержащим *IP*-адрес устройства защиты, что маскирует топологию внутренней сети.

Защита информации при передаче между виртуальными подсетями реализуется на алгоритмах асимметричных ключей и электронной подписи, защищающей информацию от подделки. Фактически данные, подлежащие межсегментной передаче, кодируются на выходе из одной сети, и декодируются на входе другой сети, а алгоритм управления ключами обеспечивает их защищенное распределение между оконечными устройствами. Все манипуляции с данными прозрачны для работающих в сети приложений. При этом возможно организовать защиту информации на любом уровне: защита трафика, то есть всей информации, передаваемой по каналу связи, например, между географически удаленными филиалами компании; между сервером и пользователем; между клиентами; организовать защищенный доступ мобильных пользователей в локальную сеть организации.

Защита телекоммуникационного оборудования

Важным аспектом защиты живучести компьютерных сетей является защита телекоммуникационного оборудования. В этой области практически все решения по обеспечению информационной безопасности ведутся на аппаратном уровне. Такой подход естественен и обусловлен автономностью функционирования современных средств телекоммуникации, которые по своей сложности не уступают, а иногда даже превосходят персональные компьютеры. Основные средства поддержки сетей телекоммуникаций — коммутаторы и маршрутизаторы — обладают высокими аппаратно-программными характеристиками, имея в своем составе несколько процессоров, работающих под управлением специализированной операционной системы реального времени и нескольких высокоскоростных сетевых интерфейсов. Они являются также аппаратной основой реализации упомянутых выше *VPN* сетей. С другой стороны, телекоммуникационное оборудование ответственно за качество обслуживания клиентов сетей передачи данных, которые требуют все большей пропускной способности сети для обмена аудио- и видео- информацией в реальном масштабе времени.

Встроенные системы безопасности современных пакетных коммутаторов и маршрутизаторов включают:

- средства защиты от широковещательных штормов в сетях;
- защиту данных на втором (канальном) и третьем (сетевом) уровнях модели *OSI*;

- многоуровневые средства безопасности при доступе через консольный порт, не позволяющие неавторизованным пользователям изменять конфигурацию устройства.

Встроенное программное обеспечение позволяет анализировать трафик на одном порту, в группе портов или во всем телекоммуникационном устройстве. Особо выделяются аппаратно-программные межсетевые экраны, обладающие, по сравнению с чисто программными реализациями, множеством достоинств. Например, продукт *Cisco PIX Firewall* (производитель *Cisco Systems*) обладает возможностями аппаратного аудита сетевых пакетов и номеров сетевых портов; аппаратного шифрования трафика; программно-аппаратной фильтрацией активных *Java* скриптов и *ActiveX*; аппаратной трансляцией сетевых адресов.

Защита речевых сообщений в сотовой связи

Сотовые системы мобильной связи нового поколения требуют обеспечения двух из основных функций безопасности: конфиденциальности и аутентификации. Конфиденциальность должна исключить возможность извлечения информации из каналов связи кем-либо, кроме санкционированного получателя. Функция аутентификации заключается в том, чтобы помешать кому-либо, кроме санкционированного пользователя (отправителя), изменить канал, то есть получатель должен быть уверен, что в настоящий момент он принимает сообщение от санкционированного пользователя. Основным способом обеспечения конфиденциальности является шифрование, и, относительно недавно, оно стало использоваться для задачи аутентификации. Аутентификация сообщений через шифрование осуществляется за счет включения в текст кода идентификации, то есть фиксированного, или зависящего от передаваемых данных слова, которое знают отправитель и получатель, или которое они могут выделить в процессе передачи. Получатель расшифровывает сообщение, путем сравнения получает удостоверение, что принимаемые данные являются именно данными санкционированного отправителя.

К системе шифрования речевых сообщений в сотовой связи предъявляются следующие основные требования:

- нелинейные связи между исходным текстом и зашифрованным текстом;
- изменение параметров шифрования во времени.

В сетях сотовой связи *GSM* механизмы аутентификации осуществляются с помощью переносимого мобильным пользователем модулем под-

линности — *SIM* (*Security Identification Module*) карты. Она содержит международный идентификационный номер подвижного абонента, индивидуальный ключ аутентификации, алгоритм аутентификации. С помощью заложенной в *SIM* карту информации в результате взаимного обмена данными между подвижной станцией и сетью осуществляется полный цикл аутентификации и разрешается доступ абонента к сотовой сети.

Для обеспечения конфиденциальности разговоров пользователей сети связи *GSM* все сообщения передаются в защищенном виде. *SIM* карта абонента содержит алгоритм формирования ключей шифрования. Кроме этого, для исключения идентификации абонента путем перехвата сообщений, передаваемых по радиоканалу, каждому абоненту системы связи присваивается «временное удостоверение личности» — временный международный идентификационный номер пользователя *TMSI*, который действителен только в пределах зоны расположения. В другой зоне расположения ему присваивается новый временный идентификатор *TMSI*. Если абоненту еще не присвоен временный номер (например, при первом включении подвижной станции), идентификация проводится через международный идентификационный номер *IMSI*. После окончания процедуры аутентификации и начала режима шифрования временный идентификационный номер *TMSI* передается на подвижную станцию только в зашифрованном виде. Этот *TMSI* будет использоваться при всех последующих доступах к системе. Если подвижная станция переходит в новую область расположения, то ее *TMSI* должен передаваться вместе с идентификационным номером зоны, в которой *TMSI* был присвоен абоненту.

8.3. Другие устройства защиты информации

Электронные средства аутентификации

В этом разделе будут рассмотрены устройства, которые не имеют прямого отношения к компьютерной технике, однако, используются очень широко в качестве устройств аутентификации доступа к компьютерным системам и сетям. К ним относятся аппаратные ключи, смарт-карты, электронные метки, устройства биометрической защиты.

Первые из названных, *электронные ключи*, также исторически появились первыми. Электронный ключ представляет собой устройство аппаратной защиты, которое присоединяется к последовательному или параллельному порту персонального компьютера или разъему *PCMCIA* ноутбука. Это аппаратный ключ, использующий «зашитые» в него коды и пароли для регулирования доступа к программным приложениям.

Программный продукт с интегрированным в него электронным ключом будет работать только в то время, когда этот ключ прикреплен к компьютеру. Электронный ключ обычно используется для целей лицензионного контроля программного обеспечения. В постоянной памяти ключа (от 64 до 512 байт) хранится в зашифрованном по некоторому алгоритму виде, чаще всего *IDEA* или *DES* последовательность байтов приватного ключа, которая задействуется при прохождении процедуры аутентификации.

Смарт-карта, имея вид обычной пластиковой кредитной карточки, содержит в себе интегральную схему, которая наделяет ее способностями к хранению и обработке информации. В зависимости от встроенной микросхемы все смарт-карты делятся на несколько основных типов, кардинально различающихся по выполняемым функциям: карты памяти; микропроцессорные карты; карты с криптографической логикой. Карты памяти предназначены для хранения информации. Память на таких типах карт может быть свободной для доступа или содержать логику контроля доступа к памяти карты для ограничения операций чтения и записи данных. Микропроцессорные карты также предназначены для хранения информации, но в отличие обычных карт памяти они содержат в себе специальную программу или небольшую операционную систему, которая позволяет преобразовывать данные по определенному алгоритму, осуществлять защиту информации, хранящейся на карте при передаче, чтении и записи. Карты с криптографической логикой используются в системах защиты информации для принятия непосредственного участия в процессе шифрования данных или выработки криптографических ключей, электронных цифровых подписей и другой необходимой информации для работы системы.

Для смарт-карт существует несколько международных стандартов, определяющих практически все свойства карт, начиная от размеров, свойств и типов пластика, и заканчивая содержанием информации на карточке, протоколов работы и форматов данных. Наиболее распространены смарт-карты с контактным считыванием по стандарту *ISO* 7816.

Электронные метки, по своей сути, новое воплощение электронных ключей. В связи с появлением новых поколений однокристальных микропроцессоров, обладающих малыми габаритами и мощными вычислительными способностями, появлением у персональных компьютеров порта *USB*, можно считать оконченной эпоху других видов электронных ключей. Одним из самых распространенных представителей семейства электронных меток является *eToken* — первый представитель нового поколения *token*-устройств, которые могут напрямую подключаться к

универсальной последовательной шине *USB* и не требуют дополнительного оборудования для считывания. *eToken* выполнен в виде брелка со световой индикацией режимов работы. Он позволяет реализовать множество различных защитных функций, например:

- защитить конфиденциальные файлы с помощью шифрования и ограничить доступ к защищенным ресурсам системы с помощью аутентификации;
- обезопасить переписку, используя *eToken* для подписи и шифрования электронных сообщений;
- защитить программное обеспечение (прежде всего, финансовое) от несанкционированного изменения, в частности, от внесения так называемых программных «закладок» и «логических бомб»;
- идентифицировать поступающие в сеть или на Интернет-сервер запросы, гарантируя, что пользователи будут работать только с той информацией, доступ к которой им разрешен.

Разные реализации *eToken* реализуют разные алгоритмы шифрования, в том числе, начиная от *3DES*, с ключом 64 бита, до ГОСТ 28147-89, с размером ключа шифрования 256 битов.

Биометрические средства защиты

Понятие биометрии появилось в конце XIX века как раздел науки, занимающейся количественными биологическими экспериментами с привлечением методов математической статистики. Результаты этой науки относительно недавно стали использоваться в биометрических технологиях с целью использования физиологической или поведенческой характеристики типа отпечатка пальца или образца голоса, для однозначной идентификации личности человека. Основными доводами в пользу развития биометрических компьютерных технологий являются слабые места перечисленных выше традиционных методов аутентификации пользователей:

- идентифицирующий признак (пароль, карточка, ключ) может быть утерян, передан другому физическому лицу или иным образом скомпрометирован;
- не обеспечивается строгая аутентификация пользователя, то есть достоверная проверка того факта, что пароль или другой аутентифицирующий признак был предъявлен лично, а не третьим лицом;
- криптостойкие пароли трудно запоминать, так как они имеют большую длину и сложность, а также их необходимо достаточно часто менять, к тому же их может быть много для каждой системы.

Налицо основное преимущество биометрического подхода к обеспечению информационной безопасности — сам пользователь является реальным подтверждением подлинности субъекта, получающего права доступа. Методы, которыми пользуется биометрика, делятся на статистические (исследования характеристик, данных человеку от рождения, таких как отпечатки пальцев, сетчатка глаза и др.) и динамические (основывающиеся на поведенческой характеристике человека, таких как рукописный и клавиатурный почерки, голос и др.).

Технологические методы получения и обработки биометрической информации, в основном, заключаются в следующем. Пользователь размещает свою ладонь (или, в зависимости от требований метода, другую часть тела) на поверхности считывающего устройства. При необходимости пользователю даются указания точнее позиционировать ладонь в соответствии с ориентирами на считывающем устройстве. Затем устройство сканирования, в качестве которого в большинстве случаев используются приборы с зарядовой связью (ПЗС), считывает данные длины, ширины, поверхностной области руки и пальцев, разделяет шумы от изображений силуэтов окружающих предметов, проецируемых на сканирующее устройство. При сканировании производится несколько десятков измерений, и характеристики руки и пальцев представляются в виде байтового массива измерений. Далее производится сложная обработка и фильтрация информации с использованием математических методов теории распознавания образов и, зачастую, корреляционный анализ, на основе которого принимается решение о соответствии предъявляемого отпечатка тому, который хранится в базе данных.

Биометрические технологии начинают постепенно занимать позиции рынка компьютерных технологий. Из продуктов на биометрической основе представлены биометрическая мышь и биометрический сканер *MatchBook* (производитель *BioLink*) для снятия отпечатков пальцев. Характеристики устройств:

- разрешающая способность ПЗС 284x400 пикселей;
- размер модели отпечатка около 500 байт.

Литература

1. *Симонович С.В.* и др. Информатика. Базовый курс / С.В. Симонович [и др.]. –СПб.: «Питер», 1999.

2. http://www.kbsu.ru/~book

3. *Бутакова М.А., Гуда, А.Н.* Основы информатики: учебное пособие. — Ростов н/Д: РГУПС, 2004.

4. *Фигурнов В.Э.* IBM PC для пользователя. 7-е изд. -С.-Пб.:, АО «Коруна», НПО «Информатика и компьютеры», 1997.

5. *Столлингс В.* Операционные системы: пер. с англ. 4-е изд.. — М.: Изд. дом «Вильямс», 2002.

6. *Соломон Д., Руссинович, М.* Внутреннее устройство Microsoft Windows: Windows 2003, WindowsXP, Windows 2000. Мастер-класс: пер. с англ. — СПб.: Питер; М.: Русская редакция, 2006.

7. *Чернов А.В.* Технология программирования для операционной системы WINDOWS: учеб. пособие. — Ростов н/Д: РГУПС, 2001.

8. *Чернов А.В.* Операционная система Unix: учебное пособие. — Ростов н/Д: РГУПС, 2003.

9. *Робачевский А.М.* Операционная система UNIX. — СПб.: БВХ-Петербург, 2001.

10. *Чернов А.В., Филоненков А.И.* Операционные системы для персональных компьютеров. Элементы теории. Ч. 1: Учебное пособие. — Ростов н/Д: РГУПС, 1999.

11. *Мураховский В.И., Евсеев Г.А.* Железо ПК — 2002. Практическое руководство. — М.: ДЕСС КОМ, 2002.

12. *Таненбаум Э.Б.* Архитектура компьютера. — СПб: Питер, 2003.

13. *Нечитайло Н.М., Потапина Т.В.* Информатика. Устройство персонального компьютера: учебное пособие. — Ростов н/Д: РГУПС, 2004.

14. *Долженков В.А., Колесников Ю.В.* Microsoft Excel 2003. Наиболее полное руководство. — СПб.: БХВ-Петербург, 2004.

15. *Ульрих Л.* Microsoft Office 2003. How to do Everything with Microsoft Office 2003. — М.: АСТ, Астрель, 2005.

16. *Нечитайло, Н.М., Потапина, Т.В.* Информатика. Эффективная работа с Microsoft Office: учебное пособие. — Ростов н/Д: РГУПС, 2004.

17. *Горев А., Ахаян, Р, Макашарипов, С.* Эффективная работа с СУБД — СПб.: Питер, 1997.

18. *Гуда А.Н., Бутакова М.А.* Алгоритмизация и программирование: Учебное пособие. — Ростов н/Д: РГУПС, 2003.

19. *Фаронов В.В.* Турбо Паскаль 7.0. Начальный курс: учеб. пособие. — М. «Нолидж», 1997.

20. *Бондарев В.М., Рублинецкий, В.И., Качко, Е.Г.* Основы программирования.– Харьков: Фолио; Ростов н/Д: Феникс, 1997.

21. *Моргун А.Н., Кривель И.А. Программирование на языке Паскаль. Основы обработки структур данных.* — М.: Вильямс, 2006.

22. *Штайнер Г.* VBA 6.3. — М.: Лаборатория Базовых Знаний, 2002.

23. *Гуда А.Н., Чернов А.В., Бутакова, М.А.* Лабораторный практикум по курсу «Информатика». Метод. указ. к лаборат. работам. — Ростов н/Д: РГУПС, 2005.

24. *Гусева А.И.* Учимся информатике: задачи и методы их решения. — М.: Диалог-МИФИ, 1999.

25. *Олифер В.Г., Олифер Н.А.* Компьютерные сети. Принципы, технологии, протоколы. — СПб.:Питер, 2000.

26. *Галкин В.А., Григорьев Ю.А.* Телекоммуникации и сети: учеб. пособие для вузов. –М.: Изд-во МГТУ им. Н.Э.Баумана, 2003.

27. *Кутузов О.И., Татарникова Т.М., Петров К.О.* Распределенные информационные системы управления. Учебное пособие по курсовому проектированию. С.-Пб.: ГУТ, 2003.

28. *Торокин А.А.* Инженерно-техническая защита информации. — М.: Гелиос АРВ, 2005.

29. *Михеев Р.* VBA и программирование в MS Office для пользователей. — СПб.: БХВ-Петербург, 2006.

30. *Моргун А.Н., Кривель И.А.* Программирование на языке Паскаль. Основы обработки структур данных. — М.: Вильямс, 2006.

31. *Одом У.* Компьютерные сети. Первый шаг. Computer Networking: First-step. — М.: Вильямс, 2006.

32. *Дьяконов В.П.* Internet. Настольная кгнига пользователя. — М.: Солон-Пресс, 2005.

Предметный указатель

Иностранные термины

ANSI 20
API 63, 77
ASCII 20
BIOS 28
Bluetooth 348
Brandmauer 383
CD-ROM 36
CISC 47
CMOS-память 33
CPU 27, 361
DLL 64
DNS 351
DRAM 34
DVD 37
Ethernet 344
FAT 84
FDD 36
Firewall 383
Flash Drive 37
FM 23
FTP 356
HAL 76
Hardware 5
HDD 35, 361
HTML 352
HTTP 352
Hyper-Threading 50
IBM 12
IP 349, 350
MBR 80
OSI 336
Plug-and-Play 57
RAM 33, 361
RISC 47
Shell 86
SIMD 48
Software 5
TCP 338, 350
TCP/IP 336, 383
Unicode 20
URL 352
VBA 316, 319, 321, 323, 327
Wave-Table 24
WWW 350

А

Адаптер 30
Адрес 25
Алгоритм 222
  рекурсивный 277
АЛУ 25, 26
Ассемблер 226
Атака 371

Б

База данных 206
Байт 8
Банк данных 206
Бит 7

В

Векторная графика 22
Видеоадаптер 34
Видеомонитор
  жидкокристаллический 40
  с электронно-лучевой трубкой 38
Видеопамять 34

Г

Гигабайт 8
Гиперссылка 117
Глубина цвета 40

Д

Датологическая модель 212
Двоичная система счисления 11
Двоичное дерево 302
Двоичное кодирование 7
Диаграммы Excel 153
Диспетчер 78
  ввода-вывода 78
  виртуальной памяти 78
  кэша 79
  объектов 78
  окон и графики 79
  процессов 78
Домен 351
Драйверы устройств 76

Ж

Жесткий магнитный диск 35

З

Запросы Access 158, 172
Зарезервированные слова 227

И

Идентификатор 227, 275
  объекта 220
  пользователя 88
Инкапсуляция 312
Интегральные схемы 11, 13
Интерфейс 31, 57
  AGP 58
  CryptoAPI 382
  IDE 58
  PCI 57
  USB 58
  сетевой 334
Инфологическая модель 212, 218
Информационная безопасность 371
Информационные процессы 8
Информационные ресурсы 8
Информационные технологии 8
Информация 6, 7
  защищаемая 369

К

Канал связи 340
Кибернетика 9
Килобайт 8
Класс объектов 216
Кластер 84
Компилятор 226
Контроллер 30
Корневой каталог 84, 87
Кэш 48
  1-го уровня 48
  2-го уровня 48

М

Макрос 104
Массив 249
Мастер 97, 160, 184
Мегабайт 8
Метод пошаговой детализации 223, 332
Микропроцессор 13
Микроядро 77
  ОС 65
Многозадачный режим 72
Модем 46

Н

Наследование 314
Нисходящее проектирование 331

О

Обработчик прерывания 31
ОЗУ 33
Оператор 240
  ввода данных 242
  вывода данных 240
  присваивания 240
  составной 243
  условный 243
  цикла с параметром 245
  цикла с постусловием 248
  цикла с предусловием 247
Операционная система 63
Операционная среда 63
Открытая архитектура 14
Отладка программы 224
Отчеты Access 157, 182
Очередь 299

П

Память 25, 26, 288
  асинхронная динамическая 56
  виртуальная 70, 77
  динамическая 56
  оперативная 29
  основная 29
  синхронная динамическая 56
  системная 55
Панель инструментов 106
Параллельный порт 31
Параметры подпрограмм 274
  значения 276
  переменные 276
  фактические 274
  формальные 274
Периферийное устройство 29
ПЗУ 28
Пиксель 22
Подпрограмма 272
Полиморфизм 315
Порт ввода-вывода 30
Последовательный порт 31
Прерываниями 27
Приложение ОС 63
Примитив 23
Принтер 44
  лазерный 44
  струйный 44
Принципы Джона фон Неймана 25
Программное обеспечение 59
  антивирусное 379

прикладное 60
системное 59
Протокол 334
Процесс 67, 73
Процессор 27

Р

Раздел 83
Размер зерна экрана 40
Разрешающая способность 40
Разрядность 27
Растровое изображение 22
Регистры 27
Рекурсия 278
Ресурс вычислительной системы 70

С

Связанные списки 295
Северный мост 55
Сектор 83
Сетевой адаптер 45, 343
Сеть ЭВМ 333
Система прерываний 31
Система счисления 15
Системная плата 33, 53
Системная шина 29
Системный блок 32
Сканер 45
Сокет 49
Среда передачи данных 340
Ссылка 150
абсолютная 151
относительная 151
смешанная 152
Стек 279, 298
протоколов 334, 338
Стиль 126, 127
Структурное программирование 272, 329
СУБД 157, 182, 207
Супервизор 66, 72
сэмпл 24

Т

Таблица Access 157
Таблицы Access 160, 163
Тактовая частота 27
Типы данных 228
Топология сети 333
Транслятор 208, 226
Триггер 11

У

Указатель 290
нетипизированный 291
типизированный 291
Уровни сетевого взаимодействия
канальный 336, 337
представительный 339
прикладной 339
сеансовый 339
транспортный 338
физический 336
Устройство управления 25
Утилита 208
Утилиты 59

Ф

Файл 267
текстовый 270
типизированный 268
Файловая система 83
Формула в Excel 148
Формы Access 157
Функции в Excel 149

Ч

Чипсет 55

Ш

Шаблон документа 97
Шина адреса 28, 32
Шина данных 28, 32
Шина управления 28, 32

Ю

Южный мост 55

Я

Ядро
ОС 76
СУБД 208
Язык программирования 12, 225
высокого уровня 225
низкого уровня 225

Оглавление

ПРЕДИСЛОВИЕ 3

Глава 1. ПРЕДМЕТ И ЗАДАЧИ ИНФОРМАТИКИ 5
1.1. Основные понятия информатики 5
1.2. История развития вычислительной техники 9
1.3. Представление информации в компьютере 15
1.3.1. Системы счисления 15
1.3.1.1. Правила перевода чисел из одной системы счисления в другую 17
1.3.2. Формы представления данных 19

Глава 2. УСТРОЙСТВО ПЕРСОНАЛЬНОГО КОМПЬЮТЕРА 25
2.1. Принципы функционирования компьютера 25
2.1.1. Общие принципы 25
1.1.2. Начальная загрузка персонального компьютера 28
1.1.3. Логическая структура ПК 29
2.2. Состав персонального компьютера и периферийные устройства 32
2.2.1. Состав ПК 32
2.2.2. Внешние накопители информации 36
2.2.3. Мониторы 38
2.2.4. Принтеры 44
2.2.5. Сканеры 45
2.2.6. Сетевой адаптер (сетевая карта) 45
2.2.7. Модемы 46
2.2.8. Корпус ПК 46
2.3. Аппаратное обеспечение современного ПК 47
2.3.1. Микропроцессоры 47
2.3.2. Системные платы 53
2.3.3. Системный набор 55
2.3.4. Системная память 55
2.3.5. Интерфейсы 57
2.4. Программное обеспечение ПК 59

Глава 3. ОПЕРАЦИОННЫЕ СИСТЕМЫ 63
3.1. Основные сведения из теории операционных систем 63
3.2. ОС семейства Windows 74
3.2.1. Общая структура ОС Windows XP 75
3.2.2. Процесс начальной загрузки 79
3.2.3. Файловые системы Windows XP 83
3.3. ОС семейства UNIX 86
3.3.1. Общие сведения 86
3.3.2. Файловая система Unix 88
3.3.3. Командный язык системы UNIX 89
3.3.3.1. Справочные команды 89

3.3.3.2. Команды работы с каталогами ... 89
3.3.3.3. Команды работы с файлами ... 90
3.3.3.4. Команды работы с текстовыми файлами ... 93
3.3.3.5. Команды работы с процессами ... 94

Глава 4. ОФИСНЫЕ ПРИЛОЖЕНИЯ ОПЕРАЦИОННЫХ СИСТЕМ ... 95
4.1. Введение в MICROSOFT OFFICE ... 95
4.2. Основы работы с Office: окна и панели инструментов ... 99
4.3. Текстовый редактор MS WORD ... 118
4.3.1. Ввод и редактирование текста в MS Word ... 118
4.3.2. Форматирование документа Word ... 124
4.3.2.1. Непосредственное форматирование символов ... 124
4.3.2.2. Настройка стилей и шаблонов Word ... 126
4.3.3. Проверочные средства Word ... 130
4.4. Редактор таблиц MS EXCEL ... 136
4.4.1. Создание и форматирование книги и листа ... 136
4.4.2. Формулы и функции в MS Excel ... 148
4.4.3. Создание диаграмм Excel ... 153
4.5.Система управления базами данных MS ACCESS ... 157
4.5.1. Общие сведения. Создание таблиц ... 157
4.5.2.Проектирование структуры БД в MS Access ... 165
4.5.3. Запросы MS Access ... 172
4.5.4. Отчёты MS Access ... 182
4.6. Интеграция приложений MS OFFICE ... 188
4.7. Модели решения функциональных и вычислительных задач средствами MS Office ... 192
4.7.1. Задача о ранце ... 192
4.7.2. Задача о распределении средств по предприятиям ... 195
4.7.3. Основная задача линейного программирования (ОЗЛП) ... 200
4.7.4. Транспортная задача ... 202

Глава 5. БАЗЫ ДАННЫХ ... 206
5.1. Введение в базы данных ... 206
5.2. Структура и пользователи банка данных ... 207
5.3. Классификация банков и баз данных ... 210
5.4. Этапы проектирования баз данных ... 212
5.4.1. Инфологическое моделирование ... 215
5.4.2. Датологическое моделирование ... 218
5.5. Проектирование реляционных баз данных ... 220

Глава 6. АЛГОРИТМИЗАЦИЯ И ПРОГРАММИРОВАНИЕ ... 222
6.1. Основные этапы решения задач на ЭВМ ... 222
6.2. Общие сведения о языке Паскаль ... 226
6.3. Данные в Паскале. Простые типы данных ... 228
6.4. Операции и выражения ... 232
6.5. Структура программы ... 236
6.6. Операторы языка Паскаль ... 240
6.6.1. Оператор присваивания ... 240
6.6.2. Операторы вывода и вывода информации ... 240

6.6.3. Составной оператор ... 243
6.6.4. Условный оператор ... 243
6.6.5. Оператор варианта CASE ... 244
6.6.6. Операторы цикла ... 245
6.6.6.1. Оператор цикла по счетчику (цикл с параметром) ... 245
6.6.6.2. Оператор цикла с предусловием ... 247
6.6.6.3. Оператор цикла с постусловием ... 248
6.7. Структурированные типы данных ... 249
6.7.1. Массивы ... 249
6.7.2. Строки ... 254
6.7.3. Множества ... 256
6.7.4. Комбинированный тип (записи) ... 262
6.8. Типизированные константы ... 264
6.9. Файлы ... 267
6.9.1. Типизированные файлы ... 268
6.9.2. Текстовые файлы ... 270
6.10. Подпрограммы ... 272
6.10.1. Процедуры и функции ... 273
6.10.2. Параметры подпрограмм ... 276
6.11. Рекурсии ... 277
6.11.1. Рекурсивные алгоритмы и рекурсивные определения ... 277
6.11.2. Рекурсивные процедуры и функции ... 278
6.11.3. Виды рекурсивных процедур ... 280
6.12. Программные модули ... 282
6.12.1. Структура программного модуля ... 283
6.12.2. Трансляция модуля. «Сборка» программы ... 285
6.12.3. Ссылки на модули ... 286
6.13. Динамическая память ... 288
6.13.1. Указатели ... 290
6.13.2. Создание и уничтожение динамических переменных ... 292
6.13.3. Примеры использования указателей ... 293
6.14. Использование указателей для организации связанных динамических структур ... 295
6.14.1. Списки ... 295
6.14.2. Организация стека в динамической памяти ... 298
6.14.3. Очередь ... 299
6.14.4. Деревья ... 301
6.15. Введение в объектно-ориентированное программирование ... 311
6.15.1. Тип Объекты ... 311
6.15.2. Основные понятия ООП. Инкапсуляция. аследование ... 312
6.15.3. Полиморфизм ... 315
6.16. Основы программирования в среде Visual Basic for Application (VBA) ... 316
6.16.1. Типы данных VBA ... 316
6.16.2. Описание переменных ... 317
6.16.3. Константы ... 318
6.16.4. Операции, операторы и встроенные функции VBA ... 319

6.16.5. Ввод и вывод информации ... 321
6.16.6. Реализация разветвляющихся алгоритмов в VBA ... 323
6.16.7. Операторы цикла ... 325
6.16.8. Массивы ... 327
6.17. Структурный подход к программированию ... 329
6.17.1. Основные принципы структурного подхода ... 329
6.17.2. Спецификация программ ... 331
6.17.3. Метод пошаговой детализации ... 332

Глава 7. КОМПЬЮТЕРНЫЕ СЕТИ ... 333
7.1. Общие принципы построения компьютерных сетей ... 333
7.1.1. Концепция открытых систем OSI ... 335
7.1.2. Аппаратное обеспечение компьютерных сетей ... 340
7.1.3. Основы технологий локальных сетей ... 344
7.1.3.1. Локальные сети Ethernet ... 344
7.1.3.2. Беспроводные локальные сети ... 348
7.2. Основы Internet-технологии ... 349
7.2.1. История возникновения и развития Internet ... 349
7.2.2. Базовые протоколы и адресация в Internet ... 350
7.2.3. Организация подключения и работы с Internet ... 353
7.2.4. Информационные и коммуникационные сервисы Internet ... 354
7.2.5. Поиск информации в Internet ... 357
7.3. Моделирование компьютерных сетей средствам теории очередей ... 358
7.3.1. Общие сведения о моделировании информационных систем ... 358
7.3.2. Пример практического применения аппарата сетей массового обслуживания для моделирования компьютерной сети ... 360

Глава 8. ИНФОРМАЦИОННАЯ БЕЗОПАСНОСТЬ И ЗАЩИТА ДАННЫХ ... 368
8.1. Задача обеспечения информационной безопасности в современных условиях ... 368
8.1.1. Информация как объект защиты ... 368
8.1.2. Источники угроз и способы нарушения информационной безопасности ... 370
8.2. Аспекты практической компьютерной безопасности ... 377
8.2.1. Средства анализа защищенности систем и сетей ... 377
8.2.2. Защита компьютеров и сетей ... 379
8.3. Электронные устройства защиты информации ... 387

Литература ... 391

Предметный указатель ... 393

Учебное издание

Гуда А. Н., Бутакова М. А.,
Нечитайло Н. М., Чернов А. В.

ИНФОРМАТИКА
общий курс

Редакционно-издательские работы
выполнены ИП Шаповалов С. В.

Санитарно-эпидемиологическое заключение
№ 77.99.02.953.Д.004609.07.04 от 13.07.2004 г.

Подписано в печать 10.09.2008. Формат 60×88 1/16.
Печать офсетная. Бумага офсетная №1.
Печ. л. 25,0. Тираж 2000 экз. Заказ №6534.

Издательско-торговая корпорация «Дашков и К°»
129347, Москва, Ярославское шоссе, д. 142, к. 732.
Для писем: 129347, Москва, п/о И-347;
Тел./факс: 8 (499) 182-01-58, 182-11-79, 183-93-01.
E-mail: sales@dashkov.ru — отдел продаж;
office@dashkov.ru — офис; http://www.dashkov.ru

Издательство «Наука-Спектр»
344010, г. Ростов-на-Дону, ул. Лермонтовская, 90, к. 8.
e-mail: nauka@aaanet.ru

Отпечатано в соответствии с качеством предоставленных диапозитивов
в ФГУП «Производственно-издательский комбинат ВИНИТИ»,
140010, г. Люберцы Московской обл., Октябрьский пр-т, 403. Тел.: 554-21-86