МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

В.Б. Немировский, А.К. Стоянов

Информатика

*Рекомендовано в качестве учебно-методического пособия
Редакционно-издательским советом
Томского политехнического университета*

Издательство
Томского политехнического университета
2012

УДК 681.3(075)
ББК 32.973.2я73
С37

Немировский В.Б.
С37 Информатика: учебное пособие / В.Б. Немировский, А.К. Стоянов. – Томск: Изд-во Томского политехнического университета, 2012. – 312 с.

В пособии приведены сведения, охватывающие основные разделы курса «Информатика». Рассмотрены понятия информации, информационного процесса, информационных технологий. Приводятся краткие сведения по аппаратно-программным средствам информационных технологий, распространенным операционным системам. Рассмотрены используемые в инженерной деятельности информационные технологии и их инструментальные средства. В разделе «Программирование для инженеров» рассмотрены основные понятия и принципы программирования, основные понятия и технология программирования в среде Delphi. Приводятся сведения по языку программирования Object Pascal, средствам описания данных и средствам описания действий над данными.

Предназначено для студентов направления 140400 ЭНИН ТПУ.

УДК 681.3(075)
ББК 32.973.2я7

Рецензенты

Доктор технических наук, профессор ТПУ
А.Ф. Тузовский

Кандидат технических наук, профессор
Севастопольского национального университета
ядерной энергии и промышленности
А.Г. Рипп

Доктор физико-математических наук,
заведующий лабораторией ИМКЭС СО РАН
Тартаковский В.А.

Содержание

Предисловие

Учебное пособие содержит как основной лекционный материал курса «Информатика», так и его расширение, предназначенное для самостоятельного изучения. В нем приведены сведения, охватывающие основные разделы базового курса информатики. Рассмотрены понятия информации, информационного процесса и его этапов, информационных технологий. Приводятся краткие сведения по аппаратно-программным средствам информационных технологий, распространенным операционным системам. Основное внимание при изложении материала, относящегося к операционной системе Windows, обращалось на объектную ориентацию среды. Такой подход созвучен психологическому восприятию графического интерфейса и облегчает как изложение материала, так и его восприятие начинающими пользователями. Более искушенные пользователи смогут использовать этот материал для выработки системного взгляда на известные им и привычные вещи.

Сведения, относящиеся к средствам сжатия данных, позволяют познакомиться с основными понятиями из этой области и особенностями использования соответствующего программного обеспечения на примере широко распространенного диспетчера архивов WinZip.

Сведения, относящиеся к компьютерной безопасности, знакомят начинающих пользователей компьютера с основными видами компьютерных вирусов и технологией антивирусной защиты. Рассматриваются как ряд организационных мероприятий по защите информации, так и вопросы использования вспомогательных средств – антивирусных программ и средств аппаратной защиты.

Рассмотрены наиболее часто используемые в деятельности технических специалистов информационные технологии и их инструментальные средства. Особое внимание уделено офисным технологиям и их средствам, использование которых сопровождает практическую деятельность как специалистов электротехнического профиля, так и любых других технических направлений.

Раздел «Программирование для инженеров» предназначен для обучения будущих технических специалистов основам программирования. Какие знания из области программирования необходимы в профессиональной инженерной деятельности – вопрос неоднозначный. Для выполнения инженерных расчетов существует значительное количество прикладных программ и специализированных пакетов, изучаемых в специальных курсах и избавляющих от необходимости непосредственного программирования в большинстве случаев. Поэтому данное пособие не касается вопросов создания сложных программных систем для

выполнения инженерных расчетов. Этим занимаются профессиональные программисты, заказчиками которых являются инженеры-специалисты в конкретной предметной области.

Современный технический специалист, на наш взгляд, должен знать программирование в объеме, достаточном для грамотной постановки задачи профессиональному разработчику программного обеспечения. В эту сумму знаний входит и возможность быстрой разработки, в случае необходимости, макета и пользовательского интерфейса необходимого приложения. Поэтому в качестве целей изучения этой дисциплины студентами мы поставили формирование представлений о технологиях и методах современного программирования, а также умений использовать средства визуального программирования для решения возникающих задач.

Средой программирования, в наибольшей степени отвечающей сформулированным выше целям, является известная и популярная среда Delphi, сочетающая объектно-ориентированный подход и технологию визуального программирования. В связи с этим в учебном пособии рассмотрены не только основные понятия и принципы программирования, но и технология программирования в современной среде разработки программных продуктов.

Изучение раздела «Программирование для инженеров» обеспечивает:

- формирование представление о методах и технологиях современного программирования;
- знание состава среды визуального программирования Delphi и технологии работы в среде;
- знание основ языка программирования Object Pascal, который используется в среде;
- знание основных алгоритмов обработки данных;
- умение проектировать и реализовывать несложные приложения в среде визуального программирования Delphi.

В пособии приведено много примеров фрагментов программного кода. Наряду с примерами, содержащими английские слова для именования программных объектов, есть и примеры с использованием для именования транслитерированных русских слов. Это сделано для облегчения понимания смысла фрагментов программ неподготовленными читателями, хотя среди профессиональных программистов такое именование не считается корректным.

Весь материал учебного пособия тесно связан с разработанным авторами компьютерным лабораторным практикумом, который постоянно обновляется, дополняется и в настоящее время включает в себя

более 20 лабораторных работ по основным разделам курса. Практикум представляет собой программную среду для обучения практическим основам информационных технологий с встроенным методическим обеспечением в гипертекстовом формате (справка Windows). Методическое обеспечение содержит весь необходимый материал для выполнения цикла лабораторных работ по темам курса.

Это накладывает отпечаток на содержание пособия. Изложение ведется с учетом того, что в материалы лабораторных работ практикума включены сведения, относящиеся к практической стороне работы с компьютером. Поэтому материал пособия содержит, прежде всего, общие, системообразующие сведения обзорного характера, часто отсутствующие или лишь упомянутые в практикуме. Сочетание этих сведений пособия с использованием практикума обеспечивает эффективное изучение курса «Информатика».

ИНФОРМАЦИЯ И ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ

ГЛАВА 1. ИНФОРМАЦИЯ

Понятие информации

Термин «**информация**» возник от латинского слова informatio – разъяснение, изложение и до середины XX в. означал сведения, передаваемые между людьми.

Существует несколько подходов к объяснению понятия «информация» – от принятого в повседневной жизни неформального подхода, до строгого математического определения в теории информации. В различных источниках можно встретить, например, такие *определения* информации:

информация – это понятие, объединяющее в себе сведения об объектах, свойства которых объясняются;

информация – это сведения об окружающем мире и протекающих в нём процессах, воспринимаемые человеком или специальным устройством;

информация – это сообщения, осведомляющие о положении дел, о состоянии чего-нибудь;

информация – это сведения о лицах, предметах, фактах, событиях, явлениях и процессах независимо от формы их представления;

информация – это сведения, неизвестные до их получения, являющиеся объектом хранения, передачи и обработки;

информация* I *о событии, вероятность появления которого равна* p, *определяется формулой $I = -\log_2 p$.

Выделим три наиболее распространенные концепции информации, каждая из которых по-своему объясняет ее сущность.

Первая концепция (количественно-информационный подход) определяет информацию как меру неопределенности (энтропию) события. Количество информации в том или ином случае зависит от вероятности его получения: чем более вероятным является сообщение, тем меньше информации содержится в нем. Этот подход, хоть и не учитывает смысловую сторону информации, оказался весьма полезным в технике связи и вычислительной технике, послужил основой для измерения информации и оптимального кодирования сообщений. При таком понимании информация – это снятая неопределенность, или результат выбора из набора возможных альтернатив.

Вторая концепция рассматривает информацию как свойство (атрибут) материи. Ее появление связано с развитием кибернетики и основано на утверждении, что информацию содержат любые сообщения,

воспринимаемые человеком или приборами. Наиболее ярко и образно эта концепция информации выражена академиком В.М. Глушковым. Он писал, что «информацию несут не только испещренные буквами листы книги или человеческая речь, но и солнечный свет, складки горного хребта, шум водопада, шелест травы». Иными словами, информация как свойство материи создает представление о ее природе и структуре, упорядоченности, разнообразии и т. д. Она не может существовать вне материи, а значит, она существовала, и будет существовать вечно, ее можно накапливать, хранить, перерабатывать.

Третья концепция основана на логико-семантическом (семантика – изучение текста с точки зрения смысла) подходе, при котором информация трактуется как знание, причем не любое знание, а та его часть, которая используется для активного действия. Иными словами, информация – это действующая, полезная, «работающая» часть знаний.

Рассмотренные подходы в определенной мере дополняют друг друга, освещают различные стороны сущности понятия информации и облегчают тем самым систематизацию ее основных свойств. Из множества определений информации наиболее целесообразным представляется следующее:

Информация – это сведения об окружающем мире и протекающих в нём процессах, являющиеся объектом хранения, передачи и обработки.

Формы представления информации

Существует несколько форм представления информации. К основным можно отнести символьную, текстовую и графическую формы представления. **Символьная** основана на использовании символов – букв, цифр, знаков, в том числе знаков пунктуации и других знаков. **Текстовая** также использует образующие тексты символы, но расположенные в определенном порядке. Самой емкой и сложной является **графическая** форма представления информации. К ней относятся различные виды изображений.

Представление информации в компьютере

В компьютере любая форма информации представляется последовательностью двоичных чисел (внутреннее представление). Для представления одного двоичного числа требуется одна единица информации, которая называется 1 БИТ. Существуют более крупные единицы:

1 байт = 8 бит,

1 Кбайт = 1024 байт (2^{10} байт),

1 Мбайт = 1024 Кбайт, и т. д.

Для представления одной литеры из множества литер требуется 8 бит или 1 байт информации, 1 страница машинописного текста (30 строк по 60 символов) требует около 2К, 1000 чисел по 4 байта – около 4 К.

Различные формы информации кодируются группами двоичных чисел по-разному. При этом коды соответствуют основным признакам некоторого вида информации (например, код символа для текста, наличие и цветовые характеристики точки изображения и т. д.).

Понятие об информационных технологиях

Технология *– совокупность средств (методов, документации и технических устройств), позволяющая получить из исходных продуктов требуемый конечный продукт.*

Метод *– способ достижения какой-либо цели, решения конкретной задачи; совокупность приемов практического и теоретического освоения (познания) действительности.*

Если исходным и конечным продуктом служит информация, то такую технологию будем называть *информационной технологией (ИТ)*. Чаще всего технической базой информационных технологий служит компьютерная техника и техника связи (коммуникационная).

Компьютер (электронная вычислительная машина, ЭВМ, вычислительная машина) – комплекс технических и программных средств, предназначенный для обработки и хранения информации.

ГЛАВА 2. АППАРАТНЫЕ СРЕДСТВА КОМПЬЮТЕРА

Краткая история развития компьютерной техники

В конце восемнадцатого века английским математиком Чарльзом Бэббиджем была изобретена «аналитическая машина» – механический прообраз современного компьютера.

В 1939 г. в США физиком Атаносовым была создана настольная электромеханическая модель вычислительной машины. Далее в хронологическом порядке приведены некоторые примечательные события и факты из истории развития компьютеров.

1941 год. Полностью работающий компьютер

В Германии **Конрад Цузе** завершает создание **Z3**, первого в мире полностью работающего компьютера. Схема Z3 использует электромагнитные реле. В Z3 реализованы все элементы современных компьютеров, **однако программы хранились на внешнем носителе** *(на кинопленке-перфоленте)*.

1944 год. Большая вычислительная машина в США

В Гарвардском университете **Говард Эйкен** представляет первую в США полностью программируемую вычислительную машину. В **MARK I** использованы электромагнитные реле и десятичная система счисления. Длина машины составляла 17 метров, операцию сложения она производила за 0,3 секунды.

1944 год. Вычислительная машина взламывает шифры

Британские взломщики шифров запускают в работу машину **COLOSSUS**. Она позволяет им расшифровывать телетайп-сообщения, которыми обмениваются вооруженные силы Германии. Машина использует электронные лампы и работает в двоичной системе. Она способна обрабатывать 5 000 символов в секунду.

1945 год. Внутреннее хранение программ

Венгерско-американский математик **Джон фон Нейман (Янош Лайош Нейман)** описал концепцию современного компьютера: ***программы должны храниться так же, как данные, − в памяти компьютера, чтобы к ним можно было получить быстрый доступ и их было легко отредактировать.***

Практически все компьютеры, созданные с тех пор имеют логическую структуру, заданную этими принципами (говорят: «**имеют архитектуру фон Неймана**»).

1946 год. Электронная вычислительная машина

В **США** создан первый компьютер с полностью электронной архитектурой. В машине **ENIAC** использовано около 18 000 электронных ламп и она примерно в 1 000 раз быстрее машин, основанных на электромагнитных реле. Программирование компьютера занимает несколько дней.

1948 год. Бит

Американский математик **Клод Шеннон** впервые использует термин «**бит**» (один двоичный разряд − 0 или 1) для наиболее мелкой единицы информации. Он утверждает, что любая информация может быть представлена в виде битов.

1948/1949 год. Хранение программ

В Великобритании запущены компьютеры, которые способны хранить программы и данные в электронном виде: экспериментальная машина BABY, созданная Манчестерским университетом и EDSAC, сконструированная Кембриджским университетом.

1951 год. Монитор

Первый монитор в истории компьютеров был разработан в США **для большой ЭВМ (мейнфрейма) WHIRLWIND**. Этот военный компьютер использовался для наблюдения за американским воздушным пространством. Вражеские самолеты представлялись на экране в виде графических символов.

1955 год. Компьютер на транзисторах

В **США** построен первый транзисторный компьютер − **TRADIC** (TRAnsistorised Airborne Digital Computer). Схемы, построенные на транзисторах, компактнее, быстрее и надежнее − а в недалеком будущем становятся и дешевле, − чем схемы на электронных лампах. Начинается коммерческое использование компьютеров.

1958 год. На пути к микрочипу

Американский инженер **Джек Килби** разрабатывает **интегральную схему**. Транзисторы, резисторы и другие электронные компоненты производятся из одного материала и объединены в один модуль. Таким образом компьютеры становятся значительно компактнее и эффективнее. *(Следует заметить, что ИС, созданная Джеком Килби, была выполнена из дорогого германия. Кремниевый микрочип был впервые получен через полгода Робертом Нойсом, впоследствии вместе с Городоном Муром основавшим компанию Intel.)*

1964 год. «Семейство компьютеров»

Американская фирма IBM представляет System/360. Благодаря модульной конструкции этого компьютера, IBM удается добиться долговременного сотрудничества со своими клиентами, которые могут компоновать свои собственные вычислительные системы, выбирая из шести различных по возможностям мейнфреймов и 40 периферийных устройств.

1965 год. Миникомпьютер

На рынке появляется первый **миникомпьютер, PDP-8**. Миникомпьютеры значительно дешевле мейнфреймов и могут сравнительно легко программироваться самими пользователями. Благодаря этим преимуществам компьютеры начинают появляться в небольших фирмах и научных отделах.

1968 год. Компьютерная мышь

Американский исследователь **Дуглас Энгельбарт** представляет свой «указатель положения X-Y для системы отображения». Этот механизм, предназначенный для работы с графическими интерфейсами, теперь известен как «мышь». *(Одновременно с мышью Энгельбарт продемонстрировал концепты электронной почты, гипертекста, видеоконференций, систем обработки текста, совместного одновременного редактирования файлов, мультимедиа, графического интерфейса. А также множества других вещей, которые в то время выглядели совершенно фантастично, а много позже, спустя десятилетия, прочно вошли в жизнь людей.)*

1969 год. Начало эры Интернета

Через телефонное соединение в Америке объединены компьютеры четырех исследовательских институтов. К 1973 году эта компьютерная сеть содержит 35 узлов. Некоторое время спустя во Франции построена первая европейская компьютерная сеть.

1975 год. Микрокомпьютер

Микрокомпьютер **Altair 8800**, сперва продававшийся исключительно в виде набора деталей «сделай сам», стал фантастически успешным. В эру микрокомпьютеров ключевым элементом становятся микрочипы: эти миниатюрные элементы содержат в себе полноценный процессор.

1975 год. Фирмы, разрабатывающие программное обеспечение

Билл Гейтс и Пол Аллен основывают компанию **Microsoft**. Она быстро приобретает известность благодаря языку программирования **BASIC**, разработанному для компьютера Altair. Теперь даже любители могут писать простые программы.

1977 год. Персональный компьютер

Компания **Apple** рекламирует свой **Apple II** как «персональный компьютер». В отличие от его предшественника, Apple I, который покупатели должны были собирать сами, Apple II − первый микрокомпьютер, который можно купить полностью собранным.

1981 год. Портативный компьютер

Первым портативным компьютером, который попал в продажу, стал **Osborne 1**. Компьютер, оборудованный экраном размером с кредитную карточку, весил **12 килограмм**, и по контрасту с будущими ноутбуками его скорее следовало бы назвать «переносносным», а не «портативным».

1982 год. C64

Домашний компьютер **Commodore 64** продается в количестве 30 миллионов экземпляров и становится самой продаваемой моделью компьютера всех времен. Благодаря своим мощным звуковому и графическому чипам, C64 становится лучшим компьютером для фанатов компьютерных игр. ***(В те времена было модно указывать в названии компьютера объем оперативной памяти, в данном случае 64 Кбайт**.)*

1991 год. Всемирная паутина

Разработанная **Европейским центром ядерных исследований (CERN)** Всемирная паутина открыта для общего пользования. Благодаря специальному протоколу передачи данных, унифицированным сетевым адресам и языку разметки страниц HTML, теперь можно обмениваться информацией по всему миру.

1996 год. Компьютер побеждает мирового чемпиона по шахматам

Компьютер, разработанный для игры в шахматы, впервые побеждает сильнейшего в мире игрока-человека. Компьютер **IBM Deep Blue** выигрывает партию в матче против многократного чемпиона мира по шахматам, Гарри Каспарова. В 1997 году компьютер выигрывает у Каспарова и весь матч.

1998 год. Google

Появляется и быстро становится лидером рынка поисковая система Google. Компания занимается интенсивными исследованиями алгоритмов сортировки, которые приводят к хорошей точности результатов поиска.

2003 год. Социальные сети

Создана первая социальная сеть − **Myspace**. Через полгода за ней следует **Facebook**. Люди могут создавать бесплатные учетные записи в Интернете и обмениваться текстами, фотографиями, музыкой и видео.

2007 год. Компьютер в кармане

Компания Apple представляет **iPhone**. Он и другие так называемые смартфоны демонстрируют направление интеграции изначально отдельных устройств − таких как мобильный телефон, компьютер, цифровая камера − в одно многофункциональное устройство.

2010 год. Суперкомпьютеры

В июне 2010 года список самых мощных суперкомпьютеров возглавляет американский **Cray Jaguar**, за ним с небольшим отставанием следует китайский **Nebulae**. Оба этих высокопроизводительных суперкомпьютера могут выполнять более **триллиона вычислений в секунду**.

Будущее. Вычисления с помощью квантов?

У истории компьютеров пока нет окончания. К примеру, уже много лет проводится интенсивное изучение квантовых вычислений. Компьютер, использующий изменение квантовых состояний − так называемых «кубитов», или квантовых битов, может стать в несколько раз более быстрым, чем привычные для нас системы.

Классификация компьютеров

Классификация компьютеров по принципу действия

По форме представления информации, с которой работают вычислительные машины, они делятся на три больших класса: аналоговые (АВМ), цифровые (ЦВМ) и гибридные (ГВМ).

АВМ работают с информацией, представленной в непрерывной (аналоговой) форме. Чаще всего это непрерывный ряд значений электрического напряжения. Аналоговые вычислительные машины весьма просты и удобны в эксплуатации; программирование задач для решения на них, как правило, нетрудоемкое; скорость решения задач изменяется по желанию оператора и может быть сделана сколь угодно большой, но точность решения задач очень низкая (относительная погрешность 2–5 %). На АВМ наиболее эффективно решать математические задачи, содержащие дифференциальные уравнения, не требующие сложной логики.

ЦВМ – вычислительные машины, предназначенные для работы с информацией, представленной в дискретной, цифровой форме.

ГВМ – вычислительные машины комбинированного действия, работают с информацией, представленной в цифровой и в аналоговой форме; они совмещают в себе достоинства АВМ и ЦВМ. Гибридные вычислительные машины целесообразно использовать для решения задач управления сложными быстродействующими техническими комплексами.

Классификация компьютеров по назначению

По назначению компьютеры можно разделить на три группы: универсальные (общего назначения), проблемно-ориентированные и специализированные (рис. 1).

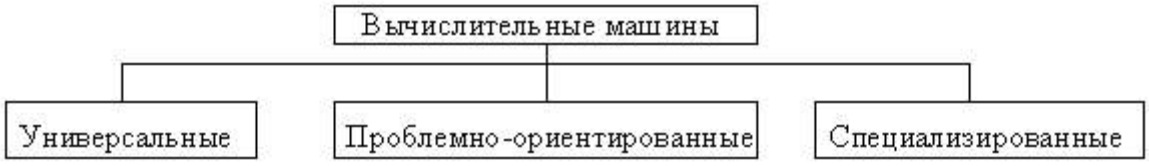


Рис. 1. Классификация компьютеров по назначению

Универсальные компьютеры предназначены для решения самых различных видов задач: научных, инженерно-технических, экономических, информационных, управленческих и других задач. В качестве универсальных компьютеров используются различные типы компьютеров, начиная от суперкомпьютеров и кончая персональными компьютерами. Решаемые на этих компьютерах задачи отличаются сложностью алгоритмов и объемами обрабатываемых данных.

Проблемно-ориентированные компьютеры служат для решения более узкого круга задач, связанных, как правило, с управлением технологическими объектами; регистрацией, накоплением и обработкой относительно небольших объемов данных; выполнением расчетов по относительно несложным алгоритмам. На проблемно-ориентированных компьютерах, в частности, создаются всевозможные управляющие вычислительные комплексы.

Специализированные компьютеры используются для решения еще более узкого круга задач или реализации строго определенной группы функций. Такая узкая ориентация компьютеров позволяет четко специализировать их структуру, во многих случаях существенно снизить их сложность и стоимость при сохранении высокой производительности и надежности их работы.

Классификация компьютеров по производительности

В связи с развитием элементной базы, во многом общей для цифровых вычислительных машин, грань между классами компьютеров становится весьма *размытой* и во многом начинает носить условный характер. Поэтому обсуждаемая здесь классификация – скорее дань традиции, чем отражение реалий развития современной вычислительной техники, носящих принципиальный характер. Лишь одна тенденция принимает доминирующий характер – *ускорение* сближения вычислительных возможностей компьютеров *разных* классов.

В соответствии с этой классификацией, по производительности компьютеры принято было разделять (рис. 2) на суперкомпьютеры, большие, малые и микрокомпьютеры.

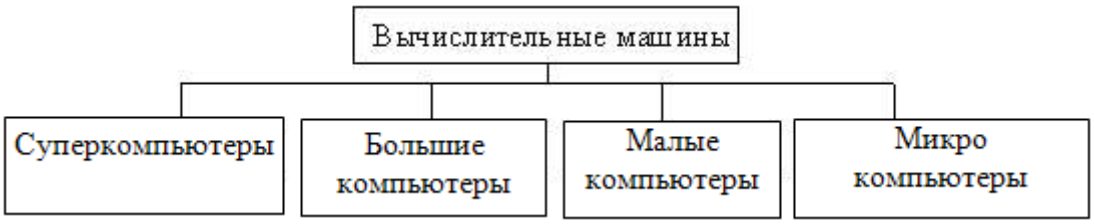


Рис. 2. Классификация компьютеров по производительности

Исторически первыми появились большие компьютеры, элементная база которых прошла путь от электронных ламп до интегральных схем со сверхвысокой степенью интеграции.

Принципиальное устройство компьютера

Схематически устройство одинаково для всех типов компьютеров с архитектурой фон Неймана (см. рис. 3).

Архитектура фон Неймана – это организация компьютера, при которой компьютер состоит из двух основных частей: линейно-адресуемой памяти и процессора. В памяти хранятся команды (программа) и данные, а процессор выбирает команды и данные из памяти и выполняет их.

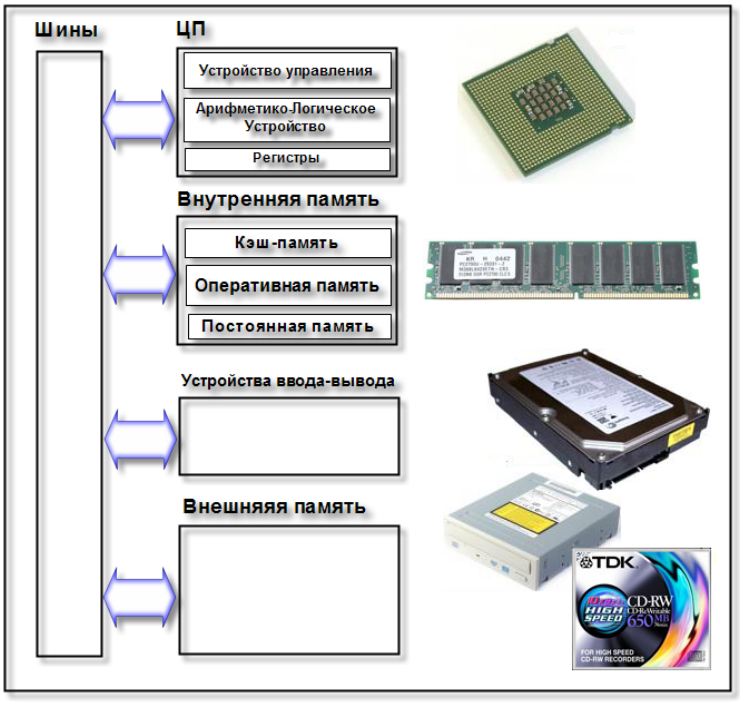


Рис. 3. Принципиальное устройство компьютера

Главная особенность этих компьютеров заключается в том, что команды и данные храняться в одной и той же памяти (например, в значительно менее распространенных компьютерах с Гарвардской архитектурой для хранения программ и данных используется разная память).

Далее изложены принципы работы компьютера с архитектурой фон Неймана.

Принципы работы компьютера

Вначале с помощью какого-либо внешнего устройства в оперативную память компьютера вводится программа.

Устройство управления ЦП считывает из ячеек памяти инструкции программы (команды) и организует их выполнение.

Эти инструкции могут задавать выполнение арифметических или логических операций с помощью арифметико-логического устройства, чтение из памяти данных для выполнения этих операций или запись их результатов в память, ввод данных из внешнего устройства в память или вывод данных из памяти на внешнее устройство.

Считав команду, устройство управления извлекает из нее информацию о коде команды и местонахождении в оперативной памяти данных, обрабатываемых ею. После этого устройство управления повторно обращается к памяти и считывает из нее обрабатываемые командой данные. В результате при выполнении команд обработки данных процессору приходится неоднократно обращаться в память.

В процессорах с Гарвардской (и подобными ей) архитектурой достигается значительное увеличение производительности за счет одновременного параллельного извлечения из разной памяти команд и обрабатываемых данных за один такт процессора.

Результаты, полученные после выполнения всей программы вычислений, передаются на устройства вывода информации. В качестве таких устройств могут использоваться экран дисплея, принтер, графопостроитель и др.

Краткая характеристика аппаратных устройств компьютера

Аппаратура компьютера представляет собой сложную систему взаимосвязанных модулей – устройств. Эти устройства поддерживают выполнение некоторых функций, нужных для работы всей системы в целом. Можно выделить несколько таких важных функций и сопоставить им те модули аппаратуры, которые поддерживают их выполнение. Такую выделенную группу устройств принято называть функциональной подсистемой. Совокупность таких подсистем представляет собой *обобщенную структуру* компьютера и полностью характеризует любой компьютер, независимо от его класса. Выделение таких подсистем позволяет облегчить понимание того, как работает вся система аппаратуры компьютера в целом.

Обобщенная структура компьютера приведена на рис. 4.

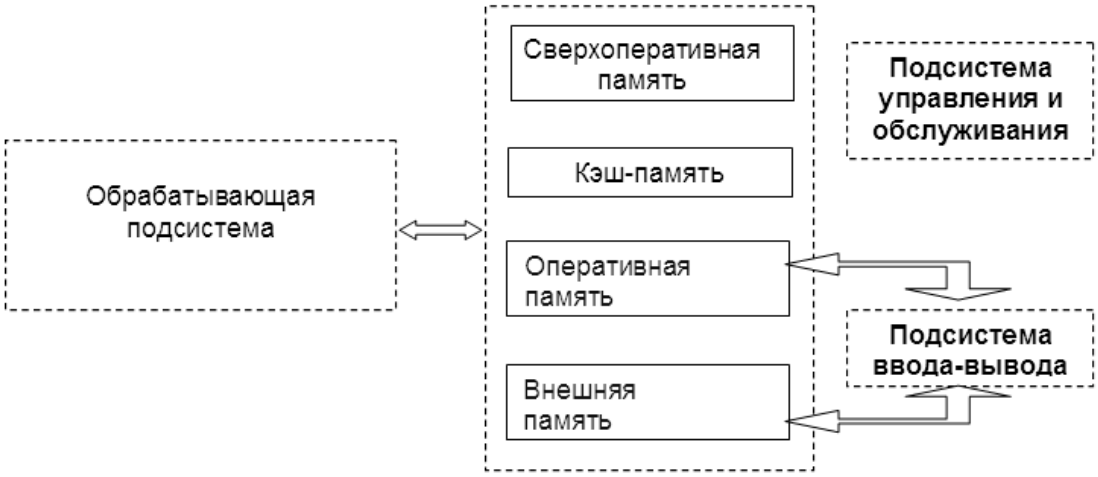


Рис. 4. Обобщенная структура компьютера

Она состоит из следующих составных частей:

- обрабатывающей подсистемы;
- подсистемы памяти;
- подсистемы ввода-вывода;
- подсистемы управления и обслуживания.

Ниже пойдет речь об основных устройствах персонального компьютера (ПК), входящих в его функциональные подсистемы.

Обрабатывающая подсистема

Основным устройством этой подсистемы является *процессор*. Процессором называется устройство, непосредственно осуществляющее обработку данных и программное управление этим процессом. Различают следующие процессоры: центральные, специализированные, ввода-вывода, передачи данных и коммуникационные.

В зависимости от набора и порядка выполнения команд процессоры подразделяются на три класса. Ранее других появились процессоры CISC. Они имеют классическую архитектуру, характеризуемую большим набором команд. Благодаря этому процессоры выполняют самые разнообразные задачи обработки данных. Вместе с этим стало необходимо повысить скорость работы процессоров. Одним из решений задачи стал процессор RISC, который характеризуется сокращенным набором быстро выполняемых команд. В сокращенный набор RISC вошли только наиболее часто используемые команды. Ряд редко встречающихся команд процессора CISC выполняется последовательностями команд процессора RISC. К третьему классу относятся процессоры, совмещающие в себе свойства CISC и RISC процессоров.

Основные задачи по обработке данных решает *центральный процессор*. В процессор входят:

- арифметико-логическое устройство для выполнения арифметических или логических операций;
- устройства управления (счетчик команд, дешифратор команд и другие устройства) организующие и контролирующие работу частей ЦП, взаимодействие их между собой и другими частями компьютера;
- регистры – внутренняя память ЦП.

Изложенные ранее принципы работы компьютера дают несколько упрощенную картину. На деле все происходит гораздо сложнее. Вначале счетчик команд извлекает команду, находящуюся по адресу, который в нем содержится, и увеличивает адрес на 1. Дешифратор определяет тип команды и из скольких байтов она состоит. Далее команда поступает в

АЛУ, которое выполняет записываемое в команде действие. Если команда является командой перехода, то в счетчик записывается новый адрес.

Сложнее организована и работа с памятью. Современные процессоры работают быстрее, чем оперативная память. Поэтому в процессор встраивают имеющую небольшой объем кэш-память, которая работает намного быстрее, чем оперативная. Для выборки следующей команды или данных для нее процессор обращается в быструю кэш-память. Только если команда не найдена в кэш-памяти, процессор, через посредство чипсета, обращается к нужному адресу оперативной памяти и копирует найденную команду или данные к ней, а также целый блок рядом расположенных ячеек оперативной памяти в кэш-память. Поэтому при следующем обращении процессора к кэш-памяти (обычно на следующем такте процессора после неудачного обращения) нужная команда, а также некоторое количество последующих команд и данных к ним будут найдены в кэш-памяти. Этим достигается значительное сокращение обращений процессора к медленной оперативной памяти, поскольку все нужные и наиболее часто используемые процессором в течение некоторого промежутка времени данные будут у него «под рукой».

Центральный процессор в ПК выполнен в виде интегральной микросхемы, называемой микропроцессором. Сейчас на мировом рынке лидируют два производителя процессоров − компании Intel и AMD. Родоначальником семейства Intel был 16-разрядный процессор Intel 8086. содержавший 29000 транзисторов. Современные процессоры содержат миллиарды транзисторов.

Линейка процессоров Intel (с 1978 года): 8086, i286, i386, i486, Pentium, Pentium II, Pentium III, Celeron (упрощённый вариант Pentium), Pentium 4, Core 2 Duo, Core 2 Quad, Core i3, Core i5, Core i7, Xeon (серия процессоров для серверов), Itanium, Atom (серия процессоров для встраиваемой техники) и др.

AMD имеет в своей линейке процессоры архитектуры x86 (аналоги 80386 и 80486, семейство K6 и семейство K7 − Athlon, Duron, Sempron) и x86-64 (Athlon 64, Athlon 64 X2, Phenom, Opteron и др.).

Микропроцессор обычно характеризуется своим типом и тактовой частотой. Она указывает, сколько элементарных операций выполняется в секунду. Частота измеряется в мегагерцах. Производительность ЦП характеризуется следующими параметрами:

- степень интеграции, которая показывает, сколько транзисторов может уместиться в микросхеме ЦП;
- разрядность обрабатываемых данных – количество бит, которые процессор способен обрабатывать одновременно (16, 32 или 64);
- тактовая частота;
- размер адресуемой памяти.

Подсистема памяти

Подсистема памяти компьютера включает в себя совокупность запоминающих устройств (носители информации), и устройств, обеспечивающих чтение и запись информации.

Запоминающие устройства организуются в виде иерархической структуры с различным быстродействием и емкостью (сверхоперативная память, кэш-память, оперативная память, внешняя память). Чем выше уровень элемента структуры (соответствующей памяти), тем выше его быстродействие, но меньше емкость. Любое запоминающее устройство компьютера характеризуется объемом (размером), то есть числом единиц информации, максимально возможным для хранения.

К верхнему (с*верхоперативному*) уровню относятся регистры процессора.

На втором уровне находится кэш-память.

На третьем уровне находится *основная* или *оперативная* память.

Эти три уровня объединяют устройства, расположенные на материнской плате или плате процессора. Эти устройства относятся к категории, называемой внутренней памятью.

На четвертом уровне иерархии размещается внешняя память, к которой относятся запоминающие устройства на магнитных и оптических носителях: на жестких и гибких магнитных дисках, магнитных лентах, оптических дисках и др. Их отличает более низкое быстродействие и очень большая емкость.

Внутренняя и внешняя память используются различными способами.

В *постоянной памяти* (**ROM**) помещаются программы, необходимые для запуска компьютера и важнейших компонентов операционной системы.

Более дорогая *оперативная память* (**RAM**) является хранилищем программного кода и данных. Главной особенностью оперативной памяти является то, что при выключении компьютера вся хранимая в нем информация бесследно исчезает. При работе компьютера данные и программа загружаются в оперативную память, откуда процессор и берет их для обработки (непосредственно, либо через кэш-память). В нее же записывают полученные результаты.

Кэш-память (**CASH**) позволяет увеличить производительность процессора за счет хранения часто используемых команд и данных.

Внешняя память обычно используется для хранения файлов, содержимое которых может быть произвольным.

Подсистема управления и обслуживания

Материнская плата

Материнская плата является основным компонентом системы, объединяющим все остальные компоненты компьютера; она определяет работу процессора и памяти, распределяет все информационные потоки компьютера, и управляет питанием каждого компонента. Именно от правильного выбора материнской платы будут зависеть возможности, стабильность и расширяемость системы.

На материнской плате расположены:

- процессор – основная микросхема, выполняющая математические и логические операции;
- чипсет (системная логика) – набор микросхем, которые руководят работой внутренних устройств ПК и определяют функциональные возможности материнской платы;
- шины – набор проводников, по которым происходит обмен сигналами между внутренними устройствами компьютера;
- оперативная память – набор микросхем, предназначенных для временного сохранения данных, пока включен компьютер;
- постоянная память – микросхема, предназначенная для долговременного хранения данных, даже при отключенном компьютере;
- разъемы для подсоединения дополнительных устройств (слоты).

Размеры платы, тип ее электропитания и структура расположения на ней компонентов определяются ее *форм-фактором*.

Форм-фактор − мировой стандарт определяющий размеры материнской платы места ее крепления к корпусу расположение на ней интерфейсов, портов, сокета ЦП, слотов для оперативной памяти, а также тип разъема для подключения блока питания. Форм-фактор определяет как материнские платы, так и форматы корпусов а также функциональные возможности блоков питания.

Спецификация форм-фактора определяет обязательные и необязательные компоненты. Однако большинство производителей системных плат предпочитают соблюдать всю спецификацию, поскольку иначе могут возникнуть с совместимостью плат и других комплектующих. В настоящее время в настольных компьютерах распространены форм-факторы ATX, microATX, NLX, WTX. Иногда используются и другие.

Чипсет

Чипсет (***Chipset)*** – набор микросхем, управляющих работой внутренних устройств компьютера, обменом данными с внешними устройствами и определяющих основные функциональные возможности материнской платы. Chipset буквально означает – набор микросхем, а поскольку микросхемы логические, часто для краткости их называют «системная логика».

Параметры производительности и функциональности материнской платы напрямую связаны с тем чипсетом, который используется на конкретной плате. Функциями чипсета является обмен и регулировка потоков данных между процессором и всеми устройствами, находящимися на плате – памятью, системными шинами, интегрированными устройствами (видео- и аудиоконтроллеры), контроллерами жестких дисков и т. д.

В настоящее время большинство чипсетов выпускаются на базе двух микросхем, получивших название «северный мост» и «южный мост».

Названия Северный и Южный – исторические. Они означают расположение чипсета моста относительно шины PCI: Северный находится выше, а Южный – ниже. Это название дали чипсетам по выполняемыми ими функциями: они служат для связи различных шин и интерфейсов. Северный мост работает с самыми скоростными устройствами, поэтому сам должен работать очень быстро, обеспечивая быструю и надежную связь процессора, памяти, шины AGP и Южного Моста. Южный мост работает с медленными устройствами, такими как жесткие диски, шина USB, PCI, периферийные устройства и т. п.

Подсистема ввода-вывода

Производительность и эффективность использования компьютера определяются не только возможностями его процессора и характеристиками основной памяти, но в очень большой степени составом его периферийных устройств (ПУ), их техническими данными и способом организации их совместной работы с ядром (процессором и основной памятью) компьютера.

Шина

Шина – это линия обмена данными между отдельными элементами и устройствами на материнской плате. По функциональному назначению различают три категории шин: шина данных, адресная шина и шина управления.

Шина данных. По этой шине происходит копирование данных из оперативной памяти в регистры процессора и наоборот. В ПК на базе

процессоров Intel Pentium шина данных 64-разрядная. Это означает, что за один такт на обработку поступает сразу 8 байт данных.

Адресная шина. Данные, которые передаются по этой шине, трактуются как адреса ячеек оперативной памяти. Именно с этой шины процессор считывает адреса команд, которые необходимо выполнить, а также данные, которые обрабатывает команда. В современных процессорах адресная шина 32-разрядная, она обеспечивает адресацию до 4 Гбайт памяти.

Шина команд. По этой шине из оперативной памяти поступают команды, выполняемые процессором. Команды представлены в виде байтов. Простые команды вкладываются в один байт, но есть и такие команды, для которых нужно два, три и больше байта. Большинство современных процессоров имеют 32-разрядную командную шину, хотя существуют 64-разрядные процессоры с командной шиной.

Шины на материнской плате используются не только для связи с процессором. Все другие внутренние устройства материнской платы, а также устройства, которые подключаются к ней, взаимодействуют между собой с помощью шин. От архитектуры этих элементов во многом зависит производительность ПК в целом.

Устройства ввода-вывода

Ниже приводится перечень некоторых устройств ввода-вывода.

- Клавиатура (для ввода команд ОС, информации, управления работой программы в режиме диалога)

Будучи единообразными по архитектуре, клавиатуры различаются по конструкции замыкающих элементов (контактные, мембранные и др.), по дизайну (эргономические, широкие, узкие) и по способу подключения. Количество клавишей может варьироваться от 101 до 105. Расположение клавиш на клавиатуре подчиняется единой общепринятой схеме, спроектированной в расчёте на английский алфавит.

- Специальные устройства управления (мышка и др.) позволяют обходиться без клавиатуры
- Монитор (монохромный или цветной) позволяет выводить на него тексты и любые графические изображения. В современных мониторах изображение формируется с помощью жидкокристаллической матрицы.
- Принтер

Матричные принтеры. Принцип печати состоит в том, что иголочки, расположенные на вращающемся барабане, пробивают красящую ленту в требуемых местах в нужное время. Качество печати невысокое, но его можно повысить за счет нескольких проходов текста при печати. Скорость печати примерно 1стр/(10−60 с).

Струйные принтеры печатают за счет чернил, выдуваемых из микросопел. Качество печати выше, скорость печати всех без исключения струйных принтеров оставляет желать лучшего − на печать одной страницы насыщенной графикой и цветом уйдет от одной до нескольких «бесконечных» минут. Печать текстов, особенно в режиме черновика идет быстрее, со скоростью нескольких (обычно трех) страниц в минуту. Стоимость выше, чем у матричных принтеров. В последнее время благодаря внедрению новых технологий увеличивается разрешающая способность печати, достигается великолепная цветопередача, увеличивается скорость печати, а сама печать по качеству не уступает фотографии.

Лазерные принтеры дают качество печати близкое к типографскому. Принцип действия основан на том, что печатающий барабан электризуется с помощью лазера по командам из компьютера. Для закрепления налипшего на электризованные места порошка тонера лист носителя проходит устройство термической обработки, в котором происходит микроотжиг тонера. Разрешение 300, 600, 800 и более точек на дюйм, скорость – 1 стр/(5–15 с). Популярные модели лазерных принтеров стоят прилично дороже струйных. Но на их стороне высокое качество монохромной печати (обычно с разрешением 600×600 точек на дюйм), высокая скорость печати (во много раз выше, чему струйных). Лазерные принтеры почти бесшумны, но потребляют больше энергии, чем остальные типы принтеров.

- Приводы для работы с компакт-дисками и DVD-дисками
- Сканер – устройство для построчного считывания графической и текстовой информации в компьютер. В результате изображение представляется в виде матрицы с большим числом точек разложения. Необходимо подчеркнуть, что сканеры работают только с графическим изображением, что бы ни было на обрабатываемом носителе. Если это текст, то дальнейшее его преобразование к текстовым символам – их кодам может быть произведено специальным программным обеспечением. Бывают сканеры ручные и настольные, черно-белые и цветные.
- Модем – устройство для обмена информацией с другими компьютерами через телефонную сеть. Бывают встроенные в компьютер и внешние модемы.
- Многофункциональное устройство (МФУ) – устройство, объединяющее в себе копировальный аппарат, принтер и сканер. Иногда к этим функциям добавляют факс, модем и телефон.

Перечисленные выше устройства обеспечивают полноценный обмен информацией между компьютером и пользователем.

ГЛАВА 3. ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

Программное обеспечение превращает аппаратуру в персональный компьютер.

Программное обеспечение – совокупность программных средств для переработки данных, а также для управления всеми программными и аппаратными средствами компьютера и их взаимодействием.

Программное обеспечение делится на:

- системное:
  - операционные системы;
  - сервисные программы (утилиты и оболочки);
- прикладное:
  - инструментальные средства;
  - прикладные проблемные программы и системы.

Системное программное обеспечение (ПО)

Системные программы, составляющие системное программное обеспечение, предназначены для организации работы компьютера как системы в целом и для выполнения различных вспомогательных работ, например, копирования и архивирования нужной информации, выдачи справочной информации о компьютере, тестирования работоспособности устройств компьютера, лечения от заражения компьютерными вирусами и т. д.

Классификация системного ПО

К системным программам относятся:

- *ОПЕРАЦИОННЫЕ СИСТЕМЫ (ОС)* – основная часть системного программного обеспечения. ОС осуществляют взаимодействие между аппаратурой и пользователем, а также между аппаратурой и программами, позволяя отделить остальные классы программ от непосредственного взаимодействия с аппаратурой. ОС обеспечивают диалог с компьютером, управление компьютером, его ресурсами, запускают другие (прикладные) программы на выполнение. Они обеспечивают пользователю и прикладным программам удобный интерфейс (способы и средства взаимодействия) с устройствами компьютера.
- *СЕРВИСНЫЕ ПРОГРАММЫ* (оболочки и утилиты) составляют особый класс системных программ и предоставляют пользователям удобный сервис. Они не только делают наглядными часто используемые действия, но и предоставляют новые возможности для запускаемых программ.

Операционные системы (ОС)

Общие сведения

Операционная система в наибольшей степени определяет облик всей вычислительной системы в целом. Несмотря на это, пользователи, активно использующие вычислительную технику, зачастую испытывают затруднения при попытке дать определение операционной системе. Частично это связано с тем, что ОС выполняет две по существу мало связанные функции:

- повышение эффективности использования компьютера путем рационального **управления его ресурсами**;
- обеспечение пользователю удобств посредством предоставления для него **расширенной машины**.

ОС как система управления ресурсами

Ресурсом является любой компонент компьютера и предоставляемые им возможности: процессоры, память, таймеры, диски, накопители на магнитных лентах, сетевая коммуникационная аппаратура, принтеры и другие устройства. Функцией ОС как системы управления ресурсами является распределение процессоров, памяти, устройств и данных между процессами, конкурирующими за эти ресурсы. ОС должна управлять всеми ресурсами вычислительной машины таким образом, чтобы обеспечить максимальную эффективность ее функционирования.

ОС как расширенная машина

Операционная система скрывает от пользователя все детали работы с аппаратурой, предоставляя ему абстрактную, воображаемую (виртуальную) машину, с которой иметь дело гораздо проще и удобнее.

Например, для организации чтения блока данных с гибкого диска программист может использовать 16 различных команд, каждая из которых требует 13 параметров, таких, как номер блока на диске, номер сектора на дорожке и т. п. Когда выполнение операции с диском завершается, контроллер возвращает 23 значения, отражающих наличие и типы ошибок, которые, очевидно, надо анализировать. Даже если не входить в курс реальных проблем программирования ввода-вывода, ясно, что среди программистов нашлось бы не много желающих непосредственно заниматься программированием этих операций. При работе с диском программисту-пользователю достаточно представлять его в виде некоторого набора файлов, каждый из которых имеет имя. Работа с файлом заключается в его открытии, выполнении чтения или записи, а затем в закрытии файла. Вопро-

сы, подобные таким, как «следует ли при записи использовать усовершенствованную частотную модуляцию или в каком состоянии сейчас находится двигатель механизма перемещения считывающих головок?», не должны волновать пользователя. Подобные проблемы, связанные с работой реальной аппаратуры, называют низкоуровневыми (обработка прерываний, управление таймерами и оперативной памятью и др.).

Абстрактная, воображаемая машина, с которой, благодаря операционной системе, имеет дело пользователь, гораздо **проще и удобнее** в обращении, **чем реальная аппаратура**.

Интерфейс операционных систем

ОС обеспечивают пользователю и прикладным программам удобный **интерфейс** с устройствами компьютера.

Интерфейс – совокупность средств и правил, которые обеспечивают взаимодействие устройств, программ и человека.

Рассматривают несколько разновидностей интерфейса, например, программный интерфейс, пользовательский интерфейс. Пользовательский интерфейс обеспечивает взаимодействие человека с компьютером. Пользовательский интерфейс может быть символьным **(командным)** или графическим (**объектно-ориентированным**)**.**

Символьный (командный) интерфейс предполагает ввод пользователем команд с клавиатуры при работе с компьютером. Под командой традиционно понимают указание на выполнение некоторого действия. Именно таким был интерфейс старых операционных систем типа MS DOS, использовавших текстовый режим работы монитора.

Графический интерфейс предполагает управление работой компьютера путем выполнения операций над графическими образами (картинками), которые представляют **объекты** компьютерного мира (дисководы, программы, документы, файлы, папки и т. д.). Графический интерфейс по сравнению с символьным воспринимается как более понятный и интуитивно ясный. Именно такой интерфейс используется в операционных системах семейства Windows.

Классификация ОС

ОС принято классифицировать по следующим категориям:

- по количеству одновременно работающих пользователей: **однопользовательские** и **многопользовательские**;
- по числу одновременно выполняемых процессов: **однозадачные** и **многозадачные**;

- по количеству поддерживаемых процессоров: **однопроцессорные** и **многопроцессорные**;
- по разрядности кода ОС: 16-разрядные, 32-разрядные, 64-разрядные;
- по типу интерфейса: **командные** (текстовые) и **объектно-ориентированные** (графические);
- по типу доступа пользователя к компьютеру: **с пакетной обработкой**, с **разделением времени** и **реального времени**;
- по типу использования ресурсов: **сетевые** и **локальные**.

Различные ОС могут быть одновременно классифицированы по нескольким категориям. Например, можно говорить о локальной многозадачной ОС с графическим интерфейсом. Ниже будут рассмотрены примеры наиболее распространенных ОС.

Однозадачные ОС

Однозадачные ОС обладают следующими особенностями:

- диалог с компьютером осуществляется с помощью команд ОС, вводимых пользователем;
- доступная оперативная память имеет небольшой объем (640 Кбайт).

Операционная система MS-DOS

Операционная система (ОС) предложена корпорацией Microsoft. Первая версия операционной системы MS-DOS появилась в 1981 г. Затем были предложены новые версии ОС, каждая из которых значительно расширяла возможности предыдущей.

Многозадачные ОС с графическим интерфейсом пользователя

Большинство программ MS DOS рассчитано на работу в текстовом режиме. Психологи установили, что непосредственное зрительное восприятие, зрительные образы несут в себе гораздо больше информации для большинства людей, чем опосредованное (когда, например, зрительные образы рождаются в результате переработки услышанного или прочитанного).

В системах с графическим интерфейсом используется графический режим и сложная высокоуровневая графика. На этой основе разработан графический интерфейс пользователя, построенный на том принципе, что конкретное и видимое понятнее, чем абстрактное и невидимое. В системах с графическим интерфейсом для упрощения работы пользователя вместо обычных надписей на экране используются пиктограммы и многие другие элементы пользовательского интерфейса. Пик-

тограмма представляет собой картинку (символ), который служит для пользователя напоминанием о чем-либо. Пиктограммы обеспечивают конкретный, видимый символ, обозначающий команду, программу или данные. Делая обозначаемые объекты видимыми на экране, программа облегчает работу с ними. Обеспечив видимость на экране всех возможных для пользователя вариантов, программа уменьшает зависимость пользователя от того, насколько хорошо он запоминает информацию.

Многозадачность графических операционных систем означает способность поддерживать одновременное выполнение нескольких программ сразу. Например, можно слушать звуковой файл, рисовать картинку и печатать документ. Каждая из этих программ может иметь несколько параллельных потоков, т. е. независимо исполняемых частей. Различают **псевдомногозадачность** (кооперативная многозадачность), где переключение между задачами осуществляется пользователем. В настоящих многозадачных ОС реализуется так называемая **вытесняющая** многозадачность, когда переключение между задачами берет на себя ОС.

Из многозадачности вытекает такое общее свойство этих систем, как возможность **обмена данными** между одновременно работающими программами (приложениями).

Семейство Windows

Семейство WINDOWS является наиболее популярным для компьютеров IBM PC. Оно включает в себя:

Семейство WINDOWS 9x. Windows 95, Windows 98 и Windows ME.

Семейство Windows NT. Windows NT 4.0 (1996), Windows 2000, Windows XP , Windows Vista, Windows 7 , Windows 8

Семейство ОС Windows Mobile для карманных компьютеров. Windows CE, Windows Mobile

Хотя различные версии предоставляют различные возможности для рядовых и профессиональных пользователей, большая часть приводимых далее сведений в равной мере относится ко всему семейству.

Что такое Windows?

Windows представляет собой высокопроизводительную, многозадачную, 32-разрядную операционную систему с графическим интерфейсом и расширенными сетевыми возможностями.

В 32-разрядных ОС система организации памяти и управления памятью устроена иначе, чем в 16-разрядных. Не вдаваясь в подробности, можно лишь сказать, что программе доступна вся оперативная память компьютера, а использование виртуальной (дисковой) памяти при нехватке оперативной (например, при одновременной работе несколь-

ких программ) организовано оптимальнее. Это одна из главных причин более быстрой работы 32-разрядных программ.

Кроме того, в Windows реализована полноценная вытесняющая многозадачность и защита памяти, снижающая вероятность зависания всей системы из-за плохой работы какой-нибудь плохой программы.

Объектно-ориентированная платформа Windows

Благодаря тому, что операционная система Windows использует графический интерфейс, пользователь получил в руки достаточно удобную среду работы. Однако еще более важно, что основными понятиями операционной системы становятся объект, его свойства и действия, которые объект может выполнять в зависимости от запроса пользователя.

При намерении что-либо сделать в системной среде Windows, необходимо придерживаться следующей последовательности действий:

- выбрать (выделить) объект, т. е. щелкнуть левой кнопкой мыши по изображению этого объекта на экране;
- выбрать (например, при помощи контекстного меню) необходимое действие из совокупности действий, которые объект может выполнить.

В среде Windows существует множество объектов, с которыми придется работать пользователю, например, с объектами файловой системы, с объектами графического интерфейса и т. д. В дальнейшем вы познакомитесь с наиболее типичными представителями разных классов подобных объектов.

Знакомство с объектами Windows начнем с описания его базовых объектов – файлов, папок, приложений, документов.

Объекты файловой системы и ее организация

Файл, папка, ярлык и файловая структура

Под *файлом* обычно понимают именованный участок внешней памяти (например, на магнитном диске), на котором хранится любой набор данных (программы на языках, выполняемые программы в машинных кодах, данные и т. д.).

В среде Windows любой файл воспринимается как объект, имеющий уникальное имя. Файлу рекомендуется давать такое имя, которое отражает суть хранящейся в нем информации.

Над файлом можно выполнить определенный набор действий, которые переводят его из одного состояния в другое. Среди предписываемых файлу действий можно выделить некий набор стандартных действий, которые можно выполнять с файлом любого типа. В этот набор

входят следующие операции: *Открыть файл, Отправить файл, Копировать файл, Удалить файл с диска, Переименовать файл.*

Другим важным объектом Windows является *папка.* Папка Windows играет ту же роль, что и обычная папка для хранения документов в делопроизводстве: она позволяет упорядочить хранение документов. В среде Windows термин «папка» имеет широкое толкование – как хранилище объектов. Поэтому естественно говорить не «папка содержит информацию о местоположении файлов», а «папка содержит файлы». Помимо файлов папка может содержать и другие объекты (например, ярлыки). Так, папка может содержать файлы по курсовой работе или аннотации к книгам.

Над папками, как над объектами, можно выполнять стандартный набор действий, аналогичный тем, которые производятся с файлами: *создать папку, удалить папку, переименовать папку, скопировать папку* в другое место, *переместить папку* в другое место. Помимо них предусмотрены действия по открытию или закрытию папки. При открытии папки на экране появляется окно, в котором значками изображены содержащиеся в ней файлы. Закрытие папки означает закрытие этого окна.

Ярлык – еще один объект. Это ссылка на какой-либо объект, указывающая на его местоположение. Ярлык служит для ускорения запуска программ или документов. Объект и его ярлык обычно находятся в разных местах. Ярлык хранится в файле объемом 1 Кбайт. Его можно легко создать или уничтожить, что никак не влияет на связанный с ним объект. Изображается он значком с черной стрелкой в левом углу  .

Действия, которые можно совершать с ярлыком, аналогичны действиям над файлами. Открыть ярлык – означает открыть связанный с этим ярлыком объект.

Любая современная операционная система включает **файловую систему**, предназначенную для хранения данных на дисках и обеспечения доступа к ним. Принцип организации файловой системы – табличный. Данные о том, в каком месте диска записан тот или иной файл, хранятся в системной области диска в специальных *таблицах размещения файлов* (FAT-таблицах). Поскольку нарушение FAT-таблицы приводит к невозможности воспользоваться данными, записанными на диске, к ней предъявляются особые требования надежности, и она существует в двух экземплярах, идентичность которых регулярно контролируется средствами операционной системы. Файловая система обеспечивает возможность доступа к конкретному файлу по его имени и позволяет найти свободное место при записи нового файла. Она определяет *схему записи информации*, содержащейся в файлах, на физический диск.

Несмотря на то, что данные о местоположении файлов хранятся в табличной структуре, пользователю они представляются в виде **иерархической файловой структуры** – людям так удобнее, а все необходимые преобразования берет на себя операционная система. Это представление основано на следующих идеях:

- каждый файл размещается в папке;
- папка – это файл, элементы которого описывают другие файлы (а, возможно, и другие объекты) и обеспечивают доступ к ним по их именам;
- папка может находиться в другой папке вместе с обычными файлами. При этом образуется **иерархическая древовидная система вложенных папок**.

Самая внешняя папка, в которую вложены все остальные, называется корневой папкой.

Таким образом, **файловой системой** называют всю совокупность папок и файлов, хранимых на внешних носителях, а их логическую организацию для пользователя – **файловой структурой**.

Имена файлов и папок

Имя файла образуется из двух частей: собственно имени и типа файла, разделенных точкой. Имя может содержать от 1 до 255 букв, цифр и знаков, расположенных в произвольном порядке. Тип содержит от 1 до 3 букв или цифр (может и вообще отсутствовать). Тип приписывается файлу в соответствии с характером хранимой информации и задается либо самим пользователем, либо программой, создающей файл. Поскольку тип расширяет представление о файле, его часто называют расширением имени файла.

В корневой папке диска (на верхнем уровне иерархической файловой структуры) нежелательно хранить файлы с длинными именами − в отличие от прочих папок в ней ограничено количество единиц хранения, причем, чем длиннее имена, тем меньше файлов можно разместить в корневой папке.

Поскольку файлы могут храниться на разных устройствах (дисках), и внутри вложенных папок, то операционной системе необходимо указывать, где искать тот или иной файл. Для этого используется **полное имя файла**, включающее **имя устройства** и **полный маршрут (путь)** при поиске файла. Имя устройства – это буквенное обозначение дисковода с двоеточием, например, C:. Маршрут указывает местополо-

жение файла на внешнем носителе (диске). Маршрут − это цепочка из имен вложенных папок, разделенных знаком \. В цепочке каждая следующая папка вложена в предыдущую. На рис. 5 приведен пример, иллюстрирующий образование полного имени файла Rar.exe в папке Rar2.0 на диске C: .

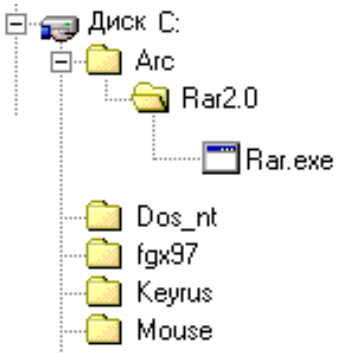


Рис. 5. Иерархия папок, соответствующая полному имени файла C:\Arc\Rar2.0\Rar.exe

Объекты пользовательского уровня – приложение и документ

Прикладные программы, которые создаются для определенной операционной системы, называются ее **приложениями**. Можно говорить о приложениях MS DOS, приложениях Windows и других операционных систем.

Многие приложения предназначены для создания информации определенного вида. Текстовый редактор используется для создания текстовой информации, графический редактор – для графической и т. д. Информация, созданная таким приложением и сохраненная в виде файла на носителе информации, называется **документом** этого приложения. В этом смысле можно говорить о документах текстового процессора MS Word или табличного процессора MS Excel, а также о документах других приложений.

Объекты графического интерфейса

Стандарт графического интерфейса пользователя

Производители программ для Windows договорились об официальном едином **стандарте пользовательского интерфейса**. Среда Windows поддерживает стандарт System Application Architecture (SAA). Использование этого стандарта упрощает работу пользователя с программами за счет подобия их интерфейса. Все программы обладают набором знакомых пользователю функций (например, работа с файлами, редактирование текста и управление окнами). Кроме того, в стандарте оговорен набор элементов пользовательского интерфейса, реализующих эти функции.

Далее перечислены основные стандартные элементы графического интерфейса пользователя.

Окно

Окно – это прямоугольная область экрана, выделяемая каждой программе для осуществления операций ввода-вывода. Положение и размеры окна можно менять. Окна могут быть перекрывающимися и заполняющими весь экран (такое окно называется развернутым). Окно может содержать в себе дочерние (подчиненные) окна, которые закрываются при закрытии основного окна. На рис. 6 обозначены основные элементы окна: рабочая область, заголовок, рамка, полосы прокрутки (горизонтальная и вертикальная). Важную роль играют кнопки управления окном: кнопка сворачивания окна на панель задач  (при щелчке по этой кнопке окно исчезает с экрана, но не закрывается), кнопка разворачивания окна  , кнопка его закрытия  .

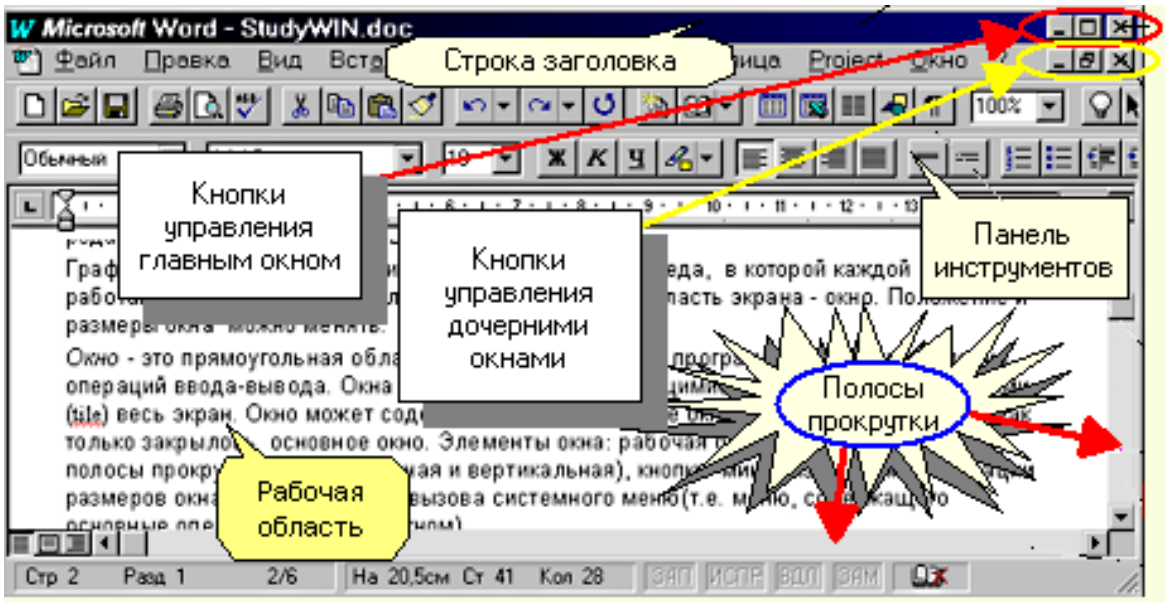


Рис. 6. Основные элементы окна Windows

Кнопка

Кнопка – это элемент управления прямоугольной формы, внешне напоминающий обычную кнопку или клавишу. На кнопку можно «нажать» при помощи мышки, используя для этого ее указатель. Нажатие на кнопку приводит к выполнению некоторых действий, программно связанных с кнопкой.

Панель диалога

Панель диалога (рис. 7) – специальный тип окна, используемый для отображения и ввода информации. Бывают модальными и немодальными. Модальная панель не позволяет программе работать до тех пор, пока не завершена работа с панелью.

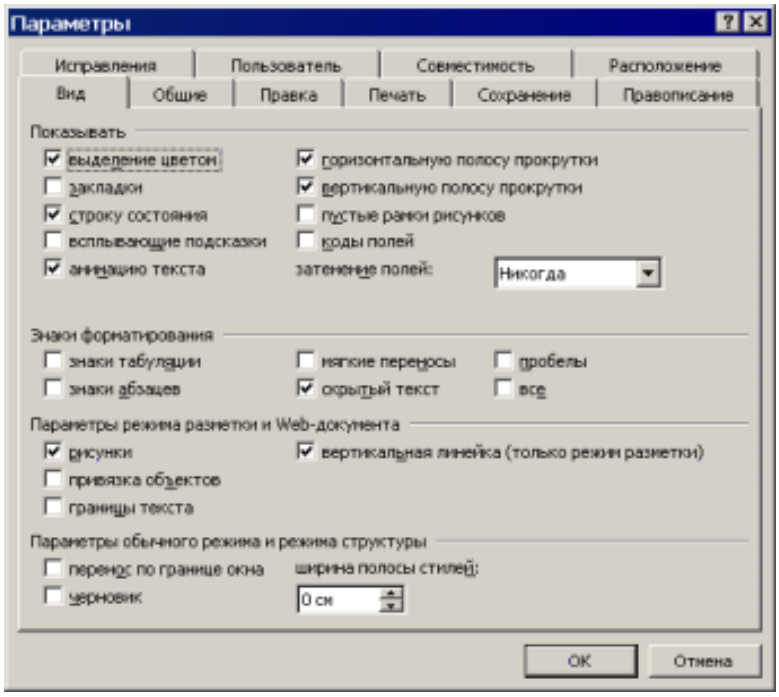


Рис. 7. Панель диалога

Панель сообщения

Панель сообщения (рис. 8) – частный случай панели диалога, предназначенный для вывода некоторых сообщений (например, об ошибке).

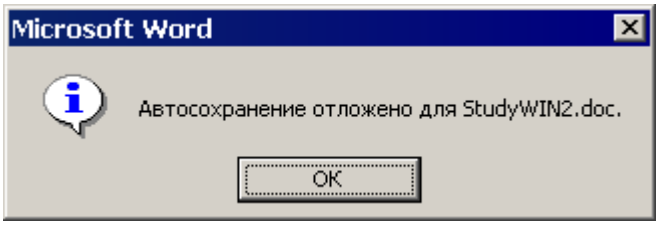


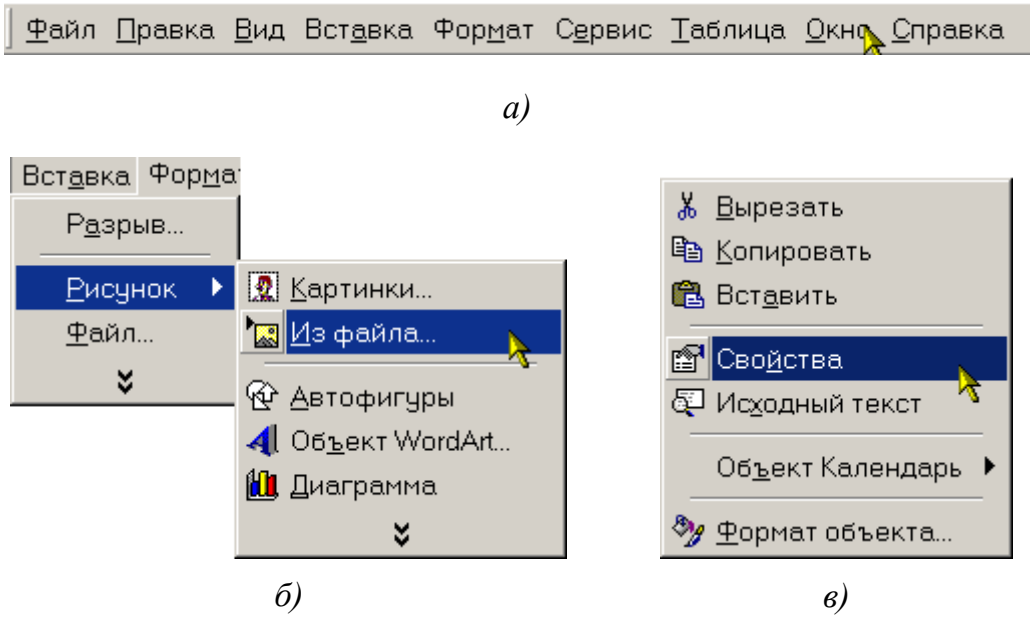
Рис. 8. Панель сообщения

Меню

Меню – совокупность команд, из которых можно делать выбор при помощи клавиатуры или мыши. Различают пять типов меню: системное, главное, выпадающее, вложенное, всплывающее (контекстное). На рис. 9 приведены примеры некоторых разновидностей меню.

Главное (горизонтальное) меню (рис. 9, *а*) – обязательный элемент интерфейса почти всех приложений (программ) Windows. Оно всегда расположено под строкой заголовка.

Из других разновидностей меню наибольший интерес представляет всплывающее (контекстное) меню (рис. 9, *в*). Появление такой разновидности меню напрямую связано с объектной ориентацией среды Windows. Это меню всплывает при щелчке правой кнопкой мыши на любом объекте Windows, а его содержание (команды меню) зависит от вида объекта, т. е. от его контекста, чем и вызвано его второе название. Для каждого объекта команды этого меню отражают главные свойства объекта и операции, которые можно выполнить над объектом.



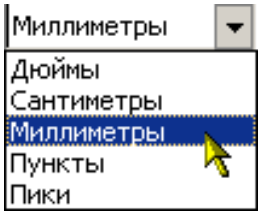
а)

б) *в)*

Рис. 9. Разновидности меню: а) главное меню; б) выпадающее и вложенное меню; в) контекстное (всплывающее) меню

Список

Список – это набор элементов, один или несколько из которых могут быть выбраны. Для раскрытия списка нужно щелкнуть по кнопке с треугольником.



Пиктограмма (значок)

Пиктограмма – это небольшое изображение, картинка (символ), который служит для пользователя напоминанием о чем-либо и используется для наглядного представления команды, программы или некоторых данных. В настоящее время вместо названия пиктограмма в русскоязычных версиях операционных систем и русскоязычной литературе чаще используется название значок.

Указатель мыши

Указатель мыши – это небольшое изображение, управляемое перемещением «мыши», которое указывает место выполнения очередного действия.

Объекты интерфейса пользователя семейства Windows

Начиная с Windows 95, в интерфейсе пользователя семейства Windows, помимо стандартных элементов графического интерфейса, появились новые объекты. Далее приведены наиболее важные из них.

Рабочий стол

Экран реализует образ рабочего стола пользователя (рис. 10), на котором может быть размещено в виде значков все, что нужно для каждодневной работы: программы, документы, папки, содержащие файлы и программы и другие объекты, все доступные диски. Чтобы открыть документ или запустить программу, нужно просто дважды щелкнуть мышкой

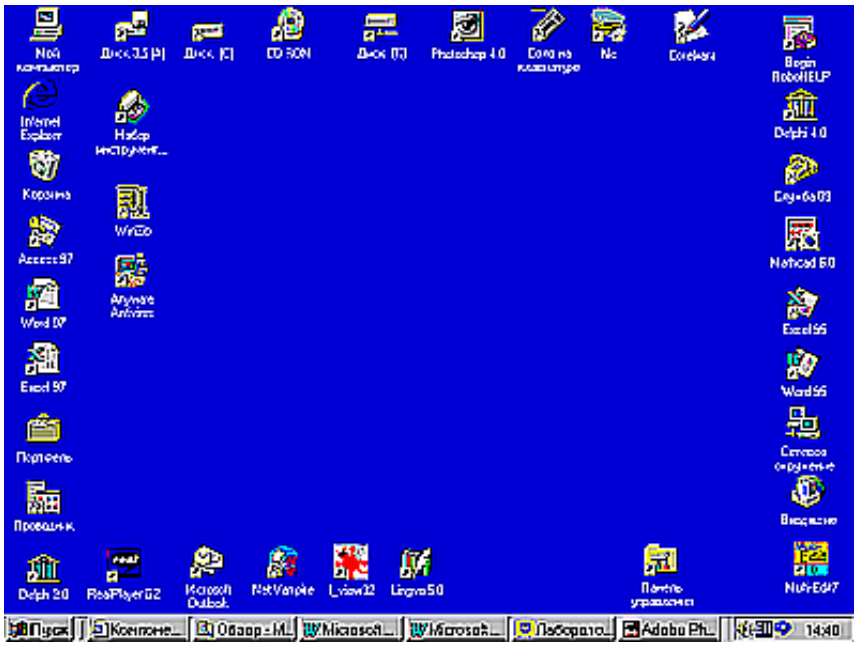


Рис. 10. Рабочий стол пользователя Windows

на значке. Папки можно вкладывать друг в друга. Рабочий стол имеет объектно-ориентированные свойства. Все, что расположено на его поверхности, является объектами, включая и сам рабочий стол. Если щелкнуть правой клавишей мышки на любом объекте, появится контекстное меню, показывающее все, что можно сделать с этим объектом. В зависимости от того, что представляет собой объект, обозначенный значком, его можно затем открыть, скопировать его содержимое или изменить любое его свойство из имеющегося набора.

Панель задач

В нижней части рабочего стола находится важный элемент интерфейса пользователя – Панель задач (рис. 11).

Рис. 11. Панель задач

У панели задач два основных назначения: *доступ к главному меню* Windows и *переключение между открытыми окнами*.

Доступ к главному меню. Слева на панели задач расположена кнопка Пуск. При щелчке по этой кнопке всплывает главное меню Windows (рис.12). Используя это меню можно быстро запустить любую программу, открыть документ, найти нужную программу, обратиться к справке Windows, завершить работу в Windows или перезагрузиться.

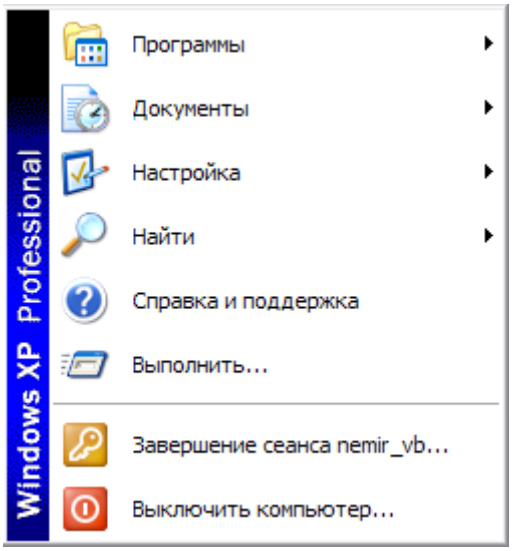


Рис. 12. Главное меню Windows

Переключение между окнами. При запуске любой программы открывается ее окно, а на панели задач появляется кнопка окна этой программы со значком программы и соответствующей надписью. Кноп-

ки можно использовать для перехода к любому окну, для переключения между окнами. Чтобы переключиться на окно конкретной программы, нужно лишь отыскать на панели задач кнопку этого окна по значку и соответствующей надписи и щелкнуть по ней. Окно сразу всплывет на передний план (говорят, станет активным). Кнопка активного окна, с которым Вы в данный момент работаете, изображается нажатой (вдавленной). Щелчок по кнопке активного окна (вдавленной) приводит к сворачиванию окна на панель задач.

Обращайте внимание на кнопки на панели задач!!! Здесь Вы увидите, не запустили ли Вы по ошибке повторные экземпляры какой-то программы.

Мой компьютер

Объект, который обеспечивает доступ к системным средствам, имеющимся в распоряжении пользователя: локальным и сетевым дискам, принтерам, панели управления. Двойной щелчок на значке объекта Мой компьютер открывает окно со значками всех доступных средств.

Корзина

Рабочий стол содержит объект **Корзина**, который можно использовать для удаления папок или файлов путем простого перетаскивания их в корзину. При таком удалении файлы сохраняются в корзине и их всегда можно восстановить.

Основные приемы управления в Windows

В Windows большую часть приемов управления работой компьютера можно выполнять с помощью мыши, воздействуя на графические объекты. С мышью связан активный элемент управления − *указатель мыши.* При перемещении мыши по плоской поверхности указатель перемещается по Рабочему столу, и его можно *позиционировать* на значках объектов или на пассивных элементах управления приложений.

Основными приемами управления с помощью мыши являются:

- *щелчок* (быстрое нажатие и отпускание левой кнопки мыши);
- *двойной щелчок* − два щелчка, выполненные с малым интервалом времени между ними**;**
- *щелчок правой кнопкой* (то же, что и *щелчок,* но с использованием правой кнопки);
- *перетаскивание* − выполняется путем перемещения мыши при нажатой левой кнопке (обычно сопровождается перемещением экранного объекта, на котором установлен указатель);

- *протягивание мыши* − выполняется, как и *перетаскивание,* но при этом происходит не перемещение экранного объекта, а изменение его формы;
- *специальное перетаскивание* − выполняется, как и *перетаскивание,* но при нажатой правой кнопке мыши, а не левой;
- *зависание* − наведение указателя мыши на значок объекта или на элемент управления и задержка его на некоторое время (при этом обычно на экране появляется *всплывающая подсказка,* кратко характеризующая свойства объекта).

Стандартные и служебные приложения Windows

Основное назначение операционных систем – обеспечение взаимодействия человека, оборудования и программ. От операционных систем не требуется наличия средств, предназначенных для исполнения конкретных прикладных задач, для этого есть прикладное программное обеспечение. Тем не менее, в операционную систему Windows входит ограниченный набор прикладных программ, с помощью которых можно решать некоторые простейшие повседневные задачи, пока на компьютере не установлены более мощные программные средства. Такие программы, входящие в поставку Windows, называют *стандартными приложениями.* В силу особой простоты их принято также рассматривать в качестве учебных. Знание приемов работы со стандартными приложениями позволяет ускорить освоение специализированных программных средств. Некоторые из стандартных приложений предназначены не для решения прикладных задач, а для выполнения служебных функций. Такие приложения называются служебными программами.

Проводник Windows

Проводник Windows − это приложение, реализующее графический интерфейс, обеспечивающий доступ пользователя к файлам и навигацию по файловой структуре в операционной системе Microsoft Windows. Но это лишь одна из функций проводника. Фактически Проводник является основой **графической оболочки пользователя** Windows. Всё, что видит пользователь после загрузки Windows (пиктограммы рабочего стола, панель задач, меню «Пуск» − кроме «обоев») − это Проводник Windows.

Иногда Проводником называют его часть, предназначенную для манипуляции файлами. Её можно вызвать двойным щелчком по иконке «Мой компьютер», запустить из «меню Пуск», воспользоваться клавиатурным сокращением Win + E .

За функционирование Проводника Windows отвечает процесс explorer.exe. Функции explorer.exe:

- Отображение папок и файлов, включая специальные папки вроде «Панель управления», «Планировщик задач», «Принтеры и факсы», «Шрифты» и т. п.
- Отображение графической оболочки Windows: панели задач с кнопкой «Пуск» и объектов рабочего стола.

Процесс explorer.exe не является критическим и может быть закрыт с помощью Диспетчера задач. После его закрытия исчезают элементы рабочего стола и панели задач, но фоновая картинка сохраняется. Процесс может быть перезапущен с помощью Диспетчера задач.

Проводник относится к классу программ, которые называют менеджерами файлов. Главная особенность пользовательского интерфейса таких программ – двухпанельная структура окна программы, обусловлена их функциями по навигации и управлению файлами.

Навигация по файловой структуре

Навигация – термин, которым мы будем обозначать совокупность приемов, обеспечивающих перемещение по файловой структуре компьютера. Окно Проводника (см. рис. 13) имеет две рабочие области: левую панель (панель папок) и правую панель (панель содержимого папки, выделенной в левой панели). Для навигации по файловой структуре используется левая панель Проводника. Эта панель отображает в виде древовидной структуры все доступные пользователю диски и папки (в том числе и сетевые).

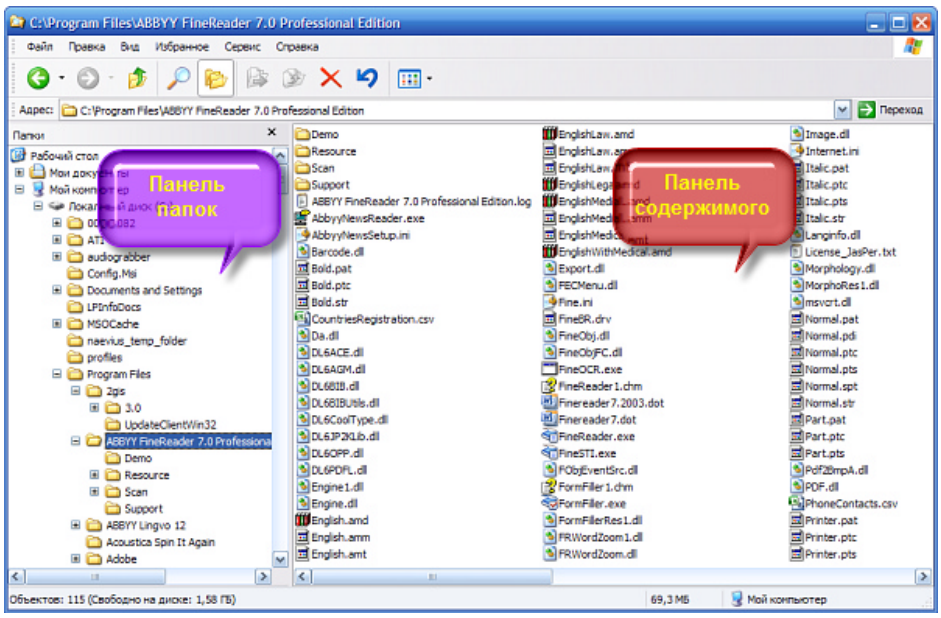


Рис. 13. Окно проводника Windows

Папки в Проводнике можно сворачивать, разворачивать и открывать (операции навигации). Открыть папку – значит отобразить ее содержимое на правой панели Проводника. Развернуть папку – значит отобразить ее внутреннюю структуру в левой панели Проводника. Развернутую папку можно, наоборот, свернуть.

Чтобы развернуть любую папку, в которой есть вложенные папки, нужно щелкнуть по значку + слева от значка папки в левой панели проводника. В левой панели сразу отобразятся все вложенные папки, а значок + рядом с папкой изменится на значок –, показывая, что структура папки раскрыта. При этом содержимое правого окна Проводника не изменится.

До щелчка После щелчка

Этот прием можно использовать, чтобы добраться до папки глубокого уровня вложенности, не отображая в правом окне содержимое промежуточных папок.

Чтобы открыть любую вложенную папку (отобразить ее содержимое в правом окне Проводника), нужно щелкнуть на самом значке такой папки.

Управление файлами

Под управлением файлами будем понимать уже упомянутые ранее стандартные операции над файлами: *Открыть файл, Отправить файл, Копировать файл, Удалить файл с диска, Переименовать файл.*

Важно отметить, что двухпанельная структура окна Проводника позволяет легко переместить или скопировать любой файл из одного диска или папки в другой, не открывая нескольких окон. Это особенно удобно при копировании и перемещении данных с использованием приема *перетаскивания*. (Для копирования необходимо дополнительно удерживать клавишу *Ctrl* на клавиатуре**).**

Управление файлами с помощью меню

Описанные ниже операции можно выполнить, используя главное или контекстное меню Проводника.

1. *Открыть файл*. Результат этого действия будет зависеть от типа файла. Так, если файл хранит документ, то при его открытии вместе с ним будет загружаться и программная среда, где создавался этот документ. Например, открывая файл с рисунком, созданным в графиче-

ском редакторе, на экране можно увидеть интерфейс этого редактора и находящийся на его рабочем поле рисунок. Если же файл является главным файлом (файлом запуска) некоей программной системы, то одноименная команда служит сигналом для её запуска и на экране появляется интерфейс этой среды с пустым рабочим полем. Для любого другого файла система предложит перечень программ, с помощью которых можно попытаться открыть данный файл.

Примечание. Открыть файл можно, не прибегая к помощи контекстного меню. Для этого надо установить указатель мыши на значок файла и выполнить двойной щелчок мышью.

2. *Отправить файл*. В результате этого действия файл либо отправляется по факсу или по электронной почте, либо перемещается в папку Мои документы или на гибкий диск.

3. *Переместить файл*. Перемещение файла в другое место осуществляется в два приема: сначала файл вырезается по команде Вырезать. При этом файл перемещается в буфер обмена – специальную область памяти. Затем с помощью указателя мыши выбирается место, куда следует переместить файл, и в контекстном меню выбирается команда Вставить. При этом файл из буфера вставляется в месте вставки.

Примечание. Буфер обмена – область оперативной памяти, которая выделяется операционной системой специально для временного хранения переносимого, копируемого или удаляемого объекта.

4. *Копировать файл*, т. е. создать копию файла. По этой команде создается в буфере обмена копия выбранного файла, а затем по команде Вставить вы можете многократно вставлять этот файл в места, указанные указателем мыши.

5. *Удалить файл* с диска. Удаление файла с диска может выполняться как на логическом, так и на физическом уровне. По команде Отправить файл в корзину файл удаляется в специально отведенную папку, которая называется Корзина и которая всегда находится на экране монитора. Файл, отправленный в Корзину, можно восстановить на исходном месте, достав его из Корзины. По команде Удалить файл он физически удаляется с диска. Эта команда выполняется для файлов Корзины.

6. *Переименовать файл*, т. е. изменить его имя.

7. *Создать папку*. Для создания новой папки нужно сначала открыть папку, в которой необходимо поместить новую папку (т. е. отобразить ее содержимое в правой панели Проводника). После этого нужно выполнить команду меню Файл/Создать/Папку, а затем ввести имя новой папки.

Программа Блокнот

Блокнот – это простейший текстовый редактор, который можно использовать в качестве удобного средства просмотра текстовых файлов (формат .txt и некоторые другие). Для создания текстовых документов его применяют редко (только для небольших записок). Программу можно запустить на выполнение с помощью кнопки Пуск (**Пуск/Программы/Стандартные/Блокнот)**.

Графический редактор Paint

Графическими называют редакторы, предназначенные для создания и редактирования изображений (рисунков). Программа Paint – простейший графический редактор. По своим возможностям она не соответствует современным требованиям, но в силу простоты и доступности остается необходимым компонентом операционной системы. Не разобравшись с принципами управления этой программой, трудно осваивать другие, более мощные средства работы с графикой. Программа запускается командой главного меню **Пуск/Программы/ Стандартные/Paint**.

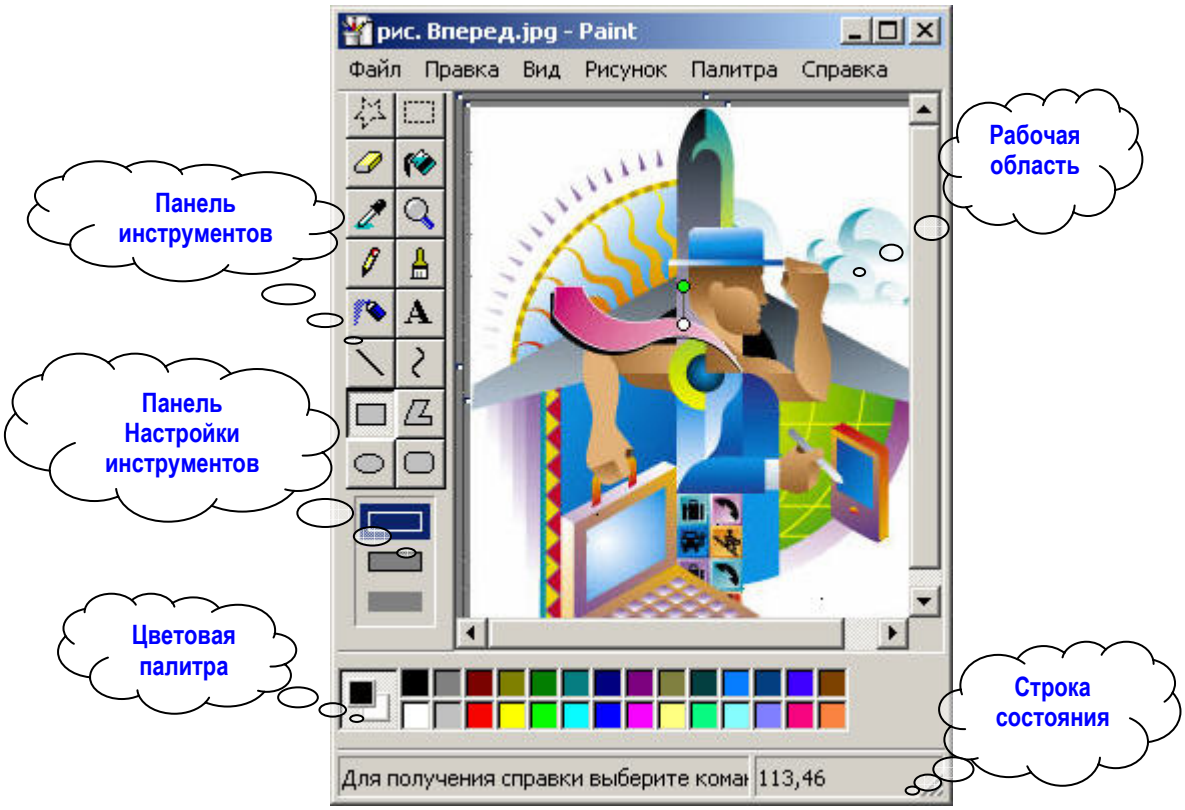


Рис. 14 . Окно графического редактора Paint

Сервисные системные программы

Программа Paint (рис. 14) является редактором растровой графики. Это важное замечание, поскольку кроме редакторов растровой графики существуют еще редакторы *векторной* графики. Приемы и методы работы с этими двумя классами программ различны. В растровой графике элементом изображения является точка, которой на экране соответствует экранная точка *(пиксел)*. Элементом векторной графики является линия, описываемая математическим выражением.

Среди различных задач, с которыми часто приходится сталкиваться пользователю компьютера, можно выделить сжатие информации (для экономии важного и дорогостоящего ресурса – памяти) и обеспечение компьютерной безопасности. Эти задачи решаются с помощью программ, относящихся к категории сервисного системного обеспечения. Краткие сведения о некоторых из них приведены ниже.

Работа со сжатыми данными

Хранение и передача информации по каналам связи стоят достаточно дорого. В связи с этим регулярно возникает необходимость сжимать данные перед тем, как размещать их в архивах или передавать по канала связи. Соответственно, существует и обратная необходимость восстановления данных из предварительно уплотненных архивов.

Теоретические основы сжатия данных

Характерной особенностью большинства «классических» типов данных, с которыми традиционно работают люди, является определенная избыточность. Степень избыточности зависит от типа данных. Например, у видеоданных степень избыточности обычно в несколько раз больше, чем у графических данных, а степень избыточности графических данных в несколько раз больше, чем текстовых.

Для человека избыточность информации нередко связана с представлением о ее качестве, поскольку избыточность, как правило, улучшает восприятие, особенно в неблагоприятных условиях (просмотр телепередач при наличии помех, восстановление поврежденного графического материала, чтение текстов в условиях недостаточной освещенности и т. п.).

Однако, когда речь заходит не об обработке, а о хранении готовых документов или их передаче, то избыточность можно уменьшить, что дает эффект сжатия данных.

Термин *сжатие данных* часто подменяют термином *архивация данных*, поскольку методы сжатия изначально стали применять для долговременного хранения готовых документов. *Архиваторы* – это про-

граммы для создания архивов. Архивы предназначены для хранения данных в удобном компактном виде. В качестве данных обычно выступают файлы и папки. Как правило, данные предварительно подвергаются процедуре сжатия или упаковки. Поэтому почти каждый архиватор одновременно является программой для сжатия данных. С другой стороны, любая программа для сжатия данных может рассматриваться как архиватор. Эффективность сжатия является важнейшей характеристикой архиваторов. От нее зависит размер создаваемых архивов. Чем меньше архив, тем меньше места требуется для его хранения. Для передачи нужна меньшая пропускная способность канала передачи или затрачивается меньшее время. Преимущества архивов очевидны, если учесть, что данные уменьшаются в размере и в 2 раза, и в 5 раз.

Если при сжатии данных происходит только изменение их структуры, метод сжатия обратим. Из результирующего кода можно восстановить исходный код путем применения обратного метода. Обратимые методы применяют для любых типов данных. Характерными форматами сжатия без потери информации являются:

- .GIF**,** .TIF, .PCX и многие другие для графических данных;
- .AVI для видеоданных;
- .ZIP, ARJ, .RAR, .LZH, .LH, .CAB и многие другие для любых типов данных.

Методы сжатия

Разработано большое количество разнообразных методов, их модификаций и подвидов для сжатия данных. Современные архиваторы, как правило, одновременно используют несколько методов одновременно. Можно выделить некоторые основные.

Кодирование длин серий (RLE – сокращение от run-length encoding – кодирование длин серий)

Очень простой метод. Последовательная серия одинаковых элементов данных заменяется на два символа: элемент и число его повторений. Широко используется как дополнительный, так и промежуточный метод. В качестве самостоятельного метода применяется, например, в графическом формате BMP.

Словарный метод (LZ – сокращение от Lempel Ziv – имена авторов).

Наиболее распространенный метод. Используется словарь, состоящий из последовательностей данных или слов. При сжатии эти слова заменяются их кодами из словаря.

Энтропийный метод (Huffman – кодирование Хаффмена, Arithmetic coding − арифметическое кодирование).

В этом методе элементы данных, которые встречаются чаще, кодируются при сжатии более коротким кодом, а более редкие элементы дан-

ных кодируются более длинным кодом. За счет того, что коротких кодов значительно больше, общий размер получается меньше исходного.

Широко используется как дополнительный метод. В качестве самостоятельного метода применяется, например, в графическом формате JPG.

Методов сжатия довольно много. Каждый метод обычно ориентирован на один вид или группу реальных данных. Хорошие результаты показывает комплексное использование методов, на котором построено большинство современных архиваторов.

Степень сжатия в основном зависит от исходных данных. Хорошо сжимаются почти все предварительно несжатые данные, например, исполняемые файлы (EXE), тексты (TXT, DOC), базы данных (DBF), простые несжатые изображения (BMP). Ограниченно сжимаются несжатый звук (WAV), сложные несжатые изображения (BMP). Не сжимаются почти все уже сжатые данные, например, архивы (ZIP, CAB), сжатые документы (PDF), сжатая графика и видео (JPG, GIF, AVI, MPG), сжатый звук (MP3). Их сжатие находится в пределах пары процентов за счет служебных блоков и небольшой избыточности.

Программные средства сжатия данных

Следует различать собственно программу-архиватор, формат архивов и методы сжатия. Даже один и тот же метод сжатия может иметь варианты реализации. Например, существует более десятка программ-архиваторов, которые могут создавать архивы в формате ZIP. В свою очередь данные в формате ZIP могут быть сжаты различными методами: Deflate, Deflate64, BZip2. Метод Deflate имеет несколько реализаций с разной скоростью и степенью сжатия (разница порядка 5 %). С помощью этого метода архиватор 7-zip позволяет создавать архивы в формате ZIP и 7Z.

Обычно архиваторы могут создавать архивы в собственном эксклюзивном формате с использованием своих оригинальных методов. Например, архиватор RAR позволяет создавать архивы RAR. В формате архива и методах сжатия заключаются основные преимущества того или иного архиватора.

В простейшем случае архиватор позволяет только упаковать или распаковать один файл. Кроме собственно сжатия данных, современные архиваторы обеспечивают некоторые дополнительные функции. Можно выделить несколько основных:

- сжатие некоторых файлов и целых директорий;
- создание самораспаковывающихся (SFX) архивов. То есть для распаковки архива программа-архиватор не требуется;
- изменение содержимого архива;
- шифрование содержимого архива;

- информация для восстановления архива при частичном повреждении и возможность восстановления поврежденных архивов;
- разбивка архива на несколько частей или томов;
- консольная версия программы для работы из командной строки;
- графическая (GUI) версия программы.

Кроме различий в функциональности, можно разбить архиваторы на две группы: асимметричные и симметричные. Асимметричные архиваторы требуют для операции распаковки значительно меньше времени и оперативной памяти, чем для операции упаковки. Это позволяет быстро получать содержимое архива на маломощных компьютерах. Симметричные архиваторы требуют для операций упаковки и распаковки одинаковое время и объем оперативной памяти. Использование таких архиваторов на широком парке компьютеров или для оперативного доступа к содержимому архива ограничено. Известный архиватор RAR в качестве основного использует асимметричный словарный метод сжатия, а для текстов может использовать симметричный PPM-метод. Таким образом, распаковка архивов RAR, сжатых с максимальной степенью сжатия, может быть невозможна на компьютерах с ограниченным объемом оперативной памяти. Все или почти все передовые архиваторы с высокой степенью сжатия являются симметричными.

Безусловно, самым распространенным архиватором форматом сжатия данных является ZIP и его модификации. По своей распространенности он значительно превосходит ближайших конкурентов. Следом идет RAR. В последние годы встречается архиватор 7-zip.

Наиболее распространенным во всем мире архиватором является WinZip, поскольку формат .zip принят как стандартный формат для передачи упакованных (сжатых) файлов в Интернет. Он обладает очень высокой скоростью упаковки и распаковки. Неплохо подходит для оперативного архивирования, но только в режиме ZIP fast. В России большой популярностью пользуется архиватор WinRar, превосходящий WinZip и по степени сжатия и по скорости. Как универсальный архиватор с хорошей степенью сжатия (на 30 % хуже лучших архиваторов) можно рекомендовать 7-zip.

Базовые требования к диспетчерам архивов

Современные программные средства для создания и обслуживания архивов отличаются большим объемом функциональных возможностей, многие из которых выходят далеко за рамки простого сжатия данных и эффективно дополняют стандартные средства операционной системы. В этом смысле современные средства архивации данных называют *диспетчерами архивов.*

К базовым функциям, которые выполняют большинство современных диспетчеров архивов, относятся:

- извлечение файлов из архивов;
- создание новых архивов;
- добавление файлов в имеющийся архив;
- создание самораспаковывающихся архивов;
- создание распределенных архивов на носителях малой емкости;
- тестирование целостности структуры архивов;
- полное или частичное восстановление поврежденных архивов;
- защита архивов от просмотра и несанкционированной модификации.

Самораспаковывающиеся архивы. В тех случаях, когда архивация производится для передачи документа потребителю, следует предусмотреть наличие у него программного средства, необходимого для извлечения исходных данных из уплотненного архива. Если таких средств у потребителя нет или нет оснований предполагать их наличие, создают самораспаковывающиеся архивы. Самораспаковывающийся архив готовится на базе обычного архива путем присоединения к нему небольшого программного модуля. Сам архив получает расширение имени .EXE, характерное для исполнимых файлов. Потребитель сможет выполнить его запуск как программы, после чего распаковка архива произойдет на его компьютере автоматически.

Распределенные архивы. В тех случаях, когда предполагается передача большого архива на носителях малой емкости, например, на гибких дисках, возможно распределение одного архива в виде малых фрагментов на нескольких носителях.

Некоторые диспетчеры (например, WinZip) выполняют разбиение сразу на гибкие диски, а некоторые (например, WinRar) позволяют выполнить предварительное разбиение архива на фрагменты заданного размера на жестком диске. Впоследствии их можно перенести на внешние носители путем копирования.

Гибкие диски являются крайне ненадежными носителями, поэтому архивы, сформированные на жестком диске, должны храниться до получения подтверждения от потребителя о том, что распределенный архив поступил к нему в неповрежденном виде и прошел распаковку. Правилом «хорошего тона» считается создание двух копий при передаче материалов на гибких дисках.

Защита архивов. В большинстве случаев защиту архивов выполняют с помощью пароля, который запрашивается при попытке просмотреть, распаковать или изменить архив. Теоретически, защита с помощью пароля считается неудовлетворительной и не рекомендуется для

особо важной информации. В то же время необходимо отметить, что основные программные средства, используемые для восстановления утраченного пароля (или взлома закрытой информации, что, по сути, то же самое), используют методы прямого перебора. Работу этих средств можно существенно затруднить и замедлить, если расширить область перебора. Пароли на базе символов английского алфавита и цифр действительно снимаются очень быстро. Однако даже незначительное увеличение числа используемых символов за счет знаков препинания многократно увеличивает криптостойкость защиты, а использование также и символов русского алфавита может существенно затруднить попытки снять пароль путем перебора, сделав сроки работы неприемлемыми.

Дополнительные требования к диспетчерам архивов

К дополнительным функциям диспетчеров архивов относятся сервисные функции, делающие работу более удобной. Они часто реализуются внешним подключением дополнительных служебных программ и обеспечивают:

- просмотр файлов различных форматов без извлечения их из архива;
- поиск файлов и данных внутри архивов;
- установку программ из архивов без предварительной распаковки;
- проверку отсутствия компьютерных вирусов в архиве до его распаковки;
- криптографическую защиту архивной информации;
- декодирование сообщений электронной почты;
- «прозрачное» уплотнение исполнимых файлов .EXE и .DLL
- создание самораспаковывающихся многотомных архивов;
- выбор или настройку коэффициента сжатия информации.

На рис. 15 изображено окно архиватора WinZip со списком файлов открытого архива.

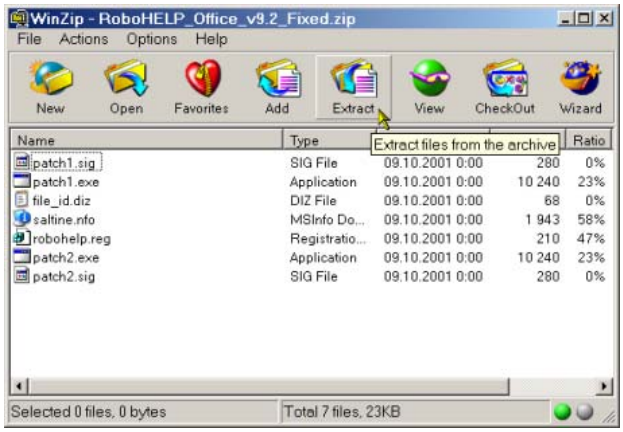


Рис. 15. Окно архиватора WinZip со списком файлов архива

Работа с интегрированными диспетчерами архивов

Во время установки WinZip или WinRar можно выбрать возможность интеграции этих архиваторов в программу Проводник. Такая интеграция дает возможность работать с архивами прямо в Проводнике с помощью контекстного меню (рис. 16). Например, чтобы создать архив из группы файлов, необходимо в правой панели Проводника выделить нужную группу файлов, щелкнуть на выделенной области правой кнопкой мышки и в появившемся меню выбрать команду **Add to Zip** для WinZip или **Добавить в архив…** для WinRar. В результате откроется окно диалога создания архива (рис. 17), в который нужно ввести необходимые параметры (например, имя архива и др.).

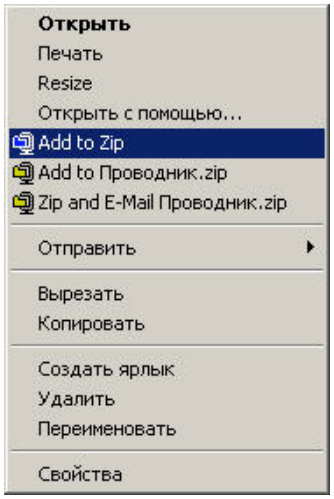


Рис. 16. Контекстное меню с интегрированными командами работы с WinZip

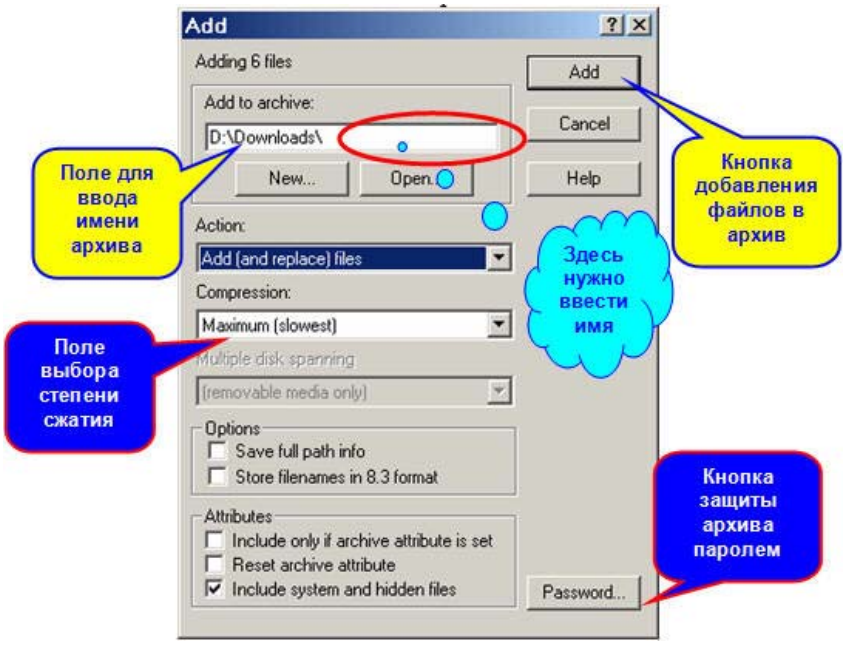


Рис. 17 . Окно диалога создания архива

При щелчке правой кнопкой мышки на файле архива в правой панели Проводника в меню появятся команды, предлагающие извлечь файлы из архива или создать из него самораспаковывающийся архив.

Вопросы компьютерной безопасности

Понятие о компьютерной безопасности

В вычислительной технике понятие безопасности является весьма широким. Оно подразумевает и надежность работы компьютера, и сохранность ценных данных, и защиту информации от внесения в нее изменений неуполномоченными лицами, и сохранение тайны переписки в электронной связи. Отдельным вопросам компьютерной безопасности посвящаются целые книги. Здесь мы рассмотрим лишь вопросы безопасности, связанные с защитой от компьютерных вирусов.

Компьютерные вирусы

Компьютерный вирус – это программный код, встроенный в другую программу, или в документ, или в определенные области носителя данных, и предназначенный для выполнения несанкционированных действий на несущем компьютере. Одна из характерных особенностей такого кода – способность многократно копировать самого себя («размножаться»).

Основными типами компьютерных вирусов являются:

- программные вирусы;
- загрузочные вирусы;
- макровирусы.

К компьютерным вирусам примыкают и так называемые *троянские кони (троянские программы, троянцы)*.

Программные вирусы. Программные вирусы − это блоки программного кода, целенаправленно внедренные внутрь других прикладных программ. При запуске программы, несущей вирус, происходит запуск имплантированного в нее вирусного кода. Работа этого кода вызывает скрытые от пользователя изменения в файловой системе жестких дисков и/или в содержании других программ. Так, например, вирусный код может воспроизводить себя в теле других программ − этот процесс называется *размножением.* По прошествии определенного времени, создав достаточное количество копий, программный вирус может перейти к разрушительным действиям − нарушению работы программ и операционной системы, удалению информации, хранящейся на жестком диске. Этот процесс называется *вирусной атакой.*

Самые разрушительные вирусы могут инициировать форматирование жестких дисков. Поскольку форматирование диска − достаточно продолжительный процесс, который не должен пройти незамеченным со стороны пользователя, во многих случаях программные вирусы ограничиваются уничтожением данных только в системных секторах жесткого диска, что эквивалентно потере таблиц файловой структуры. В этом случае данные на жестком диске остаются нетронутыми, но воспользоваться ими без применения специальных средств нельзя, поскольку неизвестно, какие сектора диска каким файлам принадлежит. Теоретически восстановить данные в этом случае можно, но трудоемкость этих работ исключительно высока.

Считается, что никакой вирус не в состоянии вывести из строя аппаратное обеспечение компьютера. Однако бывают случаи, когда аппаратное и программное обеспечение настолько взаимосвязаны, что программные повреждения приходится устранять заменой аппаратных средств. Так, например, в большинстве современных материнских плат базовая система ввода-вывода (BIOS) хранится в перезаписываемых постоянных запоминающих устройствах (так называемая *флэш-память).* Возможность перезаписи информации в микросхеме флэш-памяти используют некоторые программные вирусы для уничтожения данных BIOS. В этом случае для восстановления работоспособности компьютера требуется либо замена микросхемы, хранящей BIOS*,* либо ее перепрограммирование на специальных устройствах, называемых *программаторами.*

Программные вирусы поступают на компьютер при запуске непроверенных программ, полученных на внешнем носителе (гибкий диск, компакт-диск и т. п.) или принятых из Интернета. Особое внимание следует обратить на слова «*при запуске».* При обычном копировании зараженных файлов заражение компьютера произойти не может. В связи с этим все данные, принятые из Интернета, должны проходить обязательную проверку на безопасность, а если получены незатребованные данные из незнакомого источника, их следует уничтожать, не рассматривая. Обычный прием распространения «троянских» программ − приложение к электронному письму с «рекомендацией» извлечь и запустить якобы полезную программу.

Загрузочные вирусы. От программных вирусов загрузочные вирусы отличаются методом распространения. Они поражают не программные файлы, а определенные системные области магнитных носителей (гибких и жестких дисков). Кроме того, на включенном компьютере они могут временно располагаться в оперативной памяти.

Обычно заражение происходит при попытке загрузки компьютера с магнитного носителя, системная область которого содержит загрузоч-

ный вирус. Так, например, при попытке загрузить компьютер с гибкого диска происходит сначала проникновение вируса в оперативную память, а затем в загрузочный сектор жестких дисков. Далее этот компьютер сам становится источником распространения загрузочного вируса.

Макровирусы. Эта особая разновидность вирусов поражает документы, выполненные в некоторых прикладных программах, имеющих средства для исполнения так называемых *макрокоманд*. В частности, к таким документам относятся документы текстового процессора Microsoft Word (они имеют расширение .doc). Заражение происходит при открытии файла документа в окне программы, если в ней не отключена возможность исполнения макрокоманд. Как и для других типов вирусов, результат атаки может быть как относительно безобидным, так и разрушительным.

Методы защиты от компьютерных вирусов

Существуют три рубежа защиты от компьютерных вирусов:

- предотвращение поступления вирусов;
- предотвращение вирусной атаки, если вирус все-таки поступил на компьютер;
- предотвращение разрушительных последствий, если атака все-таки произошла.

Существуют три метода реализации защиты:

- программные методы защиты;
- аппаратные методы защиты;
- организационные методы защиты.

В вопросе защиты ценных данных важно не столько предотвратить их разрушение компьютерными вирусами, сколько обеспечить возможность их восстановления, если такое случится. Создав бастионы на пути проникновения вирусов в компьютер, нельзя положиться на их прочность и остаться неготовым к действиям после разрушительной атаки. К тому же, вирусная атака – далеко не единственная и даже не самая распространенная причина утраты важных данных. Существуют программные сбои, которые могут вывести из строя операционную систему, а также аппаратные сбои, способные сделать жесткий диск неработоспособным. Всегда существует вероятность утраты компьютера вместе с ценными данными в результате кражи, пожара или иного стихийного бедствия.

Поэтому создавать систему безопасности следует в первую очередь «с конца» – с предотвращения разрушительных последствий любого воздействия, будь то вирусная атака, кража в помещении или физический выход жесткого диска из строя. Надежная и безопасная работа с

данными достигается только тогда, когда любое неожиданное событие, в том числе и полное физическое уничтожение компьютера не приведет к катастрофическим последствиям.

Средства антивирусной защиты

Основным средством защиты информации является резервное копирование наиболее ценных данных.

В случае утраты информации по любой из вышеперечисленных причин жесткие диски переформатируют и подготавливают к новой эксплуатации. На «чистый» отформатированный диск устанавливают операционную систему с дистрибутивного компакт-диска, затем под ее управлением устанавливают все необходимое программное обеспечение, которое тоже берут с дистрибутивных носителей. Восстановление компьютера завершается восстановлением данных, которые берут с резервных носителей.

При резервировании данных следует также иметь в виду и то, что надо отдельно сохранять все регистрационные и парольные данные для доступа к сетевым службам Интернета. Их не следует хранить на компьютере. Обычное место хранения − служебный дневник в недоступном месте (например, в сейфе руководителя подразделения).

Создавая план мероприятий по резервному копированию информации, необходимо учитывать, что резервные копии должны храниться отдельно от компьютера. То есть, например, резервирование информации на отдельном жестком диске того же компьютера только создает иллюзию безопасности. Резервные копии конфиденциальных данных сохраняют на внешних носителях, которые хранят в сейфах, желательно в отдельных помещениях. При разработке организационного плана резервного копирования учитывают необходимость создания не менее двух резервных копий, сохраняемых в разных местах. Между копиями осуществляют ротацию. Например, в течение недели ежедневно копируют данные на носители резервного комплекта А, а через неделю их заменяют комплектом Б, и т. д.

Вспомогательными средствами защиты информации являются антивирусные программы и средства аппаратной защиты. Так, например, простое отключение перемычки на материнской плате не позволит осуществить стирание перепрограммируемой микросхемы ПЗУ *(флэш-BIOS),* независимо от того, кто будет пытаться это сделать: компьютерный вирус, злоумышленник или неаккуратный пользователь.

Существует достаточно много программных средств антивирусной защиты. Они предоставляют следующие возможности.

1. Регулярное сканирование жестких дисков в поисках компьютерных вирусов. Сканирование обычно выполняется автоматически при каждом включении компьютера и при размещении внешнего диска в считывающем устройстве. При сканировании следует иметь в виду, что антивирусная программа ищет вирус путем сравнения кода программ с кодами известных ей вирусов, хранящимися в базе данных. Если база данных устарела, а вирус является новым, сканирующая программа его не обнаружит. **Для надежной работы следует регулярно обновлять антивирусную программу.** Желательная периодичность обновления − один раз в неделю; допустимая − один раз в месяц. Для примера укажем, что разрушительные последствия атаки вируса «Чернобыль», вызвавшего уничтожение информации на сотнях тысяч компьютеров 26 апреля 1999 года, были связаны не с отсутствием средств защиты от него, а с длительной задержкой (более года) в обновлении этих средств.

2. Контроль изменения размеров и других атрибутов файлов. Поскольку некоторые компьютерные вирусы на этапе размножения изменяют параметры зараженных файлов, контролирующая программа может обнаружить их деятельность и предупредить пользователя.

3. Контроль обращений к жесткому диску. Поскольку наиболее опасные операции, связанные с работой компьютерных вирусов, так или иначе, обращены на модификацию данных, записанных на жестком диске, антивирусные программы могут контролировать обращения к нему и предупреждать пользователя о подозрительной активности.

Наряду с этими возможностями некоторые антивирусные средства предоставляют дополнительные возможности, например, создание образа жесткого диска на внешних носителях. Такая мера позволяет восстановить данные в системных областях жесткого диска в случае их разрушения.

Существует значительное количество мощных бесплатных и коммерческих антивирусных средств. Наиболее популярными из них являются Norton Antivirus компании Symantec и российский Антивирус Касперского AVP. Как правило, антивирусные средства такого класса представляют собой совокупность программных модулей, предназначенных для решения различных задач. В число модулей входят:

- сканер (запускается пользователем, сканирует оперативную память, системные области жесткого диска, а также указанные пользователем диски, папки или отдельные файлы);
- монитор или страж (резидентная программа, которая запускается автоматически в конце загрузки операционной системы, работает в фоновом режиме, отслеживая все контакты с «внешним миром», такие как вставка гибкого диска в приемное устройство, соединение с сетевыми дисками и т. п.);

- программа автоматического обновления антивирусной базы данных через Интернет или из указанной папки;
- центр управления или планировщик задач, который позволяет планировать и назначать регулярное выполнение специальных задач, например, ежедневное полное сканирование жесткого диска в конце рабочего дня или обновление базы данных один раз в неделю.

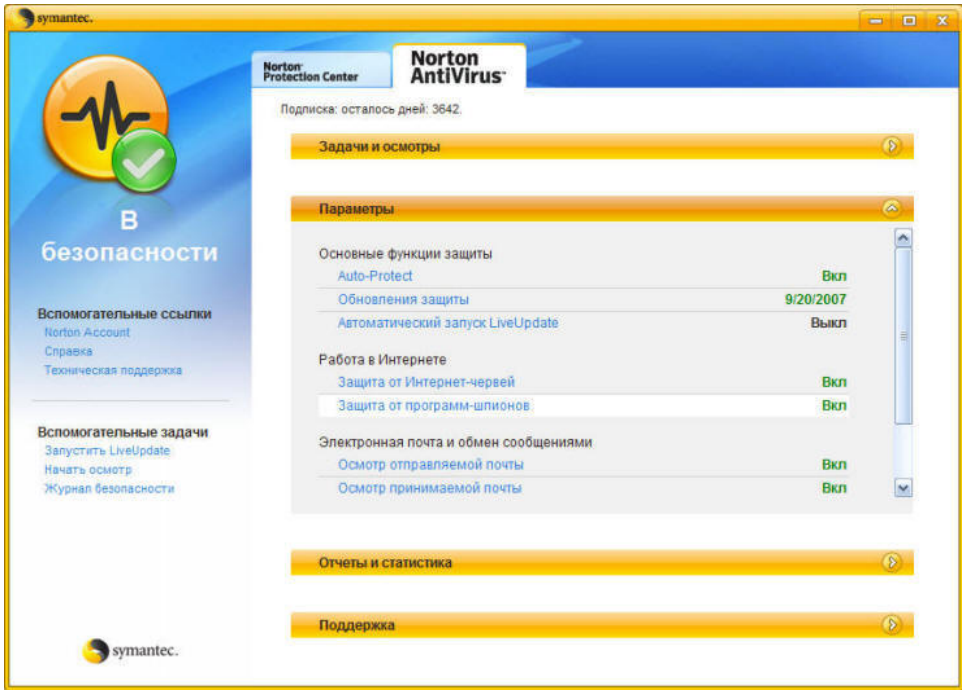


Рис. 18. Главное окно Norton Antivirus

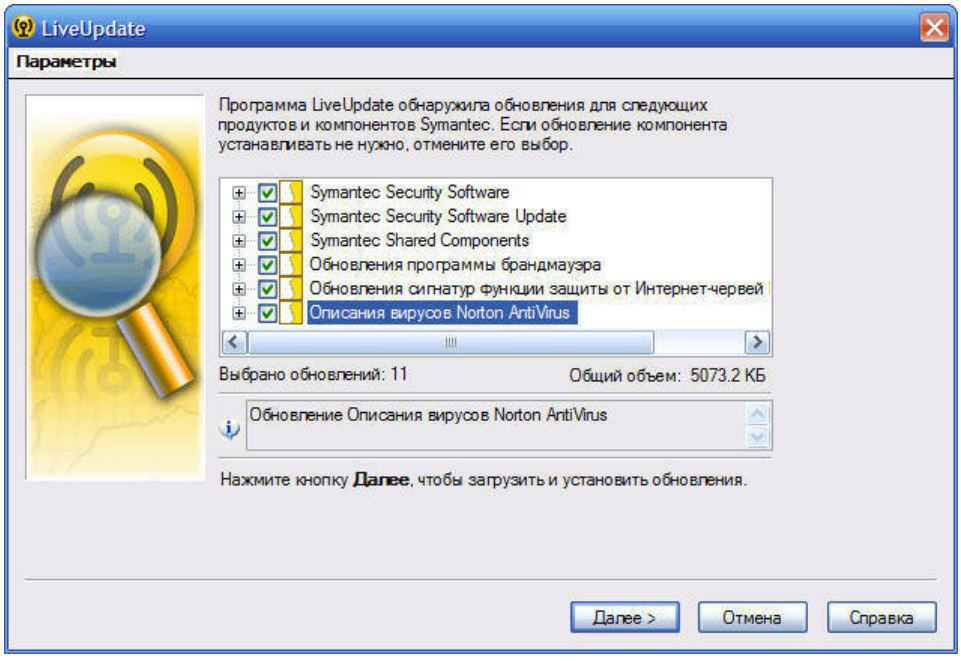


Рис. 19. Окно программы автоматического обновления Norton Antivirus

На рис. 18 представлено главное окно Norton Antivirus. Команды главного окна позволяют выполнить разнообразные настройки параметров антивирусного средства. На рис. 19 изображено окно программы автоматического обновления Norton Antivirus в режиме получения обновлений с Web-узла компании Symantec.

В заключение следует отметить, что нет необходимости использовать несколько антивирусов. Выбирайте наиболее подходящую и удобную для вас антивирусную программу и пользуйтесь только ею. Дело в том, что создатели антивирусных средств обмениваются между собой антивирусными базами данных по мере их пополнения. Поэтому любые антивирусные программы обладают примерно одинаковыми возможностями по обнаружению новых вирусов и лечению зараженных ими файлов при условии, что пользователь регулярно обновляет антивирусные базы данных.

Инструментальные средства информационных технологий

Напомним понятие «информационная технология». *Информационная технология (ИТ) – совокупность средств (методов, технических устройств и документации к ним), позволяющая получить из исходной информации конечную информацию в том виде, который нужен ее потребителю.*

Средства информационной технологии включают в себя методы получения, обработки, хранения и представления информации, а также аппаратно-программные средства реализации этих методов, поскольку чаще всего технической базой информационных технологий служит компьютерная техника и техника связи (коммуникационная).

Информационным процессом называется *процесс, в ходе которого появляется и видоизменяется информация.* Из этого определения следует, что *информационная технология – это реализация информационного процесса.*

Об информационных процессах

Несмотря на большое разнообразие информационных процессов, можно охарактеризовать их в целом, рассмотрев фазы информационного процесса. Реальный информационный процесс не обязательно содержит все выделенные фазы, возможно также, что в чистом виде та или иная фаза отсутствует. Тем не менее, содержание фаз процесса позволяет полнее охарактеризовать сам процесс. Можно выделить следующие фазы информационного процесса:

- *получение* исходной информации (от датчиков, из документации, от моделей и т. д.);

- *подготовка* информации в виде, отвечающем дальнейшему использованию (в компьютерных технологиях, для автоматизации производственных процессов и т. д.);

Например, если предстоит передача информации, данные представляются на носителе в таком виде, с помощью которого возможна модуляция электромагнитных волн. В случае хранения и во многих случаях обработки данные переводятся на магнитный носитель;

- *передача* (в случае необходимости) данных на требуемое расстояние по выбранному каналу связи;
- *обработка* (моделирование, расчеты, преобразование и т. д.);
- *хранение и использование* информации.

Средства информационных технологий

Различают следующие средства информационных технологий:

- математические;
- технические;
- алгоритмические;
- методические;
- информационные;
- программные.

Программные средства представляют собой инструменты для создания или обработки различных форм информации и часто называются инструментальными средствами. Они включают в себя:

- текстовые процессоры и издательские системы;
- табличные процессоры;
- системы управления базами данных;
- графические редакторы;
- системы автоматического проектирования;
- системы программирования;
- прикладные интегрированные системы.

Офисные технологии и их инструментальные средства

Офисные технологии являются одним из видов информационных технологий. Эти технологии и их инструментальные средства обеспечивают важный этап инженерной деятельности в любой предметной области – подготовку отчетной и другой документации на всех стадиях проведения работы.

Изложению сведений, относящихся к конкретным инструментальным средствам, предшествует материал, способствующий выработке единого подхода к освоению и использованию этих средств. Основ-

ное внимание при изложении этого материала обращалось на общность функциональных возможностей инструментальных средств и отображение этой общности в их интерфейсе. Кроме того, рассмотрены общие черты процесса создания любого документа, независимо от вида информации, представленной в нем. Такой подход облегчает как изложение последующего материала, так и его восприятие начинающими пользователями. Более опытные пользователи смогут использовать все эти предваряющие сведения для выработки системного взгляда на известные им и ставшие привычными вещи.

Сведения, относящиеся к текстовому процессору MS Word, позволяют познакомиться с основными возможностями, которые интерфейс процессора предоставляет на разных стадиях процесса создания деловой документации. Каждая такая стадия представлена, как реализация определенного этапа общего процесса создания документа. Это позволяет осваивать технологию использования текстового процессора MS Word не в отрыве от других инструментов, а как составную часть информационной офисной технологии.

Материал, посвященный табличному процессору MS Excel, излагается аналогично. Приводятся сведения по реализации этапов общего процесса создания документа средствами табличного процессора. Указываются особенности, присущие этому инструменту и обусловленные табличным видом представления информации. Главной особенностью MS Excel является то, что это не только инструмент, предназначенный для отображения табличной информации, но и средство ее обработки. Этот факт потребовал дополнительного рассмотрения приемов и средств автоматизации решения расчетных и информационных задач.

Инструментальное средство подготовки презентаций MS PowerPoint является весьма полезным для специалиста в любой предметной области. Его можно эффективно использовать и в учебном процессе. В пособии приведены основные понятия PowerPoint, этапы создания презентации. Основные возможности PowerPoint рассмотрены на примере поэтапного создания простой презентации начинающим пользователем.

Версии пакета Microsoft Office

Текстовый процессор MS Word и табличный процессор MS Excel (электронные таблицы) были разработаны еще для операционной оболочки Windows 3.1 как отдельные программы для решения наиболее часто встречающихся в офисной деятельности задач. Уже тогда в них впервые был реализован принцип соответствия экранного представления печатному – WYSIWYG (от английского «What You See Is What You Get» – «Что видите, то и получите»).

В 1995 году Microsoft выпустила первую версию пакета программ, предназначенных для решения офисных задач, для автоматизации делопроизводства − Microsoft Office 95. В пакет вошел текстовый процессор MS Word, табличный процессор MS Excel, программа подготовки презентаций MS PowerPoint и некоторые другие средства. Центральное положение в пакете занял текстовый процессор MS Word. Пакет позволил организовать эффективный обмен данными между всеми приложениями, входящими в его состав.

Очередная версия пакета MS Office 97 внесла мало полезных изменений для повседневной офисной работы и содержала ряд ошибок. Эти недостатки были устранены в версии пакета MS Office 2000. Кроме того, в ней введены мощные средства поддержки сетевых режимов работы.

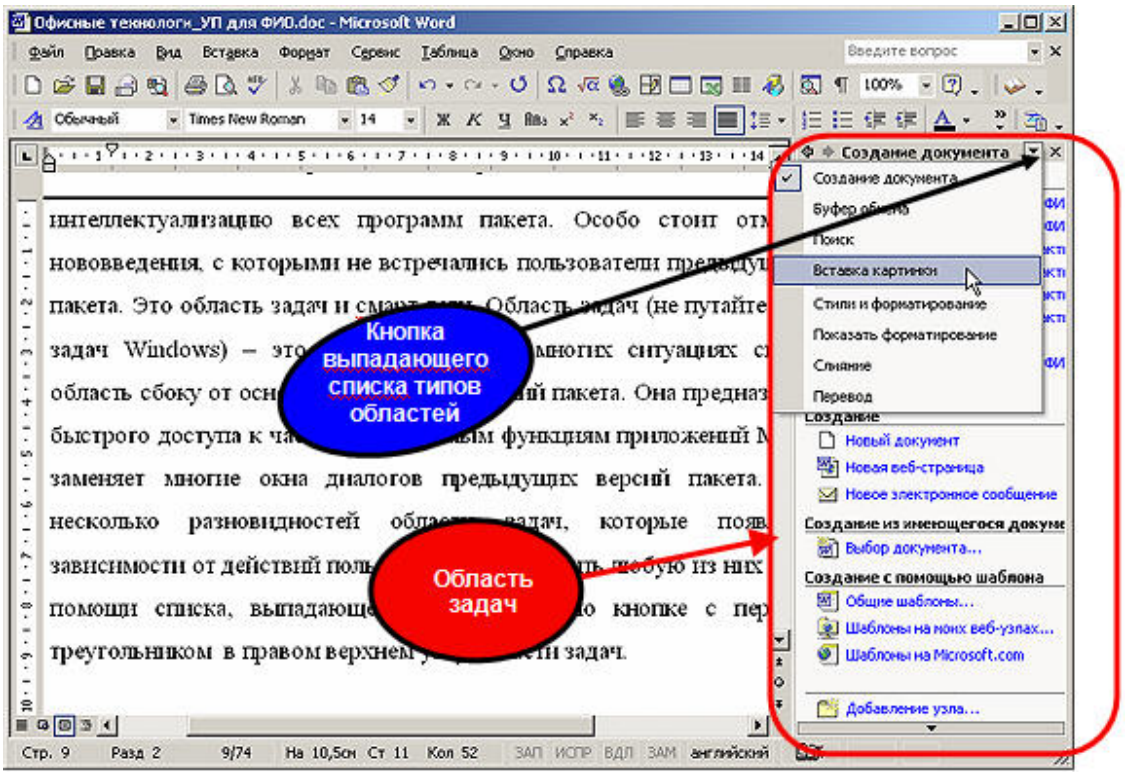


Рис. 20. Окно MS Word с областью задач и выпавшим списком типов областей (секций)

Следующая версия пакета MS Office XP (от англ. eXPerience – «жизненный опыт, опыт работы») значительно отличается от предшествующих версий. Удобство работающего – главный принцип построения всего пакета Office XP. В новом пакете исправлены многие недостатки предыдущих версий и добавлен ряд новых функций и возможностей, направленных, прежде всего, на еще большую интеллектуализацию всех программ пакета. Особо стоит отметить два нововведения, с которыми не встречались пользователи предыдущих версий пакета. Это область задач и

смарт-теги. Область задач (не следует путать с панелью задач Windows) – это появляющаяся во многих ситуациях специальная область сбоку от основного окна приложений пакета (рис. 20). Она предназначена для быстрого доступа к часто используемым функциям приложений MS Office и заменяет многие окна диалогов предыдущих версий пакета. Доступно несколько разновидностей области задач (секций), которые появляются в зависимости от действий пользователя. Отобразить любую из них можно при помощи списка, выпадающего при щелчке по кнопке с перевернутым треугольником в правом верхнем углу области задач.

Смарт-тег (интеллектуальная метка) – это появляющийся в различных местах редактируемых документов значок с маленьким меню (рис. 21), которое предоставляет пользователю выбрать какое-либо действие, связанное с тем фрагментом данных, около которого смарт-тег появляется.

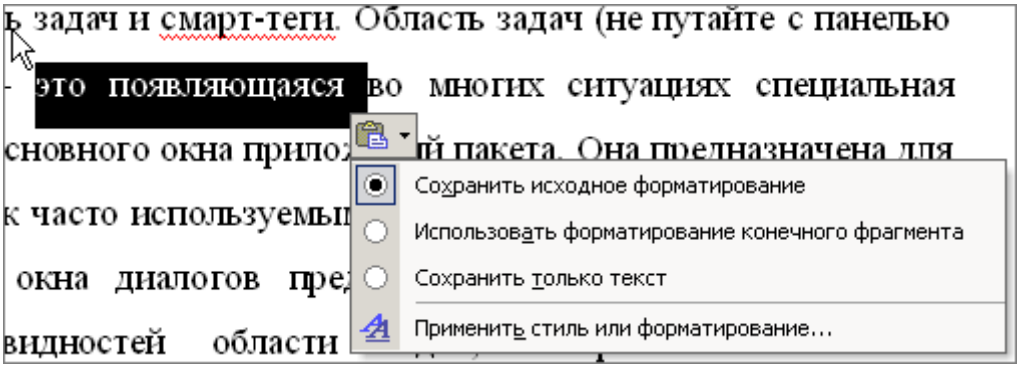


Рис. 21. Смарт-тег после копирования фрагмента текста в MS Word

В версии Microsoft Office 2003, а также в более поздних версиях Office 2007 и Office 2010 эти интеллектуальные возможности программ пакета, а также средства совместной групповой работы, были еще более развиты и усилены. Кроме того, в последних двух версиях произошло значительное изменение пользовательского интерфейса программ пакета за счет использования так называемой «ленты» вместо панелей инструментов. Часто используемые операции лента группирует по близости задач, связанных с этапами работы над документами.

Общие функциональные возможности офисных средств

Отображение общих функциональных возможностей в интерфейсе инструментальных средств информационных технологий

Чаще всего информация, с которой в повседневной деятельности имеют дело специалисты, представлена в таких формах, как:

- текстовая;
- табличная;

- графическая;
- математическая.

Независимо от формы представления содержание этапов информационных процессов (хранение, передача и получение информации), реализованных в инструментальных средствах информационных технологий, остается неизменным. Это находит свое отражение в общности некоторых задач, решаемых с помощью этих средств. Как следствие, общность задач приводит к аналогичным друг другу функциональным возможностям большинства инструментальных средств. К таким возможностям относятся:

- управление файлами (создание, редактирование, открытие и закрытие), хранящими информацию в виде, характерном для этого средства;
- обмен информацией между различного вида документами.

Кроме того, существуют общие возможности, не связанные с этапами информационных процессов. К ним относятся:

- возможность настройки внешнего вида инструмента;
- возможность располагать несколько открытых файлов в разных окнах одного приложения;
- возможность получать справочную информацию по самому инструментальному средству и решаемым им задачам.

Эти возможности находят свое отражение в элементах пользовательского интерфейса, прежде всего в меню и панели инструментальных кнопок (рис. 22).

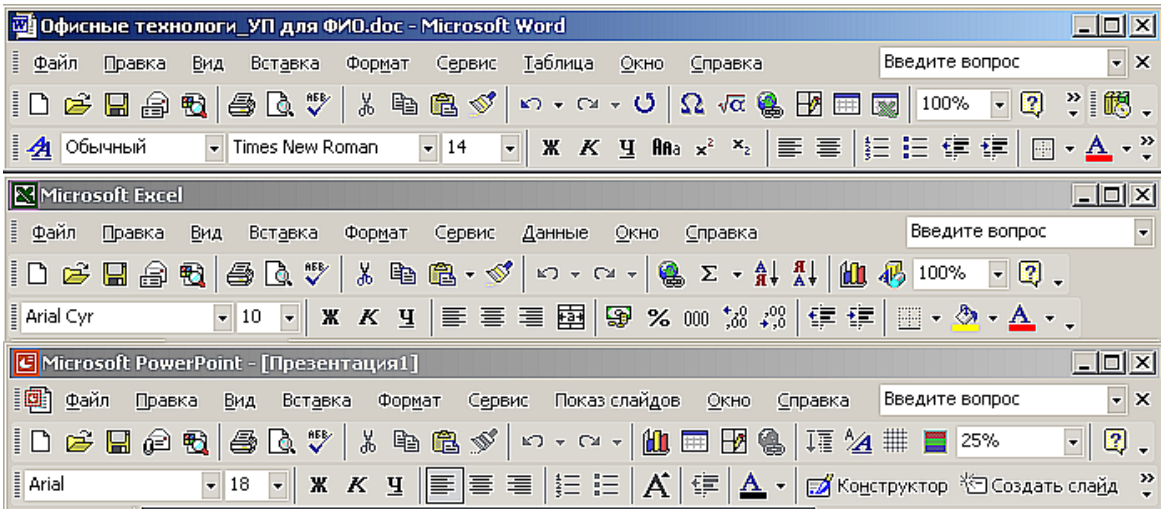


Рис. 22. Отображение общих функциональных возможностей в интерфейсе инструментальных средств

Команды меню и кнопки панели инструментов во многом дублируют друг друга. Меню представляет наибольший интерес для начина-

ющего пользователя, поскольку оно содержит полный набор команд, реализующих возможности инструментального средства.

Общими для главного меню всех приложений являются пункты: Файл (File), Правка (Edit), Вид (View), Сервис (Tools), Окно (Window) и Справка, ? (Help). В скобках приводятся названия пунктов главного меню англоязычных приложений.

Пункты **Файл** и **Правка** связаны с фазами хранения, получения и передачи информации. Остальные пункты обеспечивают настройки инструментального средства.

Файл – в выпадающем меню этого пункта сосредоточены все команды управления файлами, выводом на печать и внешним видом документов (создание нового файла, открытие существующего, сохранение файла, предварительный просмотр, печать и другие команды).

Правка – все команды редактирования, в том числе команды обмена данными через буфер обмена (Вырезать, Копировать и Вставить), команды отмены и повторения операций, а также команды контекстного поиска и замены.

Вид и **Сервис** содержат команды, позволяющие управлять внешним видом самого инструментального средства и видом документов, создаваемых им.

Окно – все команды работы с дочерними (подчиненными окнами), в том числе возможность перехода в любое открытое дочернее окно приложения.

Справка построена таким образом, что в любой момент времени можно получить информацию, как по любому интересующему разделу используемого инструмента, так и по его текущему рабочему состоянию.

Панели инструментов содержат кнопки, которые облегчают доступ к некоторым командам меню и другим командам, не входящим в него. Нажав соответствующую кнопку, можно запустить на выполнение такую команду. Более подробно со структурой главного меню и панели инструментов можно познакомиться, запустив соответствующее инструментальное средство.

В связи с важностью общих возможностей, связанных с фазами информационных процессов, далее подробно рассмотрены такие общие задачи как управление файлами и обмен информацией.

Управление файлами

Необходимо отметить, что практически все приложения Windows для операций открытия и сохранения файлов используют стандартные окна диалогов самой операционной системы. Поэтому внешний вид этих диалогов одинаков для любого приложения Windows.

Для дальнейшего изложения определим некоторые термины, связанные с работой в различных диалогах (не только упомянутых выше). Внутренние окошки диалогов, в которые нужно ввести какую-то информацию или в которые информация выводится, называются полями диалогов. Рядом с полем обычно помещается его название. Для выбора из нескольких предоставленных возможностей в диалогах используют два разных вида элементов – **переключатели** и **флажки**.

Элемент диалога, который позволяет указать выбор только одной возможности из нескольких, называется **переключателем**. Для выбора щелкают мышкой на нужном переключателе, и внутри него появляется черная точка.

Для выбора сразу нескольких возможностей в диалогах используется другой элемент в виде прямоугольного окошка. Такой элемент называется **флажком**.

Открытие существующего файла

Для открытия файла необходимо выполнить команду меню **Файл/Открыть.** В результате откроется окно диалога *Открытие документа* (рис. 23).

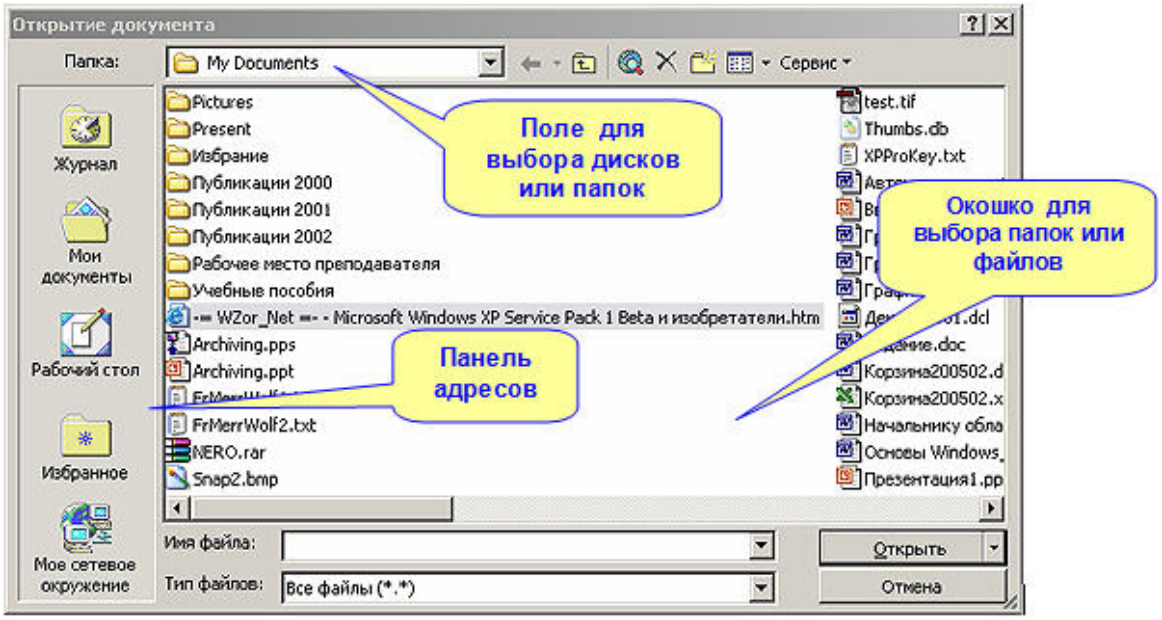


Рис. 23. Окно диалога ***Открытие документа***

Для открытия требующегося файла нужно сначала добраться до папки, в которой он находится. Путешествие по файловой структуре начинается с *поля для выбора дисков или папок.* В этом поле справа нахо-

дится кнопка выпадающего списка всех доступных дисков и папок. Щелкнув мышкой по этой кнопке, нужно выбрать из списка начальный пункт путешествия (например, диск **C:)**. Содержимое выбранного пункта (диска или папки) тут же отобразится в *окошке для выбора папок или файлов*. Далее нужно последовательно выбирать в окошке очередную папку и двойным щелчком открывать ее (вместо двойного щелчка можно, выделив такую папку, щелкнуть по кнопке *Открыть*). Названия открываемых папок будут отображаться в поле для выбора дисков или папок. Когда Вы доберетесь до папки, содержащей требуемый файл, нужно открыть его тем же приемом (двойной щелчок или кнопка *Открыть*).

Запомните! Действия, необходимые для открытия существующего файла, и используемое при этом окно диалога однотипны для всех приложений Windows.

Сохранение файла

Если требуется сохранить файл впервые, то после завершения работы с ним выполняют пункт меню Файл/Сохранить как… При этом действии открывается окно диалога Сохранение документа (рис. 24). С его помощью вводится имя файла и указывается папка, в которой следует поместить созданный файл.

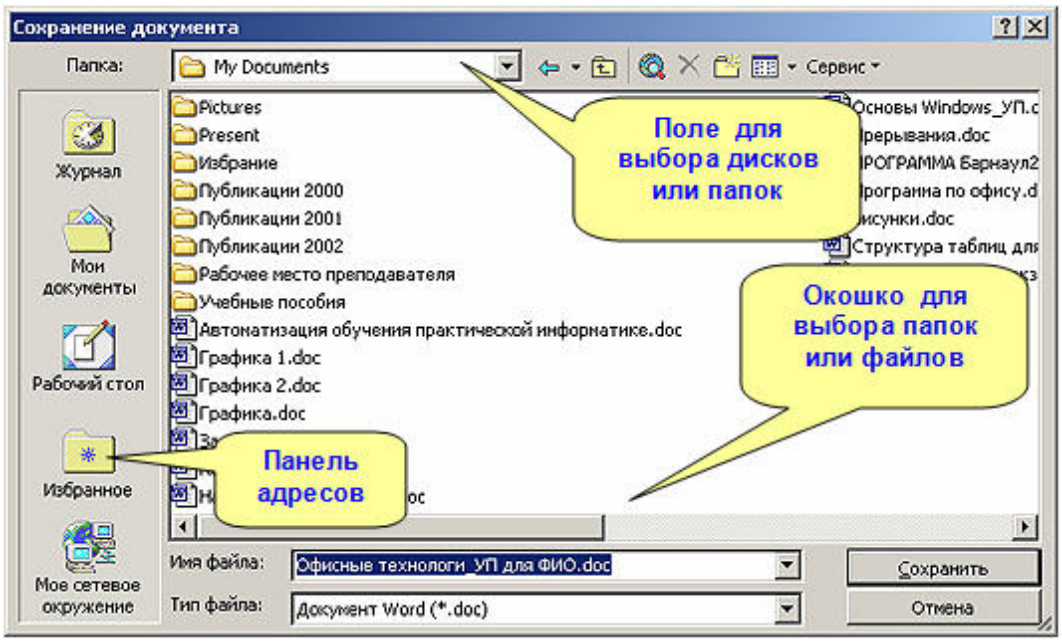


Рис. 24. Окно диалога Сохранение документа

Приемы работы *с диалогом сохранения файла* очень похожи на приемы работы с диалогом открытия файла, которые рассмотрены выше. Различие лишь в завершающих действиях. Когда Вы доберетесь до

папки, предназначенной для сохранения файла (название этой папки должно появиться в поле для выбора дисков или папок), то потребуется:

- ввести в поле диалога с названием *Имя файла* имя сохраняемого файла,
- выбрать в поле с названием *Тип файла* из предложенного списка нужный тип файла.

После этих действий следует щелкнуть по кнопке *Сохранить*.

Для сохранения изменений, сделанных в ранее созданном файле, выполняют пункт меню **Файл/Сохранить**.

Запомните! Действия, необходимые для сохранения созданного файла, и используемое при этом окно диалога однотипны для всех приложений Windows.

Обмен информацией между инструментальными средствами

Современные инструментальные средства, работающие под управлением среды Windows, предоставляют пользователю широкие возможности обмена данными во время работы с ними. Мы выделим следующие из этих возможностей:

- обмен данными через буфер обмена (Clipboard) среды Windows;
- динамический обмен данными – DDE;
- использование технологии OLE – механизма связывания и встраивания объектов.

Каждая из перечисленных возможностей имеет свои достоинства и недостатки. Во многих случаях обмен данными между программами с равным успехом можно проводить любым из этих способов. Приведем общую характеристику указанных способов.

Буфер обмена данными

Буфер обмена данными поддерживает схему обмена, приведенную на рис. 25. В программе 1 выделяется нужный фрагмент, который копируется в буфер обмена с помощью команды меню **Правка/Копировать.** Затем содержимое буфера вставляется с помощью меню **Правка/Вставить** в нужном месте программы 2. При использовании этого способа следует иметь в виду, что при копировании выделенного фрагмента происходит стирание всей той информации, что содержалась ранее в буфере.

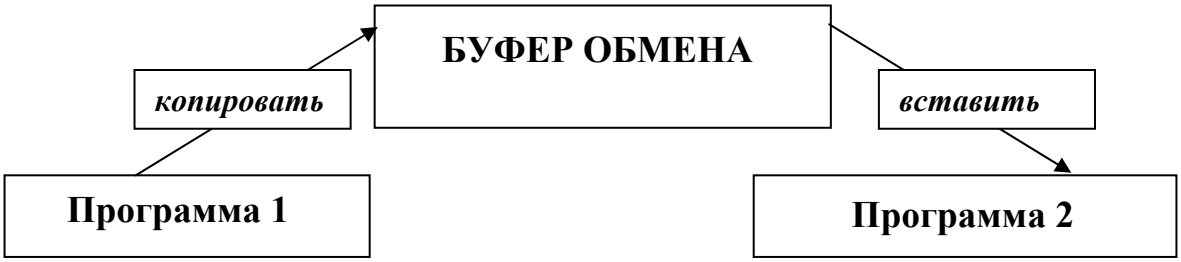


Рис. 25. Схема обмена данными между приложениями через буфер обмена

Динамический обмен данными

*Динамический обмен данными (**D**ynamic **D**ata **E**xchange)* – это такой способ обмена информацией, при котором скопированная информация отображает любые изменения, происходящие с оригиналом после момента вставки. При этом между оригиналом и копией информации устанавливается связь.

Например, в документ текстового процессора вставлена таблица из табличного процессора и установлена связь между обоими процессорами. Тогда при изменении таблицы в табличном процессоре эти изменения автоматически отобразятся в таблице в текстовом процессоре. С помощью DDE можно обмениваться данными только тогда, когда обе программы работают в Windows одновременно.

Технология OLE

В настоящее время DDE практически не применяется самостоятельно. Широко используется развитие этого механизма, которое получило название *технология OLE – механизм связывания и встраивания объектов (**O**bject **L**inking and **E**mbedding)*. Технология OLE расширяет возможности динамического обмена данными DDE. Под объектом понимается вставляемая в документ информация иного вида, чем основная информация документа. Например, рисунок в текстовом документе MS Word является объектом, созданном в графическом редакторе. С помощью технологии OLE инструментальные средства могут обмениваться самой разнообразной информацией – графиками, таблицами, текстами, видео- и звуковыми фрагментами и т. п. Причем, возможны две ситуации. Первая – вставляемый объект может быть непосредственно встроен в документ, увеличивая его объем. Вторая – документ и объект хранятся отдельно, а в документ вставляется только ссылка на объект. Эта ссылка почти не увеличивает объем документа, но позволяет отобразить такой объект при открытии документа. Первая разновидность вставки называется внедрением объекта, а вторая – связыванием. Недостатком связывания объекта с документом является невозможность отображения объекта при его случайном или намеренном удалении или перемещении.

Для вставки объектов в документ любого приложения используются разные способы. Способ вставки зависит от предыстории вставляемого объекта:

1) объект существует как часть другого документа и хранится в файле;
2) объект целиком хранится в отдельном файле, например, файл рисунка;
3) объект не существует и должен быть создан в процессе работы над основным документом.

В первом случае для вставки необходимо открыть документ, хранящий нужный объект, и скопировать этот объект. Затем перейти в окно документа, требующего вставки и выполнить команду меню **Правка/Специальная вставка…**. После появления окна диалога *Специальная вставка* необходимо в поле *Как:* из предложенного списка форматов вставки выбрать строку, в которой есть слово *объект* (возможна вставка в других форматах, которую называют *конвертированием* формата). Дополнительно для установления связи между документами по механизму DDE можно выбрать положение переключателя *Связать*.

Во втором и третьем случаях, предварительно установив курсор в месте вставки, используют команду меню **Вставка/Объект…** (в окне диалога *Вставка объекта* нужно выбрать вкладку *Создание из файла* или *Создание*, соответственно). Во втором случае можно дополнительно установить режим связывания с файлом.

При использовании технологии OLE принята следующая терминология. То инструментальное средство (программа), которое обращается к другому за нужной информацией, называется *клиентом*, а которое ее создает – *сервером*. При этом в месте вставки объекта в документ, создаваемый клиентом, предоставляется возможность работать с информацией требуемого вида (график, таблица и т. п.). Для этого приложение-клиент получает возможность связаться с приложением-сервером и использовать все его средства тогда, когда это необходимо.

Существует два метода работы с OLE. Первый из них состоит в том, что сервер при вызове запускается в отдельном окне. После того как обработка данных закончена, сервер завершает работу, и пользователь возвращается в приложение-клиент, в документ которого внедрен объект. Второй вариант заключается в том, что в приложении-клиенте при вызове сервера появляются новые меню и панели инструментов, которые и позволяют выполнить соответствующую обработку данных. После завершения обработки восстанавливается прежний интерфейс приложения-клиента.

Основные этапы работы над документом

В процессе работы над документом, независимо от вида информации, включаемой в него (текстовая, табличная и т. д.), можно выделить следующие этапы:

- выбор рабочего инструмента;
- создание пустого документа;
- создание содержания документа;
- редактирование содержания документа;
- оформление документа в соответствии с заданными требованиями;
- сохранение документа.

На первом этапе выбирается программный инструмент, наиболее подходящий для работы с тем видом информации, который преимущественно входит в создаваемый документ. Например, для текстового документа используется текстовый процессор, для создания таблиц и их обработки – табличный процессор.

На втором этапе с помощью средств выбранного инструмента создается пустой документ, т. е. документ без содержимого с некоторыми заранее заданными свойствами

Этап создания содержания документа заключается в наполнении пустого документа требуемой информацией в соответствии с замыслом документа.

Редактирование документа подразумевает изменение содержания уже созданного и наполненного нужными сведениями документа. На этом этапе используются все возможности выбранного инструмента для более точного выражения основного замысла документа.

На этапе оформления документа используются средства выбранного инструмента для придания документу вида в соответствии с заданными требованиями. Любой документ офисного приложения обладает внутренней *структурой.* Структурные элементы документа и их свойства определяются видом информации, который преимущественно входит в него. Заданные требования, как правило, связаны с внешним видом структурных элементов (например, с оформлением абзацев текстового документа).

На последнем этапе осуществляются действия по сохранению созданного документа в виде, доступном для его дальнейшего использования.

Текстовые процессоры

Текстовые редакторы и процессоры – инструментальные средства для создания любых текстов (документов, программ и др.). Существует множество программ, относящихся к этому классу инструментальных средств – от простых редакторов для работы с текстами не-

большого объема до сложных текстовых процессоров. Возможности у них различные, но все они обеспечивают операции редактирования: *выделение части текста*, ее *копирование*, *перенос с одного места на другое, удаление*, *замена слов и поиск нужных слов в тексте* и ряд других операций.

Текстовые редакторы – средства, позволяющие только вводить и редактировать тексты.

Текстовые процессоры – это многофункциональные программы обработки текстов. Кроме ввода и редактирования они позволяют форматировать тексты, а также внедрять в текстовые документы нетекстовые объекты различной природы. В дальнейшем мы будем вести речь о текстовом процессоре Microsoft Word, хотя большая часть сказанного здесь относится к любому текстовому процессору.

Как инструментальное средство информационных технологий, текстовый процессор, помимо возможностей редактирования собственно текста, обладает рядом дополнительных возможностей.

Во-первых, он поддерживает все механизмы обмена информацией, реализованные в операционной системе. Это позволяет встраивать в создаваемые тексты фрагменты, содержащие не только текстовую информацию, но и графическую, математическую, звуковую, редактировать сообщения электронной почты, и тем самым повышать информационную ценность обычного текстового документа. Дополненный такими возможностями, текстовый процессор становится совершенно необходимым средством для подготовки отчетной документации на всех стадиях проведения работы. Кроме того, создаваемый документ, обладая повышенной информационной ценностью, может служить не только для отчетности, но и для других целей, например, для обучения или исследований.

Второй отличительной чертой текстового процессора можно считать оснащение его мощной системой проверки правописания с чертами интеллектуальной системы, отслеживающей ошибки в создаваемом тексте.

Представление WYSIWYG, позволяет просмотреть на экране монитора готовый к печати документ, не затрачивая бумагу на пробную распечатку.

Еще одной отличительной чертой процессора является то, что при создании документов целый ряд процедур на стадиях форматирования и редактирования можно выполнять автоматически.

Совокупность этих свойств сближает текстовый процессор с настольной издательской системой.

Этапы работы с текстами в MS Word

При работе с текстами в MS Word выделяются следующие этапы:

- создание пустого документа;
- ввод текста;
- редактирование текста;
- форматирование текста;
- сохранение документа.

Создание пустого документа

В основе любого документа лежит шаблон. *Шаблон* – это совокупность настроек структурных элементов документа, задающих его предопределенные свойства (например, настройки параметров абзацев). При открытии MS Word автоматически создается документ самого общего вида, соответствующий основному шаблону текстовых документов **Новый документ**. Тем не менее, текстовый процессор предоставляет иные возможности для создания нового документа желаемого вида, отличающегося от общего. Для доступа к соответствующим средствам создания нужно выполнить команду меню **Файл/Создать.** Это приводит к открытию секции *Создание документа* области задач (рис. 26).

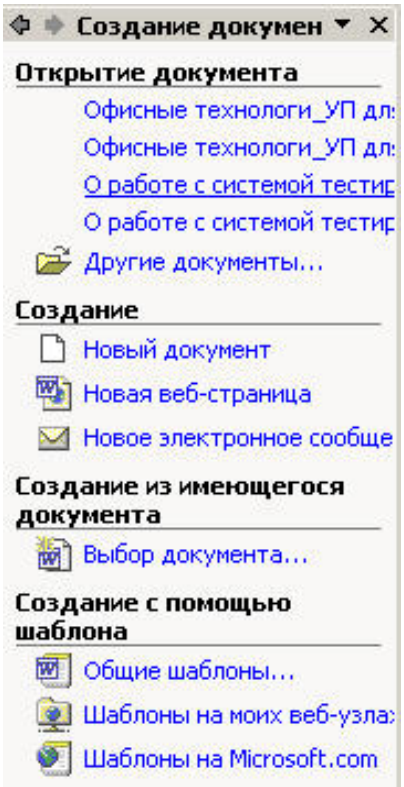


*Рис. 26 . Фрагмент секции **Создание документа** области задач, обеспечивающий доступ к средствам создания документа*

Раздел *Создание* в области задач дает возможность создать текстовый документ самого общего вида. Для этого нужно щелкнуть в упомянутой секции по ссылке **Новый документ**.

Раздел *Создание с помощью шаблона* дает возможность создать документ в тех случаях, когда он имеет какую-то специфику. В таком случае нужно щелкнуть по ссылке **Общие шаблоны…** в разделе *Создание с помощью шаблона***.** В появившемся стандартном окне диалога *Шаблоны* (рис. 27) можно отыскать подходящий шаблон и воспользоваться им. В созданном документе для работы предоставляются уже готовые настройки использованного шаблона.

Раздел *Создание из имеющегося документа* дает возможность создать документ, взяв за шаблон уже готовый документ MS Word. В таком документе удаляется все его содержимое, после чего файл документа сохраняется под другим именем и с расширением **.dot**. Для поиска требуемого файла MS Word нужно щелкнуть в разделе по ссылке **Выбор документа**, открыть окно диалога и выполнить в нем поиск. В этом случае для работы в создаваемом документе предоставляются уже готовые настройки стилей использованного документа.

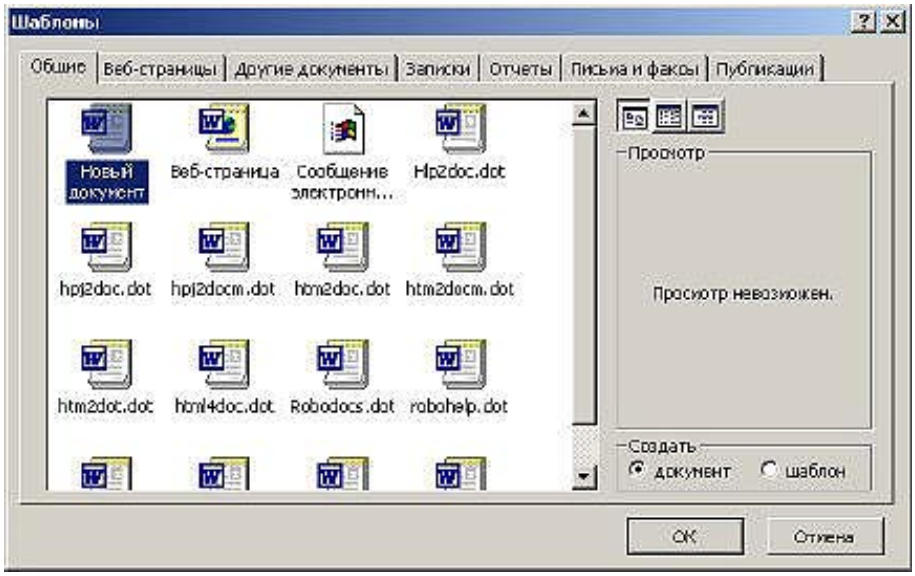


Рис. 27. Окно диалога ***Создание документа*** *с шаблонами различных документов*

Ввод текста

Большинство команд, обеспечивающих комфортные условия для работы при наборе текста, сосредоточено в пункте меню **Правка (**рис. 28).

Набор текста осуществляется с клавиатуры. Операции, выполняемые при вводе текста, запоминаются в специальном буфере памяти. Это позволяет отменять ошибочные действия, возвращая текст к тому со-

стоянию, которое было до совершения ошибки (команда **Отменить…** в меню **Правка** или соответствующая кнопка на панели инструментов).

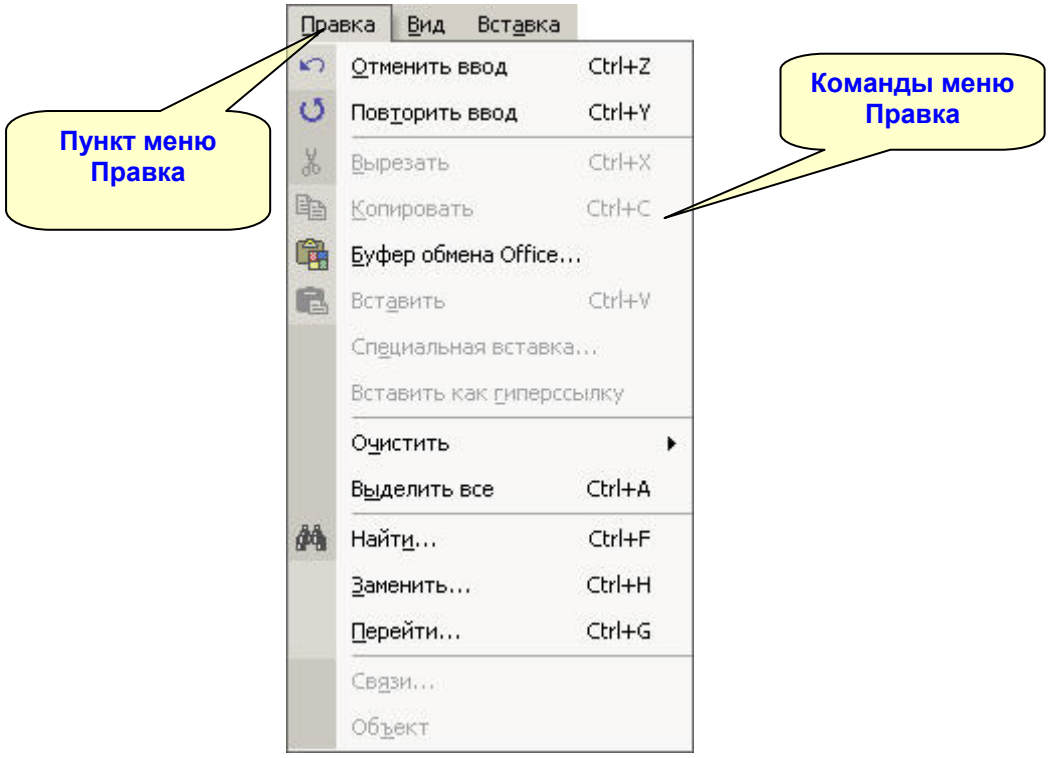


*Рис. 28. Раскрытый пункт меню **Правка***

Работа над текстом может быть автоматизирована с помощью режима **Автотекст**. Он позволяет заранее заготовлять фрагменты, часто встречающиеся в тексте и автоматически вводить их по мере надобности. Подготовка фрагментов выполняется с помощью окна диалога **Автозамена**, которое вызывается командой меню **Вставка/Автотекст/Автотекст**, их вставка – обращением к содержимому списка заготовленных фрагментов в пункте меню **Вставка/Автотекст**. Замеченные опечатки можно устранять, используя команду меню **Правка/Заменить**.

Редактирование текста

Под редактированием понимают изменение уже существующего документа. Поэтому редактирование начинается с загрузки уже существующего документа. Для этого нужно открыть пункт меню **Файл/Открыть** и воспользоваться средствами появляющегося при выполнении команды окна диалога.

В основном, средства редактирования предоставляются командами пункта меню **Правка**. Часть средств предоставляется командами пункта меню **Сервис (**рис. 29). Очень полезно в процессе редактирова-

ния использование словаря смысловых синонимов к словам, вызывающим сомнение. Для его использования нужно выполнить команду меню **Сервис/Язык/Тезаурус**. Процессор предоставляет возможность автоматизировать проверку правописания. Соответствующие средства включают в себя проверку орфографии и грамматики. Запуск средства проверки выполняют командой **Сервис/Правописание.**

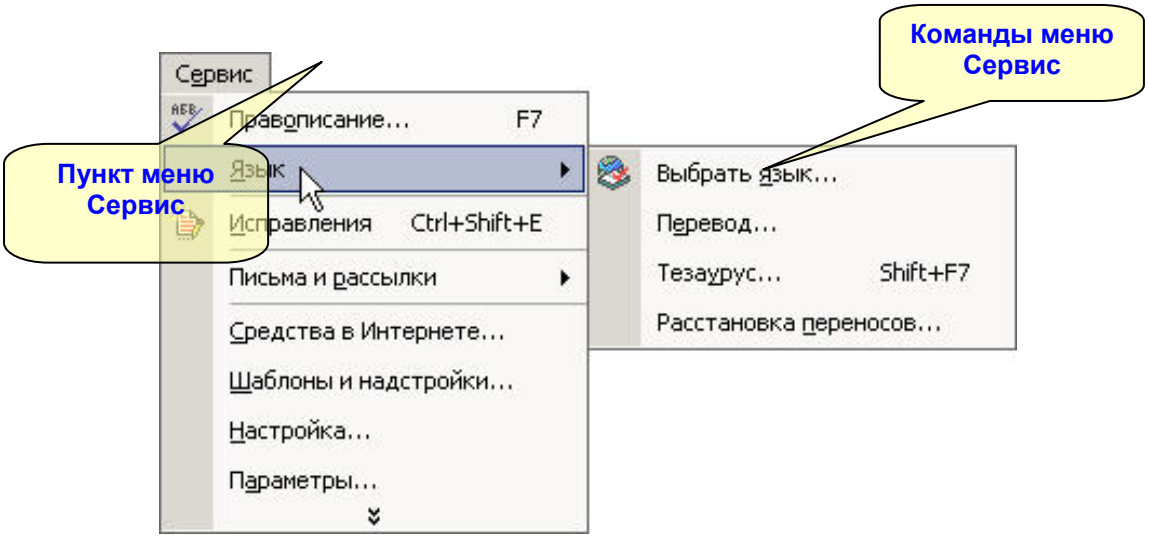


Рис. 29 . Раскрытый пункт меню ***Сервис***

Завершается редактирование сохранением измененного текста. Для этого достаточно выполнить команду меню **Файл/Сохранить**.

Форматирование текста

К оформлению созданного документа очень часто предъявляются определенные требования, выполнение которых и является содержанием этапа форматирования. Форматирование текста осуществляется средствами меню **Формат** (рис. 30).

Форматирование включает в себя:

- выбор и изменение гарнитуры шрифта;
- управление размером, начертанием и цветом шрифта;
- управление методом выравнивания текста;
- управление параметрами абзаца;
- создание списков.

Меню содержит как команды, предоставляющие возможности изменять перечисленные выше параметры, так и возможности управлять рядом других характеристик.

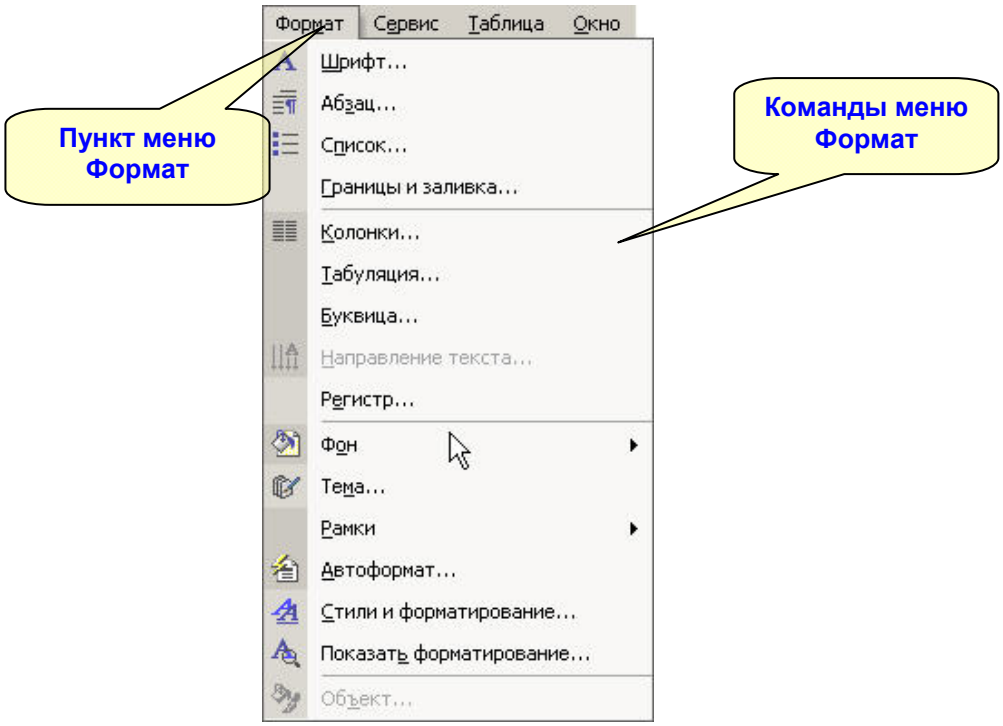


Рис. 30. Раскрытый пункт меню ***Формат***

Управление параметрами шрифта текста

Текстовый процессор дает возможность для оформления текста выбирать шрифты различных гарнитур, начертаний, размеров и цвета (команда меню **Формат/Шрифт)**. Выполнение этого пункта приводит к появлению окна диалога *Шрифт*, средствами которого можно придать шрифту нужные свойства (рис. 31).

Гарнитура шрифта – это комплект однотипных шрифтов, представленный названием. Например, «Times New Roman» или «Arial». Под начертанием понимают такие атрибуты шрифтов, как толщина (жирность), наклон, подчеркивание.

Типы шрифтов:

- растровый;
- векторный.

В **растровых** шрифтах каждый символ представляет собой матрицу, каждый элемент которой определяет наличие или отсутствие точки в изображаемом символе. Размер матрицы для всех символов одинаков, поэтому иногда шрифт измеряют размерностью матрицы. Гарнитура такого шрифта представлена в виде комплекта шрифтов разного размера и толщины.

В **векторных** шрифтах каждый символ представлен в виде отрезков кривых. Достоинством этих шрифтов является возможность программных изменений масштаба и поворота символов, а также изменение толщины линии.

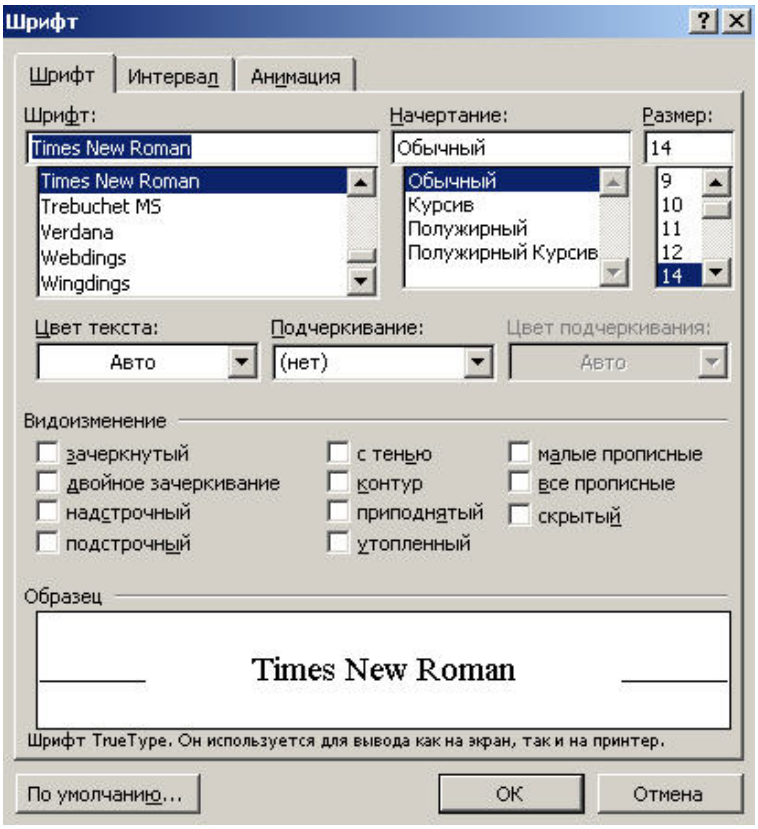


Рис. 31. Окно диалога ***Шрифт***

Шрифты **True Type** сочетают в себе достоинства растровых и векторных шрифтов. Это достигается тем, что каждый символ представлен совокупностью точек и алгоритмов преобразований; при выводе на экран или принтер символ предварительно преобразуется в растровую картинку. Это дает возможность создавать шрифты, символы которых выглядят одинаково как на экране, так и на принтере. Гарнитура **True Type** шрифтов не содержит несколько наборов, а представлена математическими соотношениями, позволяющими получать однотипные шрифты разного размера и начертаний только изменением коэффициентов таких соотношений.

Существуют две категории шрифтов: **с засечками** и **без засечек** (*рубленные*). Характерными представителями первой категории являются представители семейства Times, второй Arial.

Примеры: шрифт с засечками, шрифт без засечек.

Управление параметрами абзаца

Абзац – структурный основной элемент любого текста. Он характеризуется набором параметров, из которых следует выделить:

- режим выравнивания;
- величина отступа первой строки (красная строка);

- величина интервала перед абзацем и после него (отбивка между абзацами).

Для настройки параметров абзаца используют команду меню **Формат/Абзац**. В результате появляется окно диалога *Абзац*, средствами которого можно придать абзацу нужные свойства (рис. 32).

Текстовый процессор поддерживает 4 типа выравнивания текста:

- по левому краю;
- по центру;
- по правому краю;
- по ширине.

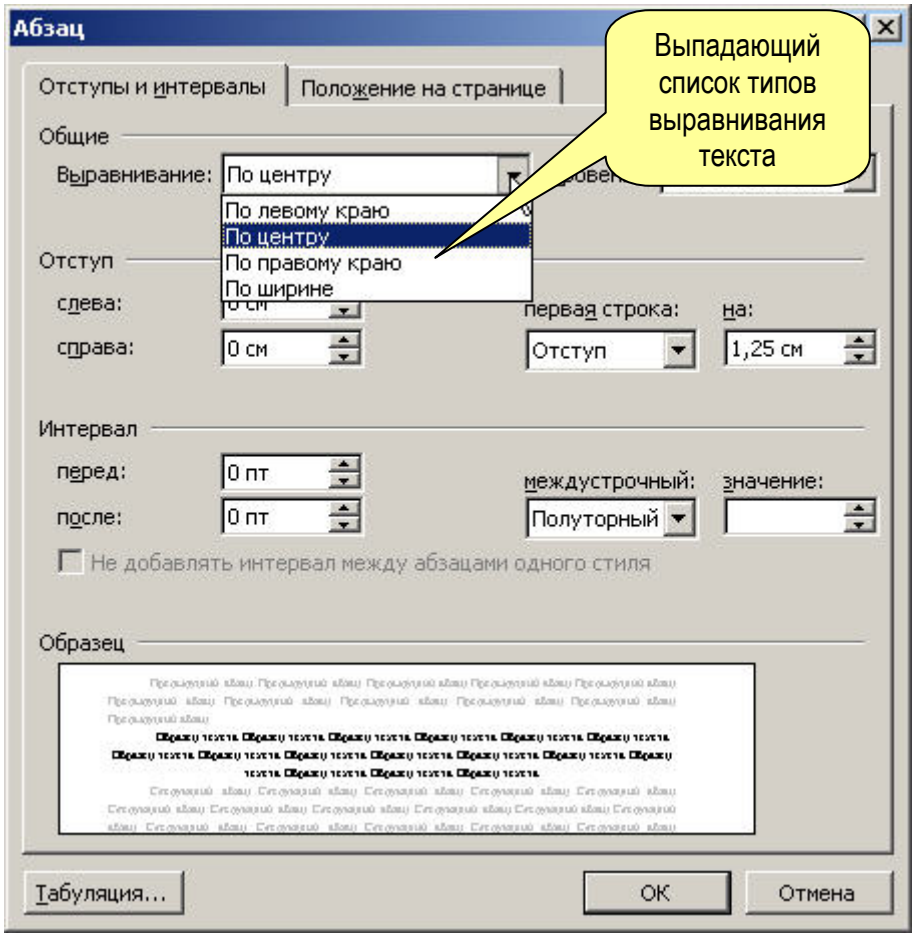


*Рис. 32 . Окно диалога **Абзац** с выпадающим списком типов выравнивания*

Для печатных документов на русском и немецком языках рекомендуется в основном тексте использовать выравнивание по ширине. Для документов на английском языке основной метод выравнивания – по левому краю.

Для задания красной строки (отступа) используют поле *Первая строка* в окне диалога **Абзац**, а для отбивки – поля секции *Интервал* в этом окне.

Отбивка между абзацами и красная строка несовместимы друг с другом. Комбинация этих стилей допускается только для нумерованных или маркированных списков. Основной текст оформляется с отступом, а списки – без него, но с отбивкой.

Создание списков

Список – это нумерованное или маркированное перечисление. Для создания списков используют соответствующие кнопки на панели инструментов либо пункт меню **Формат/Список**.

Сохранение текстового документа

По умолчанию вновь создаваемый текстовый документ имеет имя *Документ,* которое во избежание возможных недоразумений при дальнейшей работе следует изменить. Чтобы сохранить файл под новым именем следует выполнить команду меню **Файл/Сохранить как...** При этом открывается стандартное окно диалога **Сохранение документа.** Работа с этим диалогом одинакова для всех приложений Windows и описана ранее при рассмотрении общих возможностей офисных средств.

Особенность сохранения текстового файла проявляется в том, что в поле **Тип файла** по умолчанию всегда предлагается тип **Документ Word** (*.doc). Здесь .doc – стандартное расширение файла, создаваемое процессором MS Word .

Если сохраняемый документ предназначен для работы с ним в предшествующих версиях текстового процессора MS Word или вообще в иных средах, то для сохранения файла следует выбрать в этом поле из предлагаемого списка соответствующий тип документа.

Приемы и средства автоматизации оформления документов

В работе над документом, особенно на стадии оформления его внешнего вида, часто возникает потребность многократно выполнять однотипные операции или наборы операций. Для таких случаев MS Word предлагает такие мощные средства автоматизации разработки документов, как *стили* и *шаблоны*.

Работа со стилями

Абзац – структурный элемент текстового документа. Каждый заголовок документа тоже рассматривается как отдельный абзац. Форматирование абзацев представляет собой достаточно трудоемкий процесс. Диалог, который появляется после выполнения команды меню **Формат/Абзац**, имеет много различных элементов управления и выполнять

их настройку для каждого абзаца отдельно довольно утомительно. Эта работа автоматизируется при использовании стилей.

Стиль оформления – это именованная совокупность настроек параметров шрифта, абзаца, языка.

Использование стилей обеспечивает простоту форматирования абзацев и заголовков, а также единство их оформления в пределах всего документа. Кроме того, с их помощью создаются структуры и оглавления.

Стили позволяют применить к абзацу или слову целую совокупность атрибутов форматирования за одно действие.

Если требуется изменить форматирование какого-то одного элемента текста, то достаточно изменить стиль, который был применен к этому элементу.

Пусть, например, при форматировании заголовков четвертого уровня в документе использовался стиль, обеспечивающий выравнивание по левому краю и шрифт «Arial», полужирный курсив размера 13. Тогда, чтобы позже изменить форматирование всех заголовков на выравнивание по центру и шрифт «Arial» размера 16, достаточно будет изменить свойства стиля. Для этого нужно щелкнуть по любому из заголовков этого уровня и выполнить команду меню **Формат/Показать форматирование.** Это приводит к открытию секции *Показать форматирование* в области задач (рис. 33).

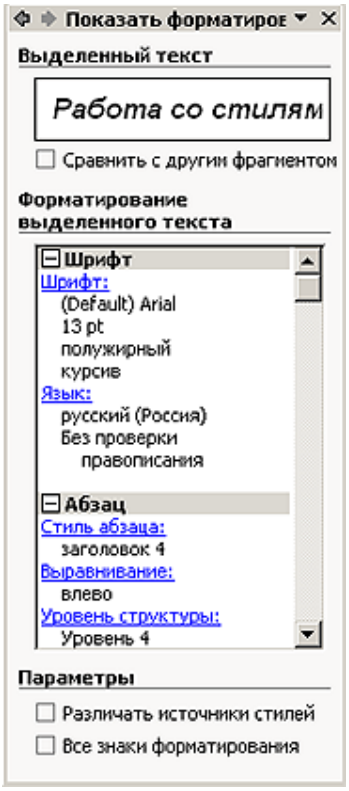


*Рис. 33. Секция **Показать форматирование** области задач*

Раздел **Абзац** в области задач дает возможность изменить стиль абзаца (заголовка). Для этого нужно щелкнуть в упомянутой секции по ссылке **Стиль абзаца**. Это приводит к появлению окна диалога *Стиль*, в котором следует щелкнуть по кнопке **Изменить**. Поверх окна диалога *Стиль* появится окно диалога *Изменение стиля* (рис. 34). Настройку стиля выполняют в этом окне.

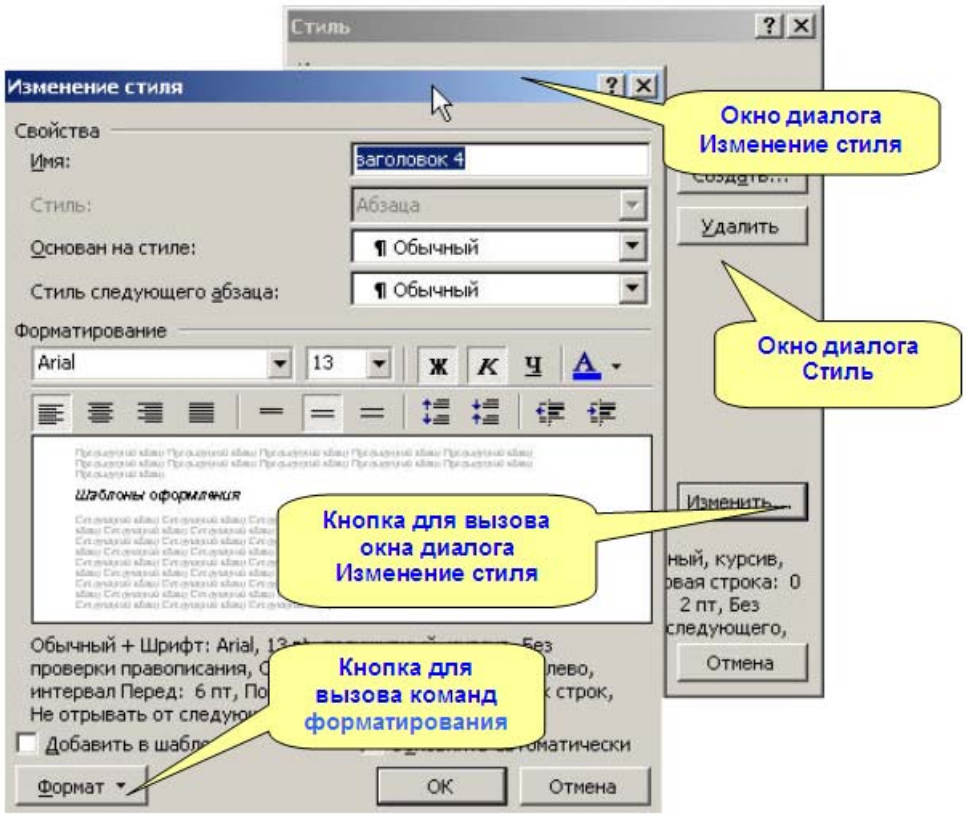


Рис. 34 . Каскад окон для настройки параметров стиля

Шаблоны оформления

Шаблон – это защищенная от непреднамеренных повреждений заготовка будущего документа, хранящая совокупность стилей его оформления. При использовании шаблонов не нужно заботиться о настройке стилей заголовков, основного текста и других стилей. Они уже определены в шаблоне. На основе шаблона создается содержание документа, оформленное в соответствии с этими стилями. При создании документа шаблон можно выбрать из списка готовых шаблонов. Этот список отображается в окне диалога **Шаблоны** (см. рис. 27). Чтобы открыть его, нужно выполнить команду меню **Файл/Создать…** и после появления области задач щелкнуть по ссылке **Общие шаблоны** в разделе *Создание с помощью шаблона*. При сохранении документа шаблон, взятый в качестве основы, остается неизменным и пригоден для дальнейшего использования.

Создание комплексных текстовых документов в MS Word

Для повышения информативности в текстовые документы очень часто внедряют (включают) наряду с текстом нетекстовые объекты (формулы, таблицы, диаграммы, иллюстрации и др.). Текстовые документы, включающие нетекстовые объекты, будем называть *комплексными*. О вставке готовых объектов, которые скопированы в буфер обмена или хранятся в отдельных файлах, говорилось в разделе «Технология OLE». Там же говорится о вставке объектов, создаваемых непосредственно в процессе работы над документом. На рис. 35 показан пример окна диалога *Вставка объекта* для случая создания объекта в процессе работы над документом. В этом окне предлагается список типов объектов (*Тип объекта*) для внедрения. После выбора нужного объекта из предложенного списка нужно щелкнуть по кнопке *ОК*.

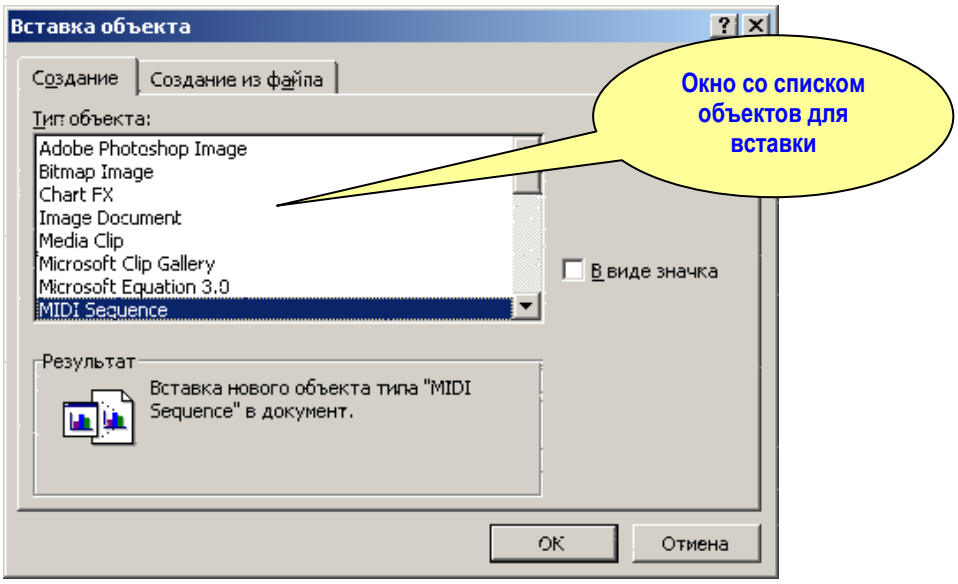


Рис. 35 . Окно диалога ***Вставка объекта***

Вставка математических формул

Очень часто в техническом тексте требуется вставить математические формулы. Делается это путем использования технологии OLE, т. е. внедрением объекта **Microsoft Equation**. При внедрении в окне диалога в списке типов объектов выбирается объект **Microsoft Equation 3.0.** После запуска вставки в месте расположения курсора появляется выделенная область, предназначенная для внесения формул, а панель инструментов текстового процессора заменяется панелью редактора формул. В рабочей области окна появляется панель *Формула*, которая предоставляет возможность пользоваться стандартными математическими символами для набора формул (рис. 36).

Рис. 36 . Панель математических символов

Завершив работу по созданию формул, нужно щелкнуть кнопкой мыши вне области, где создавалась формула.

Создание и встраивание диаграмм

Очень часто в техническом тексте требуется вставить диаграммы, графически отображающие табличную информацию. Делается это с помощью технологии OLE. Перед внедрением объекта выделяется таблица или ее нужная часть. При внедрении в окне диалога в списке типов объектов выбирается объект **Диаграмма Microsoft Graph**. Нажатие кнопки пуска выводит на экран два подчиненных окна: окно таблицы и окно диаграммы (рис. 37).

В окне таблицы расположена таблица с данными, которые были выделены ранее в исходной таблице. В случае необходимости в эти данные можно внести изменения. Используя пункты меню окна *Microsoft Graph*, можно придать диаграмме желаемый вид и размеры.

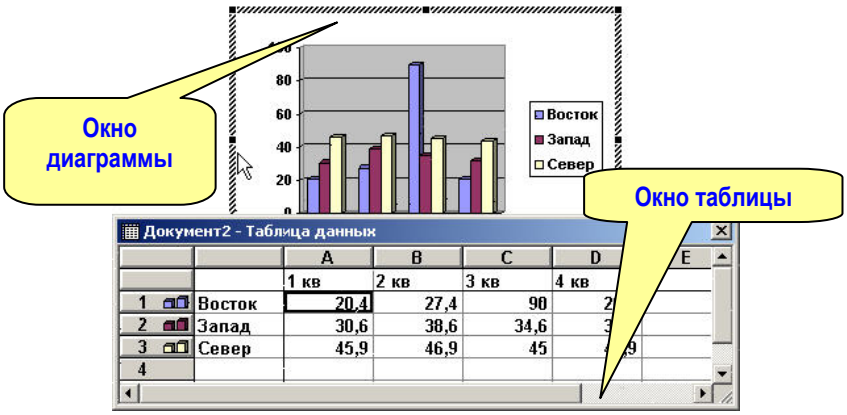


*Рис. 37. Каскад окон **Microsoft Graph***

В окончательном виде диаграмма, передающая содержание таблицы, вставляется в исходный текст после щелчка вне объекта. Для повторного изменения вставленной диаграммы достаточно в дальнейшем просто щелкнуть на ней дважды кнопкой мыши. Это приведет к автоматическому вызову программы **Microsoft Graph** со всеми ее возможностями для редактирования диаграммы.

Следует сказать, что возможности **Microsoft Graph** значительно уступают возможностям **Microsoft Excel** по созданию диаграмм. Поэтому, в большинстве случаев, предпочтительней вначале создать диа-

грамму в **Microsoft Excel**, а затем с помощью специальной вставки внедрить созданную диаграмму в документ **Microsoft Word** как объект.

Работа с таблицами

В процессоре имеется большой набор средств работы с таблицами, начиная от их создания и заканчивая выполнением некоторых вычислений с использованием логических и арифметических функций.

Создание таблицы и ее внедрение в место, указанное в тексте курсором, осуществляется выполнением команды меню **Таблица/Вставить/ Таблица** (рис. 38). Его выполнение приводит к появлению окна диалога *Вставка таблицы* (рис. 39**)**, которое позволяет указать число строк и столбцов таблицы и другие ее нужные параметры.

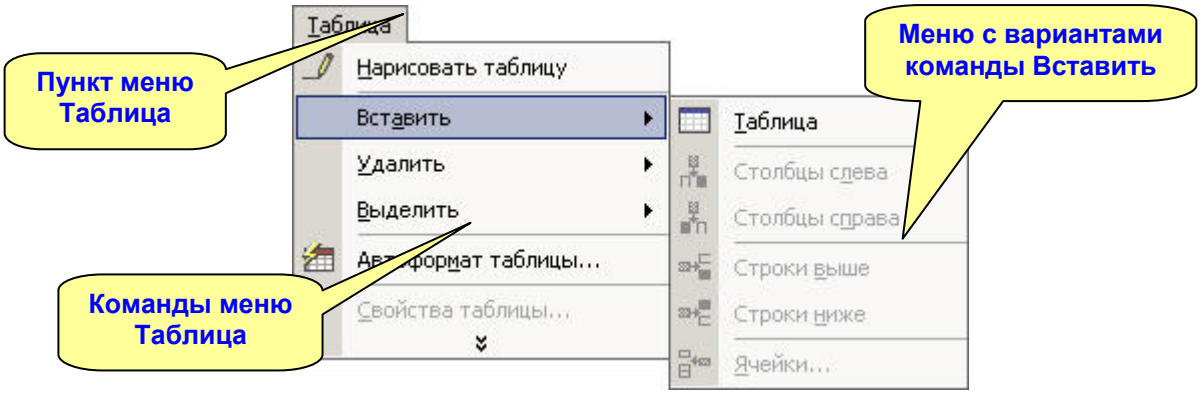


Рис. 38. Выпадающие меню при вставке таблицы

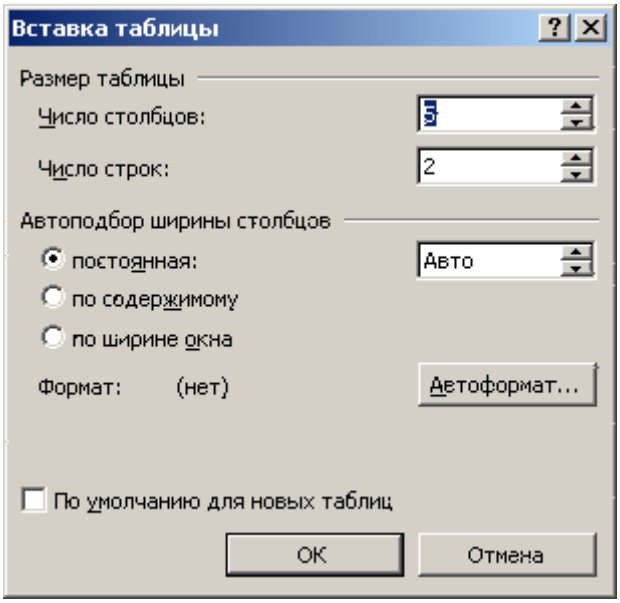


*Рис. 39. Окно диалога **Вставка таблицы***

В созданной таблице возможна обработка значений ее ячеек. Для этого выполняется команда меню **Таблица/Формула**. В результате появляется окно диалога Формула (рис. 40), в котором можно выбрать нужную математическую функцию и указать, для каких ячеек ее следует применить.

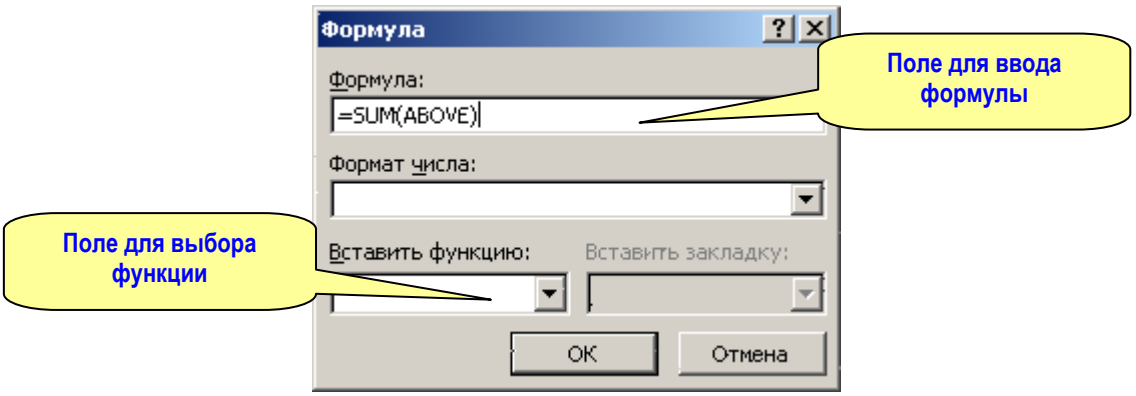


Рис. 40. Окно диалога ***Формула***

Выбрав нужные параметры для вычислений, можно их выполнить и получить в указанной ячейке результат. Например, если требуется просуммировать значения ячеек в столбце, то нужно поместить курсор в пустую ячейку этого столбца под заполненными ячейками. Затем выполнить пункт меню **Таблица/Формула,** выбрать в диалоге функцию суммирования SUM с аргументом ABOVE (SUM[ABOVE]) и запустить на выполнение. Результат тотчас же появится в ячейке, где расположен курсор.

Встраивание иллюстраций

Часто для придания текстовому документу большей наглядности в него вставляют либо готовые иллюстрации из предоставляемого набора стандартных изображений, либо иллюстрации или графики, изготовленные в других графических приложениях Windows. Делается это выполнением команды меню **Вставка/Рисунок.** При этом появляется меню следующего уровня, в котором предлагаются различные варианты вставки рисунка (см. рис. 41).

Для решения указанных выше задач используются команды из меню с вариантами вставки рисунка: **Вставка/Рисунок/Картинки…** и **Вставка/Рисунок/Из файла…**.

При выборе первой из команд открывается секция *Вставка картинки* в панели области задач. Для доступа к коллекции картинок следует щелкнуть по ссылке **Коллекция картинок**, расположенной в разделе *См. также***.** Это приводит к появлению окна *Коллекция картинок*, содержащего панели для навигации по коллекции и панель для отображения выбранного раздела коллекции (рис. 42)

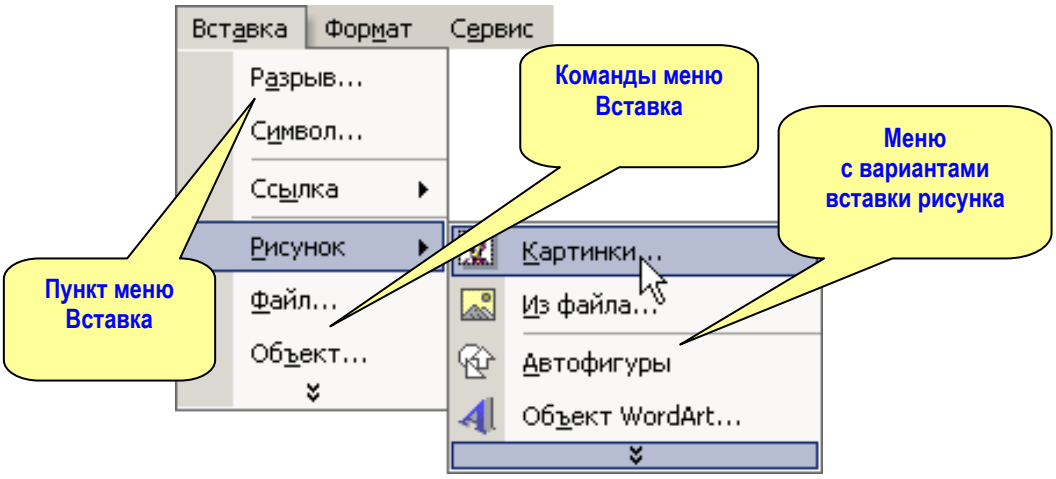


Рис. 41. Выпадающие меню при вставке рисунка

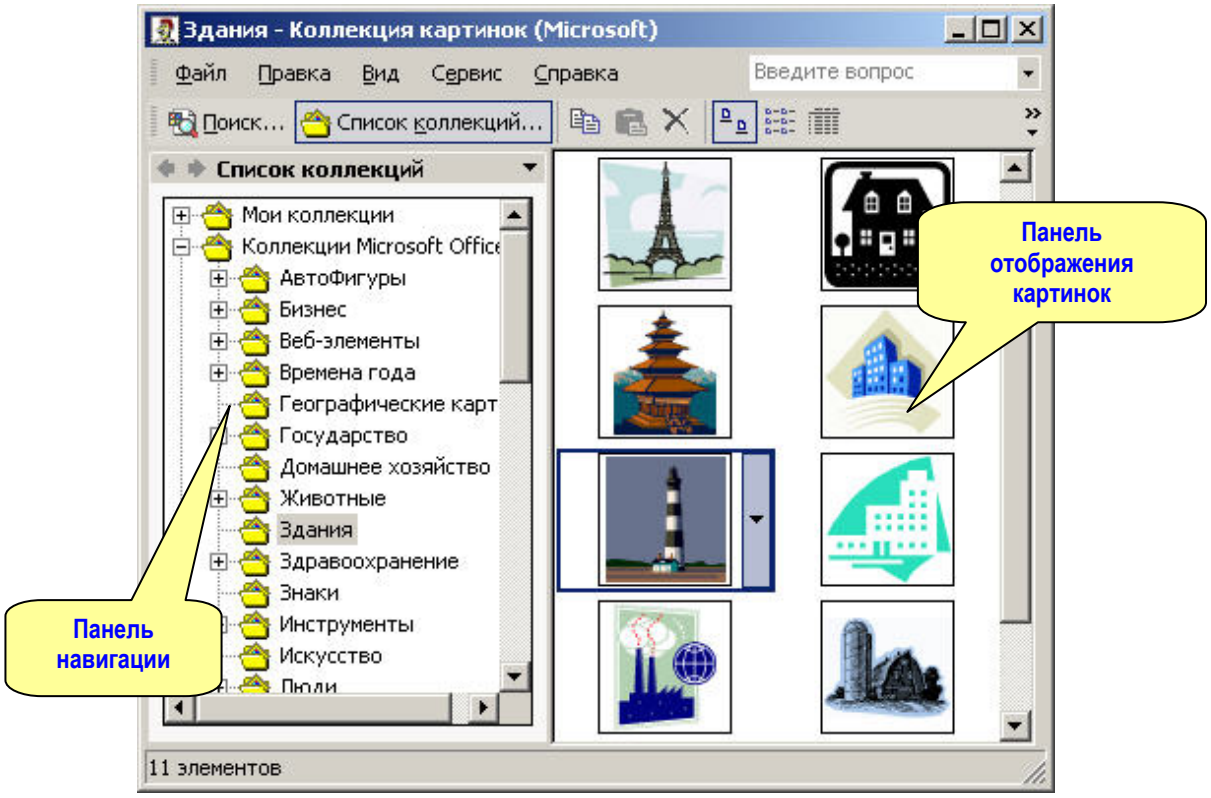


Рис. 42. Окно Коллекция картинок

При подведении указателя мыши в правой панели окна к выбранному рисунку, вокруг него появляется рамка и кнопка выпадающего списка. Щелчок по этой кнопке приводит к появлению контекстного меню (рис. 43), в котором затем нужно выбрать команду **Копировать** для помещения копии картинки в буфер обмена. Из буфера обмена картинка вставляется в документ в месте расположения курсора с помощью команды **Правка/Вставить**.

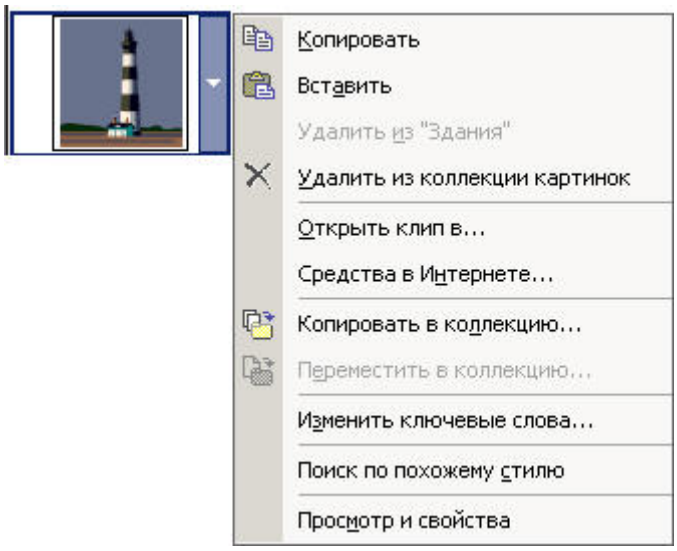


Рис. 43. Выбранная картинка с контекстным меню

Для вставки рисунка из файла нужно выполнить команду меню **Вставка/Рисунок/Из файла….** Это приведет к появлению окна диалога, средствами которого отыскивается и выделяется файл, содержащий вставляемый рисунок. После щелчка в окне по кнопке *Вставить* рисунок появится в документе в месте расположения курсора.

Работа с многостраничными документами

При работе с многостраничными документами возникает много проблем, затрудняющих ориентацию в большом объеме документа. Средством решения этих проблем служат *Вставка колонтитулов* и *Создание оглавления*.

Вставка колонтитулов

Колонтитулами называют области, расположенные в верхнем и нижнем полях страницы. Они содержат краткие сведения о документе, представленные в определенном формате, и помогают выделить различные части документа. При вставке колонтитулов можно:

- создавать колонтитулы для первой страницы документа, отличающиеся от колонтитулов остальных страниц;
- создавать различные колонтитулы для нечетных и четных страниц;
- использовать разрывы разделов и создавать отличающиеся разделы документа с различными колонтитулами.

Работа с колонтитулами выполняется в областях документа, отделенных от основного тела документа. Они называются областями колонтитулов.

Рис. 44. Области колонтитулов

Чтобы получить доступ к этим областям и добавить нужные сведения, выполняют команду меню **Вид/Колонтитулы**. Когда это выполняется в первый раз, MS Word открывает пустой верхний колонтитул и помещает в него курсор.

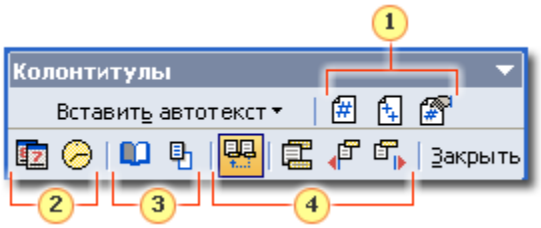


*Рис. 45. Панель инструментов **Колонтитулы***

Появляется панель инструментов **Колонтитулы**, представленная здесь в виде двух рядов кнопок (по умолчанию она появляется как один ряд кнопок).

1. Кнопки нумерации страниц.
2. Кнопки даты и времени.
3. Кнопки макета страницы.
4. Кнопки макета колонтитулов.

Если нужно добавить текст в колонтитул, можно просто ввести его в области соответствующего колонтитула. Другие сведения можно добавить с помощью панели инструментов *Колонтитулы*. Панель инструментов *Колонтитулы* – это особый тип панели инструментов: она отображается только во время работы с колонтитулами. Чтобы вернуться к основному документу, эту панель инструментов необходимо закрыть.

Содержимое колонтитулов можно отформатировать так же, как и любой другой текст (например, меняя размер или цвет шрифта и центрируя текст на странице).

Чаще всего область нижнего колонтитула используется для нумерации страниц, а область верхнего колонтитула – для дополнительной информации (например, название раздела или главы документа в колонтитулах нечетных страниц, а подраздела в колонтитулах четных страниц).

В длинных документах для различных частей документа часто требуются различные колонтитулы. Разные колонтитулы для разных разделов не только хорошо смотрятся, но и позволяют читателю легче ориентироваться в документе.

Чтобы настроить различные колонтитулы, сначала нужно разделить документ на *разделы*. Разделы − это отдельные части документа, которые могут быть по-разному отформатированы.

Для создания разделов необходимо вставить *разрывы разделов* между различными частями документа. Это можно сделать с помощью команда меню **Вставка/Разрыв**.

Разбив документ на разделы, можно создать различные колонтитулы разделов. Для этого понадобится разорвать *связи* между колонтитулами соседних разделов с помощью кнопки *Как в предыдущем*  панели инструментов *Колонтитулы*. Эта кнопка действует как переключатель, но название кнопки не меняется с изменением ее состояния.

О том, включена или разорвана связь, можно легко узнать по надписи над областью соответствующего колонтитула. Если связь включена, область колонтитула совпадает с областью соответствующего колонтитула предыдущего раздела, а в верхнем левом углу области колонтитула появляется надпись «Как в предыдущем».

Примечание. Кнопка *Как в предыдущем* доступна только тогда, когда документ разбит на разделы.

Создание оглавления

Оглавление или содержание дает читателям представление об основных темах, рассматриваемых в документе, и позволяет лучше ориентироваться и быстрее перемещаться в документе.

Для создания оглавления в MS Word выполняются два шага:

1. *Маркировка* в документе текста, который требуется включить в оглавление. Чаще всего для маркировки используют форматирование заголовков с помощью определенных в шаблоне документа стилей заголовков.
2. Сбор маркированного текста в одном месте.

Текст можно маркировать в процессе ввода. В этом случае не надо проходить по всему документу для маркировки элементов текста, которые требуется включить в оглавление (хотя возможен и такой вариант действий).

Использование для форматирования документов встроенных стилей заголовков MS Word является самым простым способом маркировки текста. Просто отформатируйте текст одним из девяти предопределенных стилей заголовков и создайте оглавление.

Простота и быстрота применения являются значительными преимуществами метода, использующего стили заголовков.

После того как текст промаркирован, его можно собрать вместе в оглавлении. Эта операция выполняется процессором MS Word. Для этого сначала нужно установить курсор в том месте, где должно выводиться оглавление, обычно это начало документа. Далее нужно выполнить команду меню **Вставка/Ссылка/Оглавление и указатели** и в появившемся окне диалога *Оглавление и указатели* открыть вкладку *Оглавление*.

Если можно использовать параметры по умолчанию, следует нажать кнопку **OK**, чтобы создать оглавление.

Табличные процессоры

Нет, пожалуй, такой области человеческой деятельности, где не использовалось бы табличное представление информации. Это один из наиболее распространенных способов представления разнообразной информации, при котором достигаются большая наглядность и упорядоченность. А такие свойства информации очень повышают полезность и ценность ее использования.

Табличный процессор (электронные таблицы) предназначен для решения проблем, связанные с обработкой и использованием табличной информации. Перечислим основные возможности процессора.

Первое – и самое важное: процессор позволяет выполнять разнообразные инженерные, научные и экономические расчеты по заданным формулам. Кроме того, с помощью табличного процессора по введенным или рассчитанным в таблице данным можно построить графики и диаграммы. Но не только это. Процессор предоставляет обширный набор средств, позволяющих выполнять действия с матрицами, комплексными числами, которые постоянно применяются в электротехнических расчетах. Он позволяет моделировать возможные последствия изменения значений параметров, характеризующих исследуемый процесс или объект. Дает возможность создавать небольшие базы данных и средства работы с нею. Словом, табличный процессор – это *инструмент для решения широкого круга задач обработки любой информации, представляемой в табличном виде, и отображения полученных результатов.*

Наиболее широкое применение табличные процессоры находят:

- при проведении однотипных расчетов (экономических, научно-технических и др.) над большими наборами данных;
- в автоматизации итоговых вычислений;
- в решении задач методом «подбора параметров»;
- в обработке результатов экспериментов;
- в решении задач оптимизации;
- в подготовке табличных документов;
- при построении диаграмм и графиков по табличным данным.

Все дальнейшее относится к табличному процессору Microsoft Excel.

Создание документа MS Excel

Точно так же, как и в случае текстового процессора, при открытии MS Excel автоматически создается документ самого общего вида, соответствующий основному шаблону табличного документа **Книга**. Тем не менее, табличный процессор предоставляет иные возможности для создания нового документа желаемого вида. Речь идет о шаблонах. Использование шаблонов позволяет создать документ с набором стилей оформления, соответствующих характеру документа. Для доступа к соответствующим средствам создания нужно выполнить команду меню **Файл/Создать…** Это приводит к открытию секции **Создание книги** в панели области задач (рис. 46).

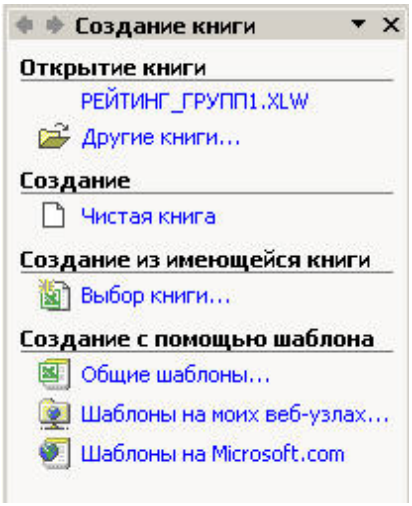


*Рис. 46. Фрагмент секции **Создание книги** области задач, обеспечивающий доступ к средствам создания книги*

Раздел *Создание* в панели области задач дает возможность создать табличный документ самого общего вида. Для этого нужно щелкнуть в упомянутой секции по ссылке **Чистая книга**.

Раздел *Создание с помощью шаблона* дает возможность создать документ в тех случаях, когда он имеет какую-то специфику. В таком случае нужно щелкнуть по ссылке **Общие шаблоны** в разделе *Создание с помощью шаблона***.** В появившемся стандартном окне диалога *Шаблоны* (рис. 47) можно отыскать подходящий шаблон и воспользоваться им. В созданном документе для работы предоставляются уже готовые настройки использованного шаблона.

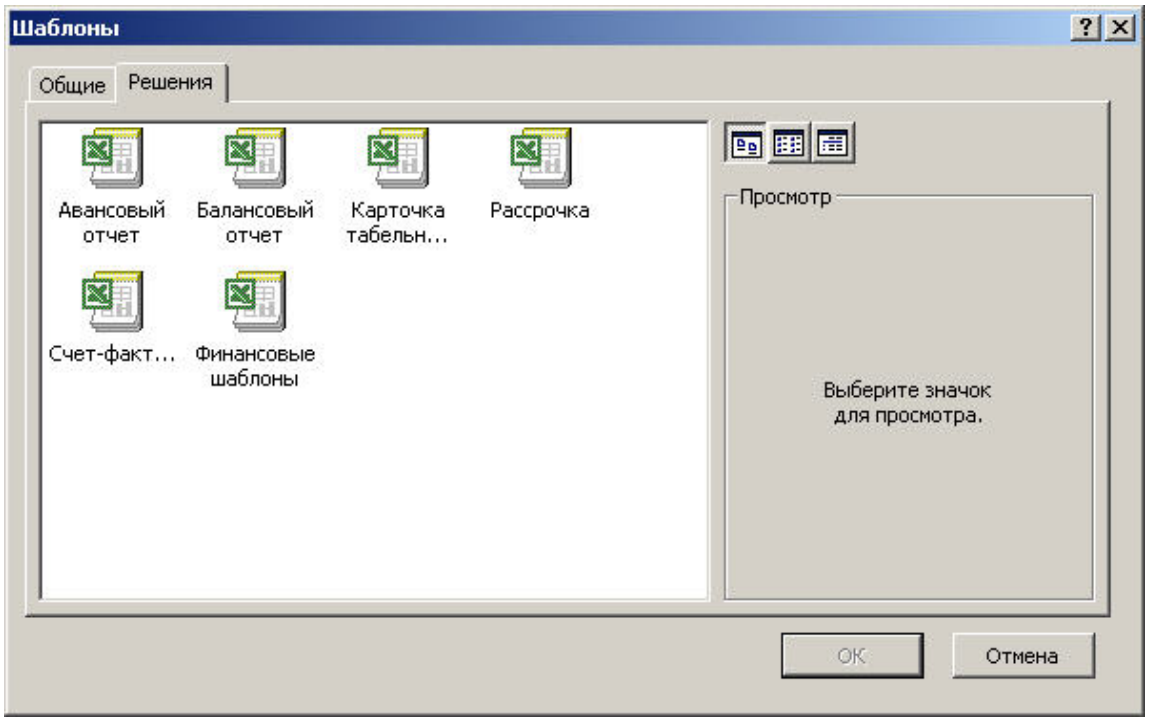


Рис. 47. Окно диалога Шаблоны с шаблонами документов MS Excel

Раздел *Создание из имеющейся книги* дает возможность создать документ, взяв за шаблон уже готовую книгу MS Excel. В такой книге удаляется все ее содержимое, после чего файл документа сохраняется под другим именем и с расширением **.xlt**. Для поиска требуемого файла MS Excel нужно щелкнуть по ссылке **Выбор книги**, открыть окно диалога и выполнить в нем поиск. В этом случае для работы в создаваемом документе предоставляются уже готовые настройки использованного документа.

Сохранение документа MS Excel

По умолчанию создаваемый документ MS Excel имеет имя *Книга1*, которое, во избежание возможных недоразумений при дальнейшей работе, следует изменить. Чтобы сохранить файл под новым именем, следует выполнить команду меню **Файл/Сохранить как...** При этом открывается стандартное окно диалога *Сохранение документа* (рис. 24).

Работа с этим окном одинакова для всех приложений Windows и описана ранее при рассмотрении общих возможностей офисных средств.

Специфика сохранения файла MS Excel проявляется в том, что в поле *Тип Файла* по умолчанию предлагается тип *Книга Microsoft Excel (*.xls)*. Здесь .xls – стандартное расширение файла, создаваемое процессором MS Excel .

Если сохраняемый документ предназначен для работы с ним в предшествующих версиях текстового процессора MS Excel или вообще в иных средах, то для сохранения файла следует выбрать в этом поле из предлагаемого списка нужный тип документа.

Процессор MS Excel при сохранении рабочей книги записывает в файл только прямоугольную область рабочих листов, примыкающую к левому верхнему углу и содержащую все заполненные ячейки.

Основные понятия

Рабочая книга и рабочие листы

Документ MS Excel называется *рабочей книгой.* Рабочая книга представляет собой набор пустых *рабочих листов,* каждый из которых имеет табличную структуру. В окне документа MS Excel отображается только *текущий* рабочий лист, с которым и ведется работа (рис. 48).

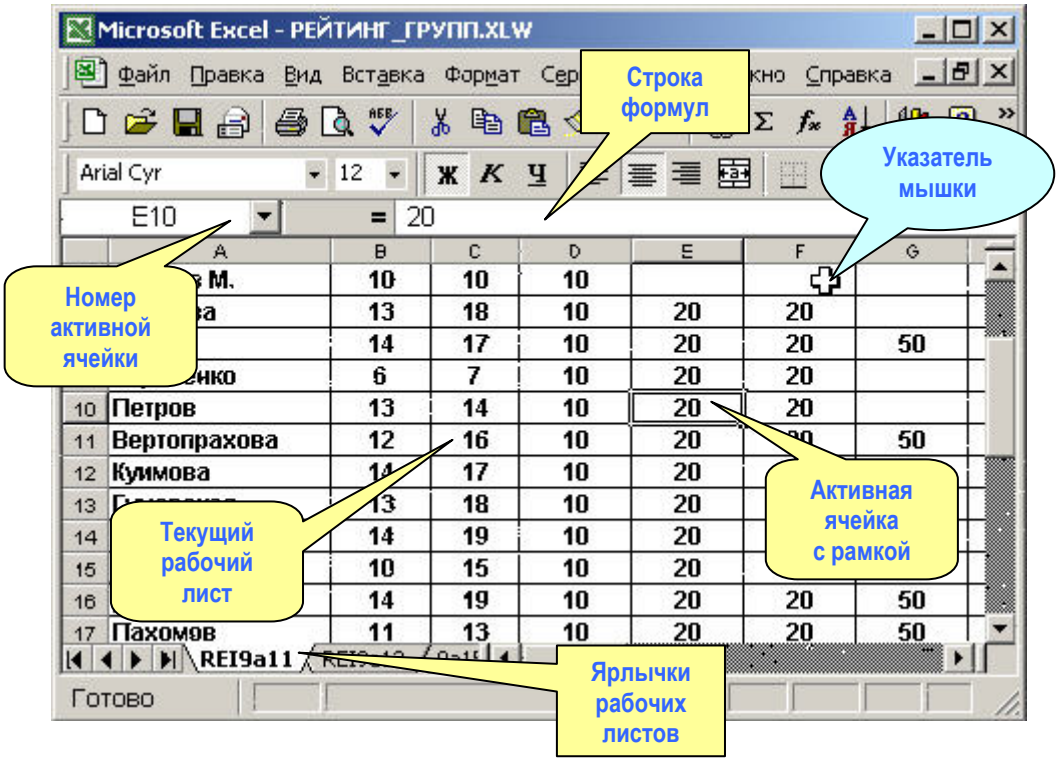


Рис. 48. Рабочий лист табличного процессора MS Excel

Каждый рабочий лист имеет *имя*. Имена листов находятся на ярлычках, расположенных в нижней части окна книги. Для перехода с одного листа на другой необходимо щелкнуть на соответствующем ярлычке. Название текущего листа всегда выделено жирным шрифтом. Листы можно переименовывать, вставлять, удалять, перемещать или копировать в пределах одной книги или из одной книги в другую. Это можно сделать с помощью команд контекстного меню, которое вызывается щелчком правой кнопки на ярлычке листа (рис. 49).

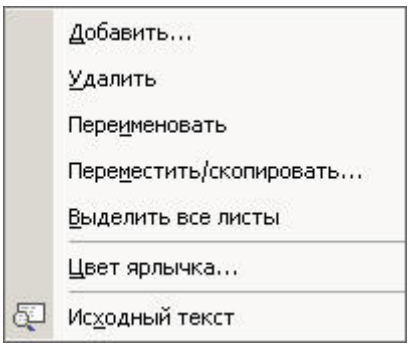


Рис. 49. Контекстное меню для ярлычка листа

Рабочий лист состоит из *ячеек*, организованных в столбцы и строки. Столбцы озаглавлены прописными латинскими буквами и, далее, двухбуквенными комбинациями. Столбцы, расположенные за столбцом с именем «Z», имеют имена AA, AB, AC, …AZ, BA, BB, … и т. д. Строки нумеруются в порядке возрастания, начиная с единицы.

Ячейки. Адресация ячеек

В ячейках хранятся данные. Отдельная ячейка может быть пустой либо содержать данные, относящиеся к одному из трех типов: *текст, число* или *формула*. *Формула* – это описание действий над ячейками, входящими в нее. Результатом этих действий является значение ячейки, в которой записана формула.

Одна из ячеек всегда является активной (текущей) и выделяется рамкой активной ячейки. Эта рамка в процессоре Excel играет роль курсора. Переместить рамку активной ячейки можно с помощью курсорных клавиш или указателя мыши. Операции ввода и редактирования всегда производятся в активной ячейке.

Каждая ячейка имеет свое обозначение (имя, идентификатор). Обозначение отдельной ячейки включает в себя имя столбца и номер строки, на пересечении которых она расположена. Этот порядок в обозначении обязателен, например: D1 или OE234. Имя столбца и номер строки в совокупности однозначно идентифицируют ячейку, которая им

(одновременно) принадлежит. Этот идентификатор называется *адресом* ячейки или *ссылкой* на ячейку.

Ячейкам, кроме того, можно присваивать собственные *имена* и использовать эти имена для решения задач наряду с адресами.

Кроме понятия ячейки, используется понятие *диапазона* (или интервала) ячеек – прямоугольной области смежных ячеек. Диапазон задается указанием адреса верхней левой и правой нижней ячеек, разделенных символом двоеточия.

Адреса ячеек используются при записи формул.

Ввод, редактирование и форматирование данных

Тип данных, размещаемых в ячейке, определяется автоматически при вводе. Если эти данные можно интерпретировать как число, процессор MS Excel так и делает. В противном случае данные рассматриваются как текст. Если текст начинается со знака «=», то MS Excel воспринимает его как формулу.

Ввод текста и чисел

Ввод данных осуществляют непосредственно в текущую ячейку или в *строку формул,* располагающуюся в верхней части окна программы непосредственно под панелями инструментов. Место ввода отмечается текстовым курсором. Если начать ввод с клавиатуры, то данные в текущей ячейке заменяются вводимым текстом. Если щелкнуть на строке формул или дважды на текущей ячейке, старое содержимое ячейки не удаляется и появляется возможность его редактирования. Вводимые данные в любом случае отображаются как в ячейке, так и в строке фор-

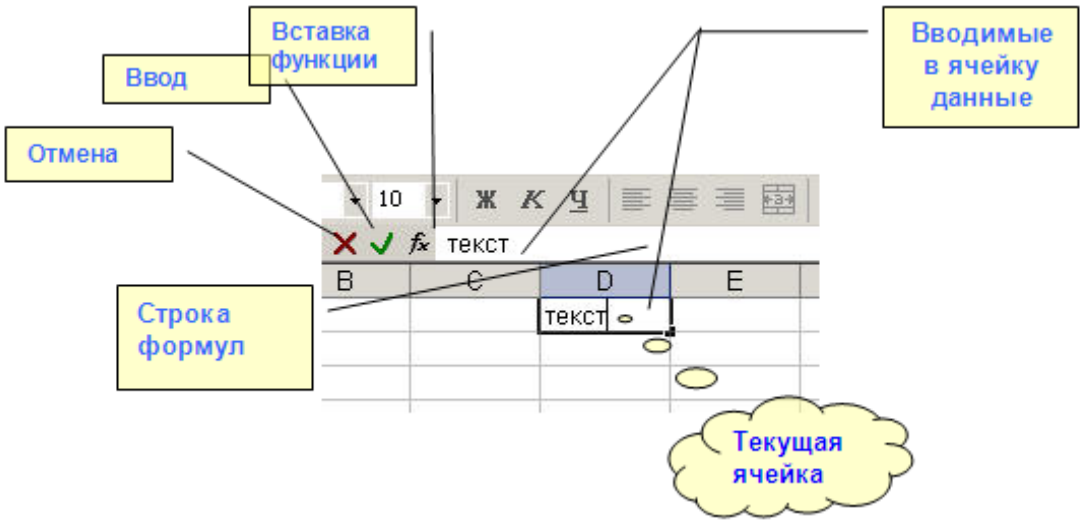


Рис. 50. Ввод данных в текущую ячейку MS Excel

мул (рис. 50). Чтобы завершить ввод, сохранив введенные данные, используют кнопку *Ввод* в строке формул или клавишу **Enter**. Чтобы отменить внесенные изменения и восстановить прежнее значение ячейки, используют кнопку *Отмена* в строке формул или клавишу **Esc**. Для очистки текущей ячейки или выделенного диапазона проще всего использовать клавишу **Delete**.

Форматирование содержимого ячеек

Текстовые данные по умолчанию выравниваются по левому краю ячейки, а числа – по правому. Чтобы изменить формат отображения данных в текущей ячейке или выбранном диапазоне, используют команду **Формат/Ячейки**. Вкладки этого диалогового окна позволяют выбирать формат записи данных (количество знаков после запятой, указание денежной единицы, способ записи даты и прочее), задавать направление текста и метод его выравнивания, определять шрифт и начертание символов, управлять отображением и видом рамок, задавать фоновый цвет.

Автоматизация ввода данных

Так как таблицы часто содержат повторяющиеся или однотипные данные, то процессор MS Excel содержит средства автоматизации ввода. К числу предоставляемых средств относятся: *автозавершение, автозаполнение данными.*

Автозавершение

Для автоматизации ввода текстовых данных используется метод *автозавершения.* Его применяют при вводе в ячейки одного столбца рабочего листа текстовых строк, среди которых есть повторяющиеся. В ходе ввода текстовых данных в очередную ячейку программа MS Excel проверяет на соответствие введенные символы и символы строк, имеющихся в этом столбце выше. Если обнаружено однозначное совпадение, введенный текст автоматически дополняется. Нажатие клавиши **Enter** подтверждает операцию автозавершения, в противном случае ввод можно продолжать, не обращая внимания на предлагаемый вариант.

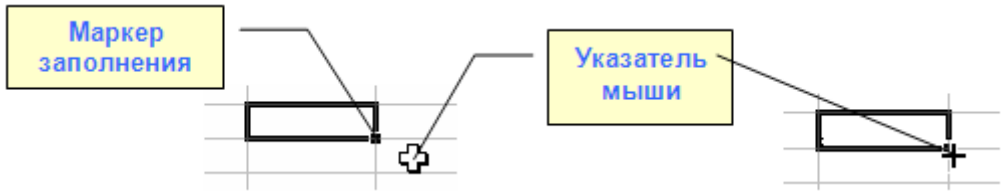
Можно прервать работу средства автозавершения, оставив в столбце пустую ячейку. И наоборот, чтобы использовать возможности средства автозавершения, заполненные ячейки должны идти подряд, без промежутков между ними.

Автозаполнение ячеек данными

При вводе данных может быть использован метод *автозаполнения.* Следует различать несколько возможностей при выполнении этого метода.

В первой из них для заполнения используется содержимое одной текущей ячейки. В правом нижнем углу рамки текущей ячейки имеется черный квадратик – *маркер заполнения*. При наведении на него указатель мыши, который обычно имеет вид контурного белого креста (рис. 51, *а*), приобретает форму тонкого черного крестика (рис. 51, *б*). Эта форма указателя фиксируется нажатием левой клавиши мыши, и при постоянно нажатой клавише выполняется протяжка маркера заполнения.

Протяжка маркера заполнения в горизонтальном или вертикальном направлении приводит к *копированию* содержимого текущей ячейки во все выделенные при перетаскивании ячейки.



а) *б)*

Рис. 51. Изменения указателя мыши при наведении на маркер заполнения

Следующая возможность метода автозаполнения – создавать *стандартные ряды* MS Excel. К ним относятся ряды, у которых значениями элементов являются названия дней недели и месяцев года. Для создания такого ряда достаточно указать в текущей ячейке первое значение (название дня недели или месяца) и выполнить описанную выше протяжку маркера заполнения до тех пор, пока в ячейках не появятся все нужные значения.

а) *б)*

Рис. 52. Начальная и конечная фазы автозаполнения числового ряда

Метод применяется также для быстрого создания числовых рядов арифметической прогрессии. Для автозаполнения числовых рядов нужно выполнить следующие действия:

- набрать в одной ячейке первый элемент ряда,
- в соседней ячейке – второй элемент ряда,
- выделить обе ячейки, для этого указать на первую ячейку, нажать левую клавишу мышки и, не отпуская ее, протянуть до второй ячейки (рис. 52, *а*)
- зафиксировать указатель мыши на маркере заполнения в правом нижнем углу выделения обоих ячеек и протянуть его до тех пор, пока в ячейках не появятся все нужные значения (рис. 52, *б*)

Приемы и средства автоматизации решения задач

Ячейки рабочего листа предназначены для того, чтобы хранить различные значения. Таким образом, ячейка может играть такую же роль, как переменная в математике: она имеет обозначение (имя или адрес) и может иметь и менять значение. Всякое вычисление состоит в том, что по значениям одних переменных вычисляются значения других переменных. Обычно последовательность вычислений описывается с помощью формулы, содержащей математические операции и операнды (переменные, функции). Но запись самой формулы – это строка, т. е. тоже значение, которое можно хранить в ячейке. В этом и состоит основная идея электронных таблиц: одни ячейки рабочего листа используются как независимые переменные (в MS Excel они называются – влияющие ячейки), которым должны быть приданы значения извне. Другие ячейки используются как зависимые переменные (они так и называются в MS Excel – зависимые ячейки), которые содержат формулы, ссылающиеся на независимые переменные. Пользователь вводит исходные данные во влияющие ячейки, автоматически производятся вычисления по формулам, находящимся в зависимых ячейках, и пользователь видит готовый результат вычислений в зависимых ячейках.

В ячейки можно вводить формулы, которые содержат ссылки на ячейки с другими формулами. Это дает ряд преимуществ при автоматизации решения.

- Цепочка зависимостей может быть сколь угодно длинной: одна ячейка зависит от другой ячейки, которая в свою очередь зависит от третьей, и т. д.
- Влияющая ячейка может влиять на множество зависимых ячеек: сколь угодно много формул в разных зависимых ячейках могут ссылаться на одну и ту же влияющую ячейку. Автоматическое вычисление может состоять из многих параллельных шагов.

- Цепочка зависимостей может быть циклической, и для вычисления таких циклических зависимостей могут быть использованы итерационные методы.
- Формула в зависимой ячейке может носить условный характер: если некоторое условие выполняется, то нужно производить вычисления по одной формуле, а если не выполняется, то по другой.
- Формула может содержать сколь угодно сложные функции, а не только арифметические операции.

Формулы и их использование

Главная особенность табличного процессора – возможность использования формул для автоматизации процессов решения многих задач.

Формула описывает зависимость значений одной ячейки от значений других ячеек. Расчет по заданной формуле выполняется автоматически.

Изменение содержимого какой-либо ячейки приводит к пересчету значений всех ячеек, которые связаны с ней формулой и, тем самым, к обновлению всей таблицы в соответствии с изменившимися данными.

В формулу могут входить следующие элементы:

- знаки операций;
- адреса ячеек;
- значения (числа, строки);
- стандартные функции Excel.

Если ячейка содержит формулу, то на рабочем листе в ней отображается текущий результат вычисления этой формулы. Если сделать ячейку текущей, то сама формула отображается в строке формул.

Замечание. Если значение ячейки **действительно** зависит от других ячеек таблицы, то **всегда** следует использовать формулу, даже если операцию легко можно выполнить в «уме».

Такой прием обеспечивает правильность вычислений в таблице при последующем ее редактировании, поскольку значения в ячейке с формулой автоматически пересчитываются при изменении значений в ячейках, входящих в нее.

Для правильного создания формулы необходимо иметь представление об адресах ячеек и использовании функций для упрощения формул.

Создание в формулах ссылок на ячейки

Формула может содержать *ссылки*. *Ссылка* – это адрес ячейки, содержимое которой используется при вычислениях в формуле.

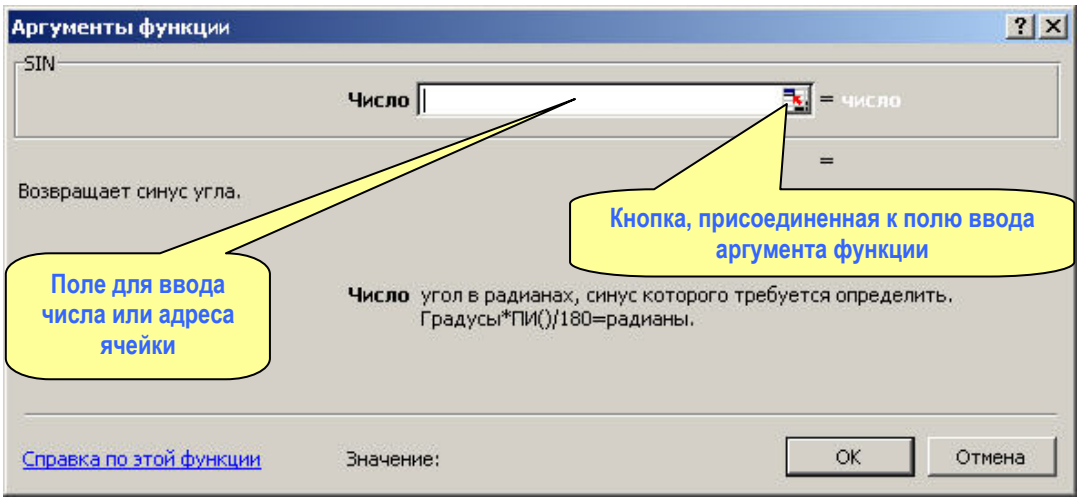
Ячейка, содержащая формулу со ссылками на другие ячейки, таким образом, является *зависимой.* Значение, отображаемое в ячейке с

формулой, пересчитывается при изменении значения ячейки, на которую указывает ссылка.

На данные, расположенные в диапазоне ячеек (соседних ячейках, образующих прямоугольную область) можно ссылаться в формулах, как на единое целое.

Ссылку на ячейку можно задать разными способами. Во-первых, адрес ячейки можно ввести вручную. Другой способ состоит в щелчке мышкой на нужной ячейке или выборе мышкой диапазона, адрес которого требуется ввести. Ячейка или диапазон при этом выделяются пунктирной рамкой.

Все диалоговые окна программы MS Excel, которые требуют указания адресов отдельных ячеек или их диапазонов, содержат кнопки, присоединенные к соответствующим полям (рис. 53, *а*).



а)

б)

Рис. 53. Окна диалога для создания ссылки на нужную ячейку или диапазон ячеек: а) – развернутое, б) – свернутое

При щелчке на такой кнопке диалоговое окно сворачивается до минимально возможного размера (рис. 53, *б*), что облегчает нужный выбор данных с помощью щелчка для отдельной ячейки или выделения для диапазона ячеек.

Если требуется выделить прямоугольный диапазон ячеек, это можно сделать протягиванием указателя от одной угловой ячейки до противоположной по диагонали. Рамка текущей ячейки при этом расширяется, охватывая весь выбранный диапазон. Чтобы выбрать столбец

или строку целиком, следует щелкнуть на заголовке столбца (строки). Протягиванием указателя по заголовкам можно выбрать несколько идущих подряд столбцов или строк.

Для редактирования формулы следует дважды щелкнуть на соответствующей ячейке. При этом ячейки (диапазоны), от которых зависит значение формулы, выделяются на рабочем листе цветными рамками, а сами ссылки отображаются в ячейке и в строке формул тем же цветом. Это облегчает редактирование и проверку правильности формул.

Виды ссылок

Ссылки бывают *относительными и абсолютными*.

Относительные ссылки ячеек записываются по правилу *<имя столбца> <номер строки>*, например: A1. Такая ссылка означает, что при копировании формулы адреса в ссылках автоматически изменяются в соответствии с относительным расположением исходной ячейки и создаваемой копии.

По умолчанию при наборе формул в MS Excel используются **относительные ссылки**.

Пусть, например, в ячейке B2 имеется ссылка на ячейку A3. В относительном представлении можно сказать, что ссылка указывает на ячейку, которая располагается на один столбец левее и на одну строку ниже данной ячейки. Если формула будет скопирована или перемещена в другую ячейку, то такое относительное указание ссылки сохранится. Например, при копировании (перемещении) формулы в ячейку E7 ссылка будет продолжать указывать на ячейку, располагающуюся левее и ниже, в данном случае на ячейку D8.

Абсолютные ссылки ячеек записываются по правилу *<**$**имя столбца><**$**номер строки>*, например: **A**1. Такая ссылка указывает на совершенно конкретную ячейку. При перемещении или копировании абсолютные ссылки в формулах не изменяются.

Кроме этого, можно использовать смешанные ссылки, например: A$1 или $A1. Часть ссылки, не содержащая знак «$», будет обновляться при копировании, а другая часть, со знаком «$», останется без изменения.

Ниже приведены правила обновления ссылок при копировании вправо или вниз.

Ссылка в исходной ячейке	Ссылка в копии ячейки	
	справа	внизу
A1(относительная)	**B1**	**A2**
$A1 (абсолютная по столбцу)	**$A1**	**$A2**
A$1 (абсолютная по строке)	**B$1**	**A$1**
A1 (абсолютная)	**A1**	**A1**

Ввод формулы

Формула записывается либо непосредственно в активную ячейку либо, после выделения такой ячейки, в строку формул. Ввод обязательно начинается со знака равенства «=».

Процессор содержит большое количество стандартных функций, которыми можно пользоваться при наборе формул.

Чтобы использовать эти функции при вводе формулы, нужно позиционировать курсор в нужном месте формулы и выполнить команду меню **Вставка/Функция.** Это приведет к появлению окна диалога *Мастер функций* (рис. 54), в котором все функции разбиты по категориям и именованы.

Замечание. Для работы с функциями категории *Инженерные* должна быть подключена надстройка «Пакет анализа». Подключение выполняется в окне диалога *Надстройки***,** которое появляется после выполнения команды меню **Сервис/Надстройки**.

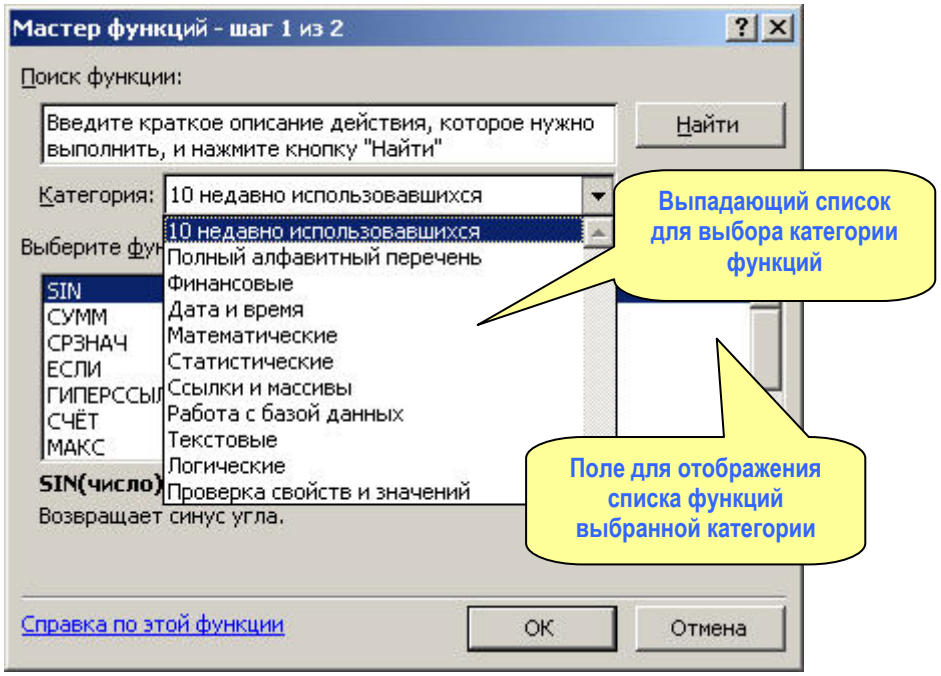


Рис. 54. Окно диалога ***Мастер функций***

В появившемся окне диалога в поле *Категория* следует щелкнуть по кнопке выпадающего списка и выбрать в списке категорию функции, в поле *Выберите функцию* выбрать имя требуемой функции. После выбора появится окно диалога для функции, с помощью которого нужно ввести числа, адреса ячеек в поля аргументов.

Если аргументами служат адреса ячеек с данными, их можно ввести, щелкнув по нужной ячейке или выделив с помощью крестообразного указателя мыши нужный диапазон ячеек пунктирной рамкой. Для удобства

ввода аргументов функций ее окно диалога можно сдвинуть или свернуть кнопкой, присоединенной к полю ввода аргументов (рис. 53).

Табличный процессор предоставляет функции для выполнения расчетов с матричными данными, комплексными числами, для выполнения финансовых и экономических расчетов и т. д. Более подробную информацию об использовании конкретных функций можно получить из справочной системы MS Excel.

Автозаполнение формулами

Эта операция выполняется так же, как автозаполнение данными. Для копирования формулы достаточно набрать в одной ячейке нужную формулу и далее протащить мышкой маркер заполнения до нужной ячейки. Особенность этой операции заключается в необходимости копирования ссылок на другие ячейки. В ходе автозаполнения учитывается характер ссылок в формуле: относительные ссылки изменяются в соответствии с относительным расположением копии и оригинала, абсолютные остаются без изменений.

Для примера предположим, что значения в третьем столбце рабочего листа (столбце **C)** вычисляются как суммы значений в соответствующих (расположенных в этой же строке) ячейках столбцов **A** и **B**. Введем в ячейку **C2** формулу **=A2+B2**. Теперь скопируем эту формулу методом автозаполнения в несколько последующих ячеек третьего столбца таблицы. Благодаря относительной адресации формула будет правильной для всех ячеек данного столбца, то есть для каждой ячейки столбца **C** будет вычисляться сумма значений соответствующих ячеек столбцов **A** и **B**. Ниже (рис. 55) приведен фрагмент таблицы MS Excel, иллюстрирующий вышесказанное. В выделенной ячейке **C4** отображено значение суммы чисел, расположенных в ячейках слева, т. е. **A4+B4.** Саму скопированную формулу можно увидеть в строке формул.

C4 f_x =A4+B4

	A	B	C
1	1-е слагаемое	2-е слагаемое	Сумма
2	1	1	2
3	2	3	5
4	5	8	13
5	13	21	34

Рис. 55. Обновление относительных ссылок

Автоматизация решения расчетных задач

С помощью таблиц MS Excel можно выполнять весьма сложные инженерно-технические расчеты, даже не вдаваясь в математические тонкости применяемых методов и средств. Хотя, разумеется, с хорошей математической подготовкой значительно легче понимать и осваивать средства MS Excel, предназначенные для расчетных задач. Мы рассмотрим некоторые классы задач, часто встречающиеся в инженерной практике, которые можно решать средствами процессора MS Excel.

Численное решение уравнений с одним неизвестным

Процессор MS Excel позволяет находить приближенные корни произвольных алгебраических уравнений. При этом важно только, чтобы корень действительно существовал. Решение выполняется численным способом с использованием команды меню **Сервис/Подбор параметра.**

Предположим, что требуется отыскать корень следующего нелинейного алгебраического уравнения:

$$\left(2 \cdot x^2 + 3\right) \cdot \left(1 - \sin\left(x\right)\right) = \ln\left(x\right) + 2$$

Для нахождения корня этого уравнения преобразуем его к виду $f(x) = 0$, перенеся все слагаемые влево. Именно этот прием позволяет успешно применять указанную выше команду меню для решения уравнения. Далее следует в одну ячейку записать в виде формулы левую часть уравнения $f(x) = 0$. В качестве аргумента уравнения используется адрес любой произвольной ячейки таблицы (изменяемая ячейка), в которую помещено произвольное число. Это число должно находиться в области допустимых значений – годится все-таки не совсем произвольное число. Например, в данном случае ноль или отрицательные числа не годятся – они не входят в область допустимых значений.

Теперь можно воспользоваться командой **Сервис/ Подбор параметра** для проведения итерационных вычислений и поиска корня. Для поиска решения команда **Подбор параметра** использует метод итераций (последовательных приближений). Это означает следующее: прежде всего, вычисляется значения функции $f(x)$ для начального значения параметра в изменяемой ячейке. Если установленное начальное значение не приводит при вычислениях к требуемому нулевому значению для функции $f(x)$, то изменяется значение параметра и заново вычисляется функция $f(x)$. Процесс продолжается до тех пор, пока не будет отыскано нужное значение параметра в пределах заданной точности (если, конечно, решение существует).

На рис. 56 приводится вид окна диалога команды **Подбор параметра** с настройками, соответствующими адресу изменяемой ячейки A4 и ячейки B4 с записью формулы для вычисления $f(x)$.

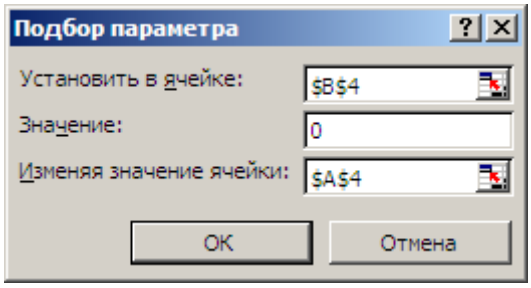


Рис. 56. Окно диалога команды ***Подбор параметра***

Результат применения выбранных настроек приведен на рис. 57. В ячейке A4 отображено значение найденного корня равное 0,635796191206763. Для этого значения корня функция $f(x)$ дает значение −0,0001906, с заданной точностью близкое к нулевому значению.

Решение алгебраического уравнения			
Корень уравнения	Значение		
0,635796191	-0,0001906	<-----формула =	(2*x*x+3)*(1-SIN(x))-LN(x)-2

Рис. 57. Отыскание корня алгебраического уравнения

Поиск оптимального решения

Большая часть задач, с которыми мы сталкиваемся в научной и инженерной деятельности, не очень просты. Зачастую значения формул зависят от большого числа переменных, и при этом чаще всего требуется отыскать наилучшее, оптимальное решение (например, дающее максимальное или минимальное значения), удовлетворяющее при этом целому ряду дополнительных условий на значения используемых параметров. Для решения таких более сложных задач требуется и намного более мощный инструмент. Именно таким могучим оружием для решения сложных задач, требующих применения методов линейного, нелинейного программирования и исследования операций, является имеющаяся в MS Excel надстройка «Поиск решения».

Для того чтобы использовать надстройку «Поиск решения», вовсе не обязательно быть знатоком методов линейного программирования и исследования операций − нужно лишь понимать, какие задачи могут быть решены этими методами. *От пользователя требуется умение с помощью серии диалоговых окон правильно сформулировать условия задачи, и если решение существует, то «Поиск решения» отыщет его.*

В основе надстройки «Поиск решения» также лежат итерационные методы. Однако эта надстройка использует гораздо более сложные

методы, чем рассмотренный ранее подбор параметра. Укажем здесь некоторые отличия этих двух инструментов:

- «Поиск решения» позволяет использовать *одновременно* большое количество (в общей сложности до 200) изменяемых ячеек;
- «Поиск решения» позволяет задавать *ограничения* для изменяемых ячеек. Например, при поиске решения, обеспечивающего максимальную прибыль, вы можете задать дополнительные условия, скажем, потребовать, чтобы при этом общий доход находился в диапазоне между 20 % и 30 % , или чтобы расходы не превосходили 1 000 000 рублей. Подобного рода условия называются *ограничениями* для решаемой задачи.
- «Поиск решения» позволяет отыскать *оптимальное* (минимальное или максимальное) решение, т. е. наилучшее из возможных решений.
- Наконец, для сложных задач «Поиск решения» может генерировать множество различных решений при различных ограничениях.

Задачи, для решения которых можно воспользоваться надстройкой «Поиск решения», имеют ряд общих свойств:

- Имеется единственная ячейка, содержащая формулу, значение которой должно быть сделано максимальным, минимальным или же равным какому-то конкретному значению. Это значение является конечной целью решения, поэтому эта ячейка называется *целевой ячейкой.*
- Формула в этой целевой ячейке содержит ссылки (прямые или косвенные) на ряд *изменяемых ячеек* (содержащих неизвестные, или переменные решаемой задачи).

Поиск решения заключается в том, чтобы подобрать такие значения этих переменных, которые бы давали оптимальное значение для формулы в целевой ячейке. Изменяемые ячейки могут содержать, например, себестоимость или цену товаров, транспортные тарифы или налоговые ставки.

Кроме того, может быть задано некоторое количество *ограничений* – условий или соотношений, которым должны удовлетворять некоторые из изменяемых ячеек.

Рассмотрим использование надстройки **Поиск решения** на примере следующей известной задачи.

Найти значения сторон прямоугольника, имеющего максимальную площадь при условии, что полупериметр прямоугольника равен 100.

Для получения результата нужно заполнить на рабочем листе две ячейки произвольными значениями сторон прямоугольника. Далее, в отдельную ячейку записать формулу для полупериметра прямоугольника, используя адреса ячеек со значениями сторон прямоугольника. За-

тем записать в отдельную ячейку формулу для площади прямоугольника, используя адреса ячеек со значениями его сторон.

Для решения задачи в такой постановке нужно выполнить команду меню **Сервис/Поиск решения.** После появления окна диалога *Поиск Решения* в него следует внести нужные значения и ограничения и щелкнуть по кнопке *Выполнить*. На рис. 58 приведен фрагмент рабочего листа MS Excel с исходными данными для решения и вид окна диалога *Поиск решения* с полями, в которые внесены нужные значения и ограничения.

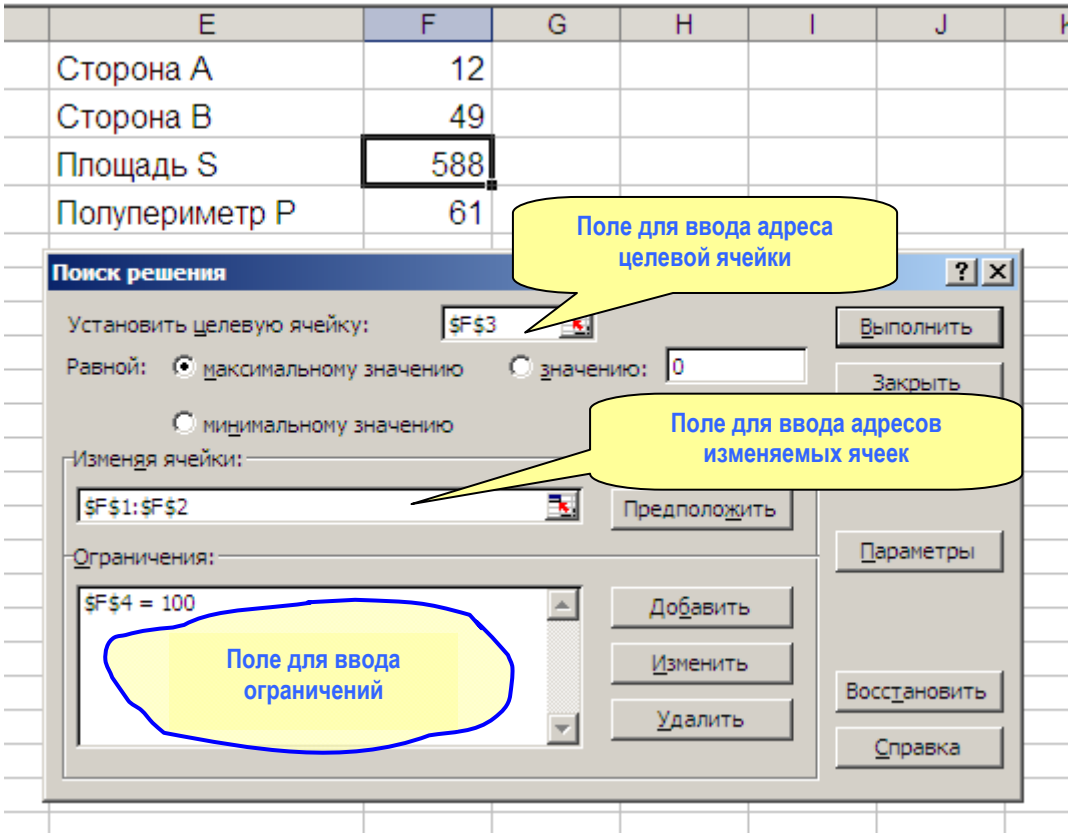


Рис. 58. Окно диалога ***Поиск решения***

На рис. 59 приведен вид окна диалога *Результаты поиска решения* и полученные значения сторон и площади. Как и следовало ожидать, результат решения – квадрат.

В различных радио- и электротехнических расчетах широко используются матрицы. Для работы с матрицами существуют специальные функции. Чтобы их использовать, нужно выполнить пункт меню **Вставка/Функция.** Это приведет к появлению окна *Мастер Функций*, в котором нужно выбрать требуемую матричную функцию. Таких функций несколько: **МУМНОЖ** (умножение матриц), **МОБР** (вычисление обратной матрицы), **МОПРЕД** (вычисление детерминанта), **ТРАНСП** (транспони-

рование матрицы). В аргументах этих функций матрицы обычно указываются своими ячейками: левой верхней и правой нижней. Прежде чем вызывать требуемую функцию, нужно создать исходные матрицы и отметить место под результат с учетом выполняемой операции. Например, при умножении двух матриц результирующая матрица должна иметь столько строчек, сколько их имеет первая из перемножаемых матриц, а ее количество столбцов равно числу столбцов второй матрицы. Для любой операции с матрицей чтобы получить результат, нужно перевести курсор в строку формул и нажать клавиши ***Ctrl / Shift / Enter***.

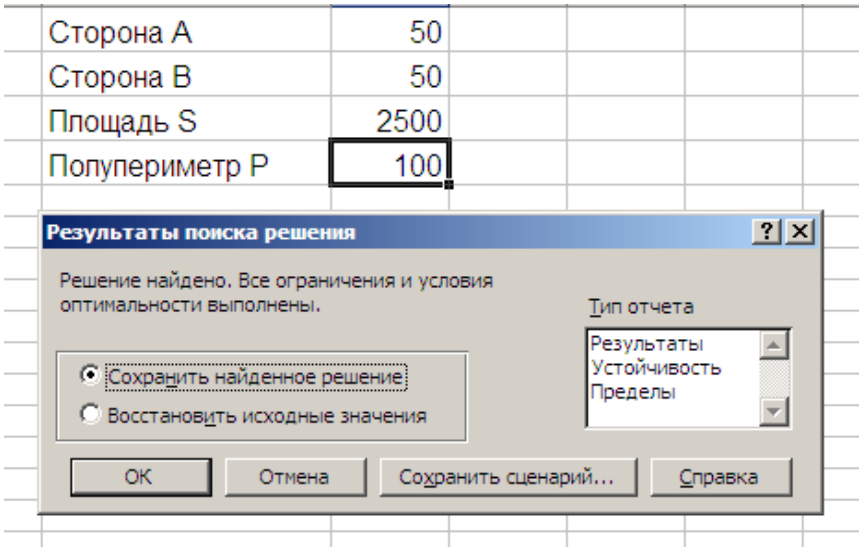


Рис. 59. Окно диалога Результаты поиска решения

Матричные вычисления

В тех случаях, когда нет специальных матричных функций, как, например, при умножении матрицы на скаляр, формулу можно просто набрать в строке формул, указав матрицу ее диапазоном, и нажать клавиши ***Ctrl / Shift / Enter.***

Вычисления с комплексными переменными

Комплексные числа широко используются при расчетах электрических цепей и в других инженерных приложениях. Процессор MS Excel предоставляет всё необходимое для работы с комплексными переменными.

Чтобы ввести в ячейку комплексное число и выполнить над ним требуемые операции, нужно выполнить команду меню **Вставка/Функция**. После этого в окне диалога *Мастер функций* нужно выбрать категорию функции *Инженерные.* Выбор этот дает возможность вводить значения чисел с помощью функции КОМПЛЕКСН. Все функции, с помощью которых можно выполнять действия над комплексными числами, начинаются с

префикса МНИМ. Например, для сложения комплексных чисел есть функция МНИМ, СУММ и т. д. Для получения результата от применения той или иной функции нужно выбрать ее из списка и далее следовать всем указаниям *Мастера функций.*

Замечание. Для работы с инженерными функциями должна быть подключена надстройка «Пакет анализа» (команда меню **Сервис/Надстройки**).

Автоматизация решения информационных задач

К классу информационных задач будем относить задачи обработки табличных данных, не связанные с выполнением математических расчетов. Таких задач в практической деятельности встречается довольно много, но мы остановимся на некоторых из них. Средства автоматизации рассматриваемых далее задач становятся доступными после выполнения команд пункта меню **Данные** (рис. 60).

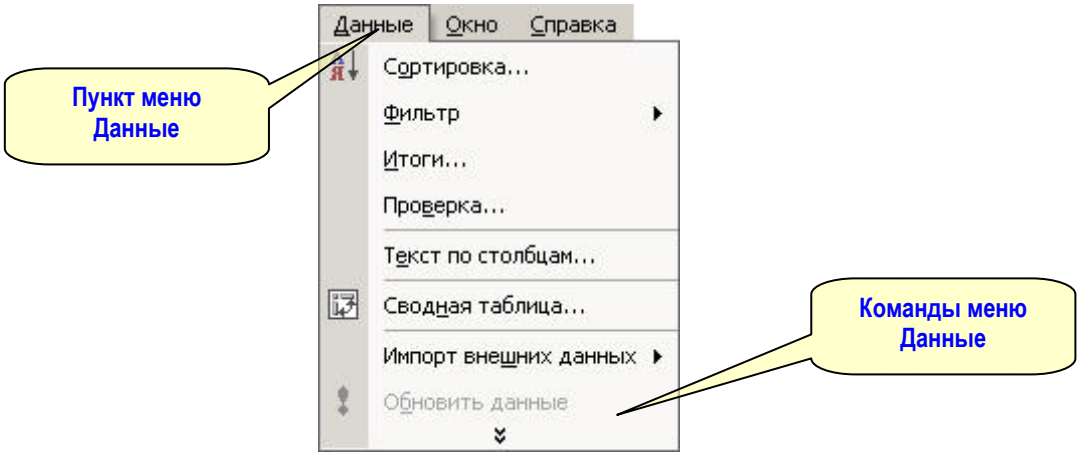


Рис. 60. Раскрытый пункт меню ***Данные***

Сортировка списков

Сортировка – это стандартный способ упорядочивания данных одного типа. Данные можно сортировать по возрастанию или убыванию (в алфавитном порядке, по датам или по величине чисел). Пустые ячейки всегда помещаются в конец списка.

Список – это совокупность строк листа, содержащих однотипные данные, например, имена клиентов и их телефонные номера.

Понятие «списка» может быть отнесено к диапазону ячеек листа. Данные в столбцах списка должны быть одного типа, например, текст в одном столбце и числа – в следующем. Работать со списком легче, если он имеет заголовки столбцов. Строки в списке можно сортировать по значениям ячеек одного или нескольких столбцов. Кроме этого, список можно сортировать по строкам.

Для выполнения сортировки нужно выделить диапазон ячеек, где предполагается такая операция, и выполнить команду меню **Данные/Сортировка**. В открывшемся окне диалога *Сортировка диапазона* устанавливаются параметры сортировки (рис. 61).

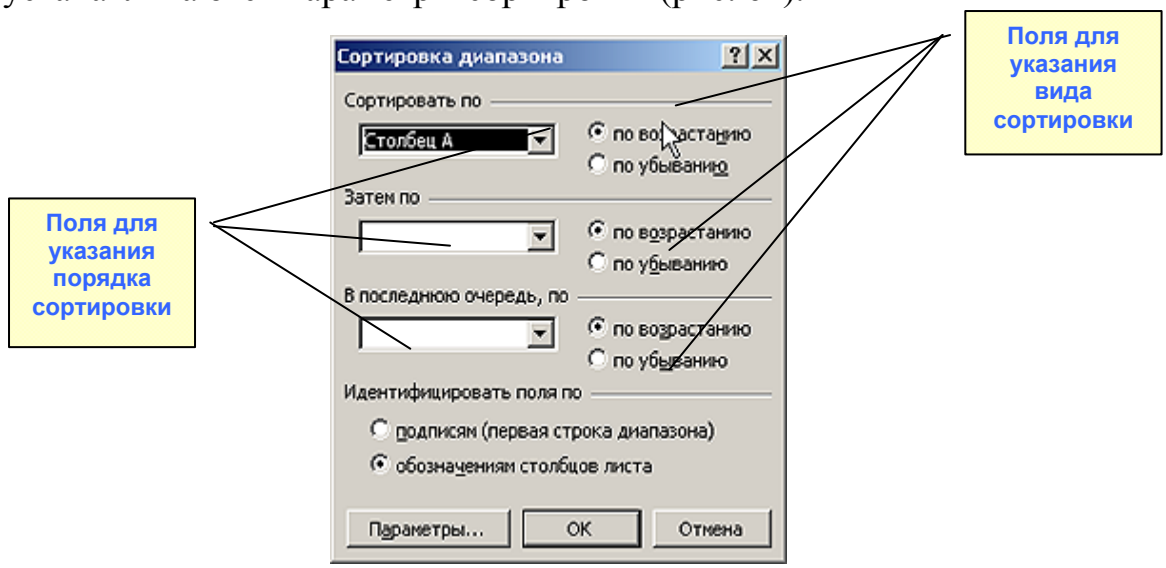


Рис. 61. Окно диалога ***Сортировка диапазона***

Строки, столбцы или отдельные ячейки в процессе сортировки переупорядочиваются в соответствии с заданным пользователем порядком сортировки.

Фильтрация списков

Под *фильтрацией* списков понимается отображение строк таблицы, удовлетворяющих заданным условиям отбора. Например, список покупок, сделанных покупателями, можно отфильтровать так, что на экран будут выведены имена только тех покупателей, которые совершили покупки более, чем на определенную сумму рублей.

В Microsoft Excel доступны два способа фильтрации списков: с помощью *автофильтра* и с помощью *расширенного фильтра*.

При использовании автофильтра нужно щелкнуть кнопкой мыши на любой ячейке фильтруемой таблицы и выполнить команду меню **Данные/Фильтр/Автофильтр**. В заголовках столбцов таблицы появятся кнопки со стрелками. Щелчок по кнопке в любом столбце приводит к выпадению списка с условиями отбора строк, соответствующими фильтруемым ячейкам столбца (рис. 62). Нужное условие выбирается из этого списка.

Автофильтр применяется для того, чтобы быстро отфильтровать данные с одним или двумя условиями, накладываемыми на ячейки отдельного столбца.

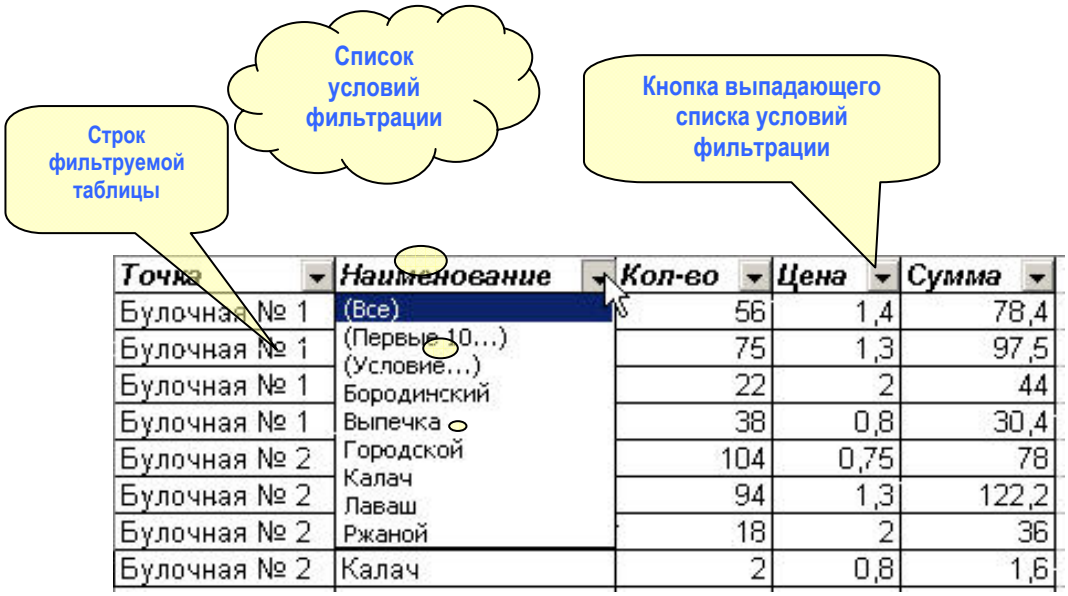


Рис. 62. Вид таблицы с раскрытым списком условий фильтрации

Использование расширенного фильтра имеет свои особенности, которые сводятся к следующему. Условия отбора строк записываются над таблицей, в которой производится фильтрация. Диапазон условий должен включать заголовки столбцов. Между последней строкой диапазона условий и заголовками столбцов используемой таблицы должно быть не менее одной свободной строки.

Пример расположения условий отбора и таблицы приведен на рис. 63. Условия отбора накладываются на столбцы с заголовками ***Точка, Наименование*** и ***Кол-во***.

Точка	***Наименование***	***Дата***	***Кол-во***	***Цена (руб.)***	***Сумма (руб.)***
Булочная					
	Лаваш				
			>50		
Точка	***Наименование***	***Дата***	***Кол-во***	***Цена (руб.)***	***Сумма (руб.)***
Булочная № 1	Городской	10.янв	56	1,4	78,4
Булочная № 1	Ржаной	10.янв	75	1,3	97,5
Булочная № 2	Лаваш	10.янв	22	2	44
Магазин "Хлеб"	Калач	10.янв	38	0,8	30,4
Чайная	Выпечка	10.янв	104	0,75	78
Магазин "Хлеб"	Ржаной	10.янв	94	1,3	122,2
Булочная № 1	Лаваш	10.янв	18	2	36
Булочная № 1	Калач	10.янв	2	0,8	1,6
Булочная № 2	Бородинский	10.янв	65	2,7	175,5
Чайная	Выпечка	11.янв	91	0,75	68,25

Рис.63. Пример расположения условий отбора и таблицы

Для работы с расширенным фильтром нужно щелкнуть кнопкой мыши на любой ячейке фильтруемой таблицы и выполнить команду меню **Данные/Фильтр/Расширенный фильтр**. Это приведет к открытию окна диалога *Расширенный фильтр* (рис. 64). В полях диалога указываются исходный диапазон и диапазон условий. Кроме того, указывается способ отображения результатов обработки.

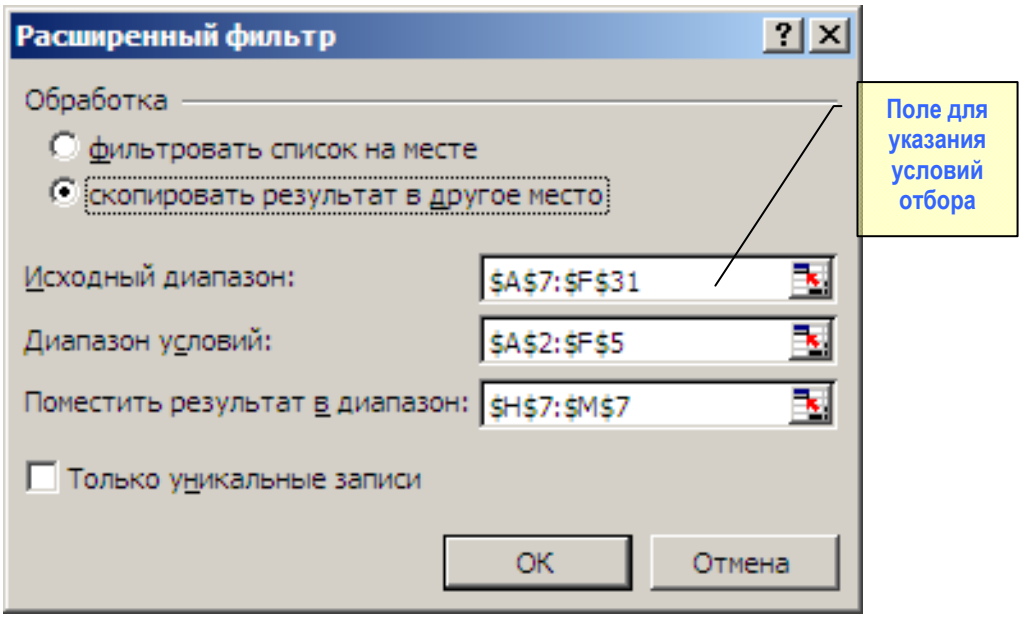


Рис. 64. Окно диалога ***Расширенный фильтр***

Команда **Расширенный фильтр** применяется для того, чтобы отфильтровать данные:

- с условием, накладываемым на ячейки двух столбцов или более;
- с более чем двумя условиями, накладываемыми на ячейки отдельного столбца.

Построение диаграмм

Очень часто в инженерной деятельности при использовании электронных таблиц требуется вставить график (или диаграмму), отображающую табличную информацию. По общему мнению, возможности построения диаграмм в MS Excel (так называемая деловая графика) относятся к числу наиболее сильных сторон этого программного продукта. Существует две возможности осуществить построение – создавать график на том же листе, где отображены нужные данные или же создавать график на отдельном листе.

Для размещения данных и графика на одном листе используют *внедренный график*.

Внедренный график связан с данными таблицы процессора и обновляется автоматически при каждом изменении данных в таблице.

Внедрение графика на текущий лист выполняется следующими действиями:

- выделить столбец значений функции вместе с ее названием (или столбцы, если в таблице представлено несколько функций одного аргумента).
- воспользоваться в меню **Вставка** пунктом **Диаграмма**;
- следовать инструкциям *Мастера диаграмм* (рис. 65).

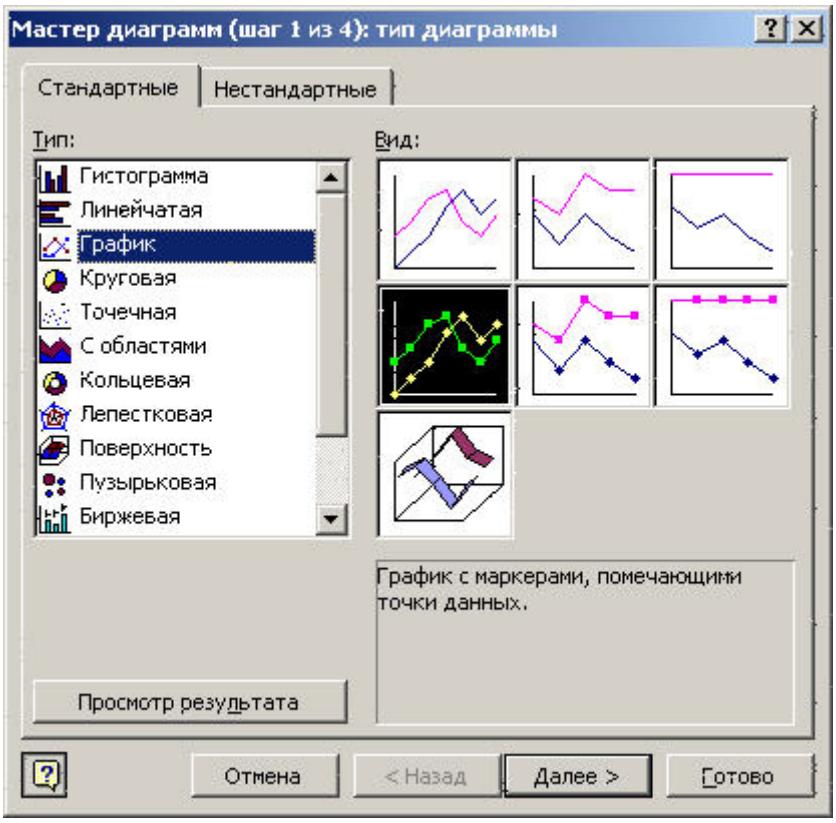


*Рис. 65. Окно диалога **Мастер диаграмм** на первом шаге создания графика*

Работа *Мастера диаграмм* происходит по шагам. Таких шагов четыре. На первом шаге выбирается тип диаграммы. Второй шаг – выделение части рабочего листа, которая рассматривается как область данных для построения диаграммы. На третьем шаге выбираются различные параметры оформления диаграммы. На последнем, четвертом, шаге определяется место размещения диаграммы.

Выделенный внедренный график можно редактировать (добавлять данные и выделять, форматировать, перемещать и изменять размеры большинства входящих в него элементов).

Внедренный график используется не только как средство визуализации табличных данных, но и как инструмент для моделирования. Зна-

чения величин, измененные на графике, автоматически обновляются в соответствующих ячейках исходной таблицы, что может быть использовано для решения некоторых задач.

Подготовка презентаций

Презентация – это представление информации по определенной теме, адресованное заинтересованным в обсуждении этой темы людям. Практически любому специалисту в предметной области приходится время от времени выступать в роли человека, проводящего презентацию при обсуждении планов, выступлении с докладом, представлении новых идей, отчете о проделанной работе и в других подобных ситуациях. Такое представление почти всегда сопровождается иллюстрационным материалом из графиков, рисунков, таблиц, диаграмм, чтобы сделать сообщение более наглядным, убедительным и аргументированным. Качество и способ изготовления иллюстрационного материала зависят от возможностей выступающего и часто определяют эффективность сообщения. Среди инструментальных средств, используемых для подготовки презентаций, большой популярностью пользуется входящая в пакет Microsoft Office программа Microsoft PowerPoint.

Возможности Microsoft PowerPoint и основные понятия

Microsoft PowerPoint – это мощное, удобное в работе и не сложное для освоения инструментальное средство подготовки мультимедийных презентаций. Подготовленные в нем даже начинающим пользователем презентации имеют профессиональный внешний вид. Microsoft PowerPoint позволяет создать презентацию для последующей демонстрации на экране или с использованием любой проекционной аппаратуры. Кроме того, можно сохранить презентацию в формате Web-страниц для размещения в Интернете. Среди дополнительных возможностей – распечатка и раздача слушателям бумажной копии иллюстрационного материала презентации для предварительного ознакомления с ее основными идеями. Мультимедийные возможности Microsoft PowerPoint позволяют оживить презентацию использованием звуковых эффектов и музыкальных фрагментов, видеоклипов и анимационных изображений. При хорошем вкусе и развитом чувстве меры создателя презентации такие возможности способны максимально усилить эффективность презентации, сделать ее яркой и запоминающейся.

Основным понятием Microsoft PowerPoint является *слайд.* Слайд является основной структурной единицей презентации, это ее отдельная «страница» или «кадр». Так же, как документ текстового процессора Microsoft Word состоит из последовательности абзацев, документ

PowerPoint (презентация) –это последовательность логически связанных слайдов, иллюстрирующих основные положения презентации. Каждый слайд может включать в себя текстовые области и любые нетекстовые объекты (рисунки, полноцветные иллюстрации, графики, диаграммы, таблицы, анимированные изображения, видеоклипы, звук и другие). При работе с PowerPoint используется множество других понятий, но все они так или иначе связаны с понятием слайда. Поэтому в дальнейшем они будут рассматриваться при первом упоминании.

Рабочая область окна Microsoft PowerPoint

После запуска программы появляется главное окно PowerPoint с открытой областью задач *Создание презентации*, разделы которой дают возможность создать презентацию одним из нескольких способов или открыть для редактирования и дальнейшей работы уже существующую презентацию. Использование этой области задач будет описано позже, а пока рассмотрим основные элементы главного окна с уже открытой презентацией (рис. 66).

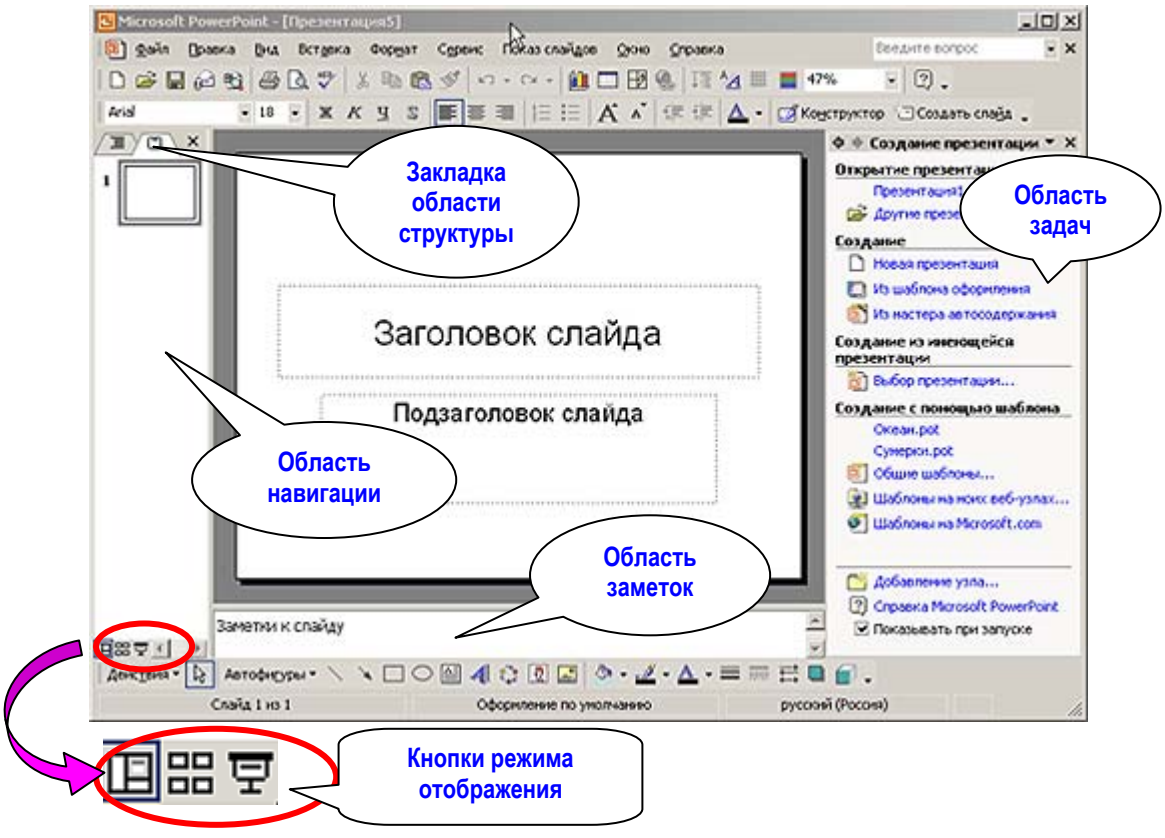


Рис. 66. Рабочая область окна Microsoft PowerPoint

Вид рабочей области окна PowerPoint зависит от режима отображения слайдов, который управляется командами пункта меню **Вид** или специальными кнопками, показанными на рис. 66. На рисунке рабочая область соответствует режиму *Обычный*. В этом режиме рабочая область разделена на три части: область навигации, область слайда и область заметок.

Область навигации предназначена для быстрого перемещения к нужному слайду. Маленькие эскизы слайдов, представленные здесь, позволяют легко в них ориентироваться, даже при отсутствии текста на них. При щелчке мышкой на соответствующей закладке вверху области (рис. 66) можно перейти к более привычной для пользователей прежних версий PowerPoint *области структуры* (прежний вариант области навигации). В этой области отображены номера и значки слайдов, и их текстовое содержание. В обоих вариантах области навигации можно изменять последовательность слайдов в структуре презентации простым перетаскиванием мышкой эскизов слайдов или их значков.

Область слайда. В этой области отображается текущий слайд со всеми его элементами. Здесь обычно выполняется основной объем работы при создании и редактировании отдельного слайда: изменение содержания текстовых областей, вставка рисунков, графиков и других нетекстовых объектов, изменение фонового рисунка и его цветовой схемы, настройка визуальных эффектов, сопровождающих демонстрацию презентации.

Область заметок. Эта область, расположенная непосредственно под областью слайда, она бывает полезна в процессе работы над презентацией. Здесь можно писать заметки докладчика, дополнительные сведения для аудитории, рабочие заметки.

Наряду с обычным режимом иногда бывает полезно использовать и другие режимы отображения. Смена режимов осуществляется с помощью пункта меню **Вид** или с помощью специальных кнопок, расположенных ниже области структуры (рис. 66). В порядке расположения эти кнопки соответствуют следующим режимам:

- обычный режим; видны и доступны все три области;
- режим сортировщика слайдов; полезен для организации презентации из подготовленных слайдов, позволяет просмотреть одновременно все слайды презентации в виде эскизов, изменить их порядок, просмотреть эффекты анимации сразу на нескольких выбранных слайдах;
- режим показа слайдов; позволяет провести полноэкранный показ текущего слайда со всеми визуальными и звуковыми эффектами.

Этапы создания презентации

В Microsoft PowerPoint возможно создание презентации несколькими способами. Выбор способа зависит от квалификации пользователя и его личных предпочтений. В этом пособии излагается способ, рассчитанный на начинающих пользователей. По мере приобретения опыта пользователи осваивают новые возможности, обогащая ими свой арсенал.

Процесс создания презентации состоит из нескольких этапов:

- создание слайдов; на этом этапе возможно создание пустых слайдов или слайдов на основе выбранного шаблона презентации;
- редактирование слайдов (редактирование текстовых областей, вставка нетекстовых объектов);
- создание презентации из подготовленных слайдов; на этом этапе выполняют настройку визуальных эффектов смены слайдов, настройку визуальных эффектов появления отдельных элементов слайда и сопровождающих их звуковых эффектов, различные другие настройки;
- репетиция презентации; выполняется полноэкранный показ презентации, во время которого можно определить время, необходимое для демонстрации каждого отдельного слайда, внести в электронную записную книжку замечания для доработки слайдов, вписать заметки докладчика по каждому слайду.

Далее рассматриваются этапы подготовки на примере создания простой презентации начинающим пользователем. Допустим, нужно создать презентацию для обучения основам работы в Microsoft PowerPoint. Начнем по порядку.

Создание слайдов

После запуска PowerPoint открывается его окно с областью задач «Создание презентации», изображенной на рис. 67. Эта область содержит разделы, позволяющие выбрать несколько способов создания презентации или открыть ранее созданную презентацию.

Начинающему пользователю в разделе *Создание* лучше выбрать возможность *Из мастера автосодержания*. Мастер автосодержания – это небольшая встроенная программа, которая создает автоматически первоначальный набор слайдов, запрашивая у пользователя необходимую для создания информацию.

При наличии некоторого опыта пользователь может создавать отдельные слайды на основе выбранного шаблона оформления. Для этого нужно выбрать возможность *Из шаблона оформления*. Более опытный пользователь, имеющий свои идеи оформления, может выбрать возможность Новая презентация, и создавать слайды, что называется, «с чистого листа».

Мастер автосодержания проводит пользователя через 5 этапов. Начальный и завершающий этап не требуют комментариев, остальные три иллюстрируются рис. 68−70. Каждый этап заканчивается щелчком по кнопке *Далее*, в конце работы с мастером нужно щелкнуть по кнопке *Готово*.

Создавая презентацию, предназначенную для обучения, уместно выбрать шаблон презентации *Учебный курс* (рис. 68, вид презентации), а в качестве стиля презентации – *презентацию на экране* (рис. 69).

При выборе стиля презентация в Интернете презентацию можно сохранить как Web-страницу и для показа ее можно будет использовать Web-обозреватель, например, Microsoft Internet Explorer.

На этапе *Параметры презентации* (рис. 70) можно ввести заголовок презентации на титульном слайде и, если необходимо, служебную информацию в поле *Нижний колонтитул*. Поле заголовка можно оставить свободным и в дальнейшем, на этапе редактирования, вставить заголовок в виде внедренного объекта *Фигурный текст*. Об этом способе будет рассказано ниже.

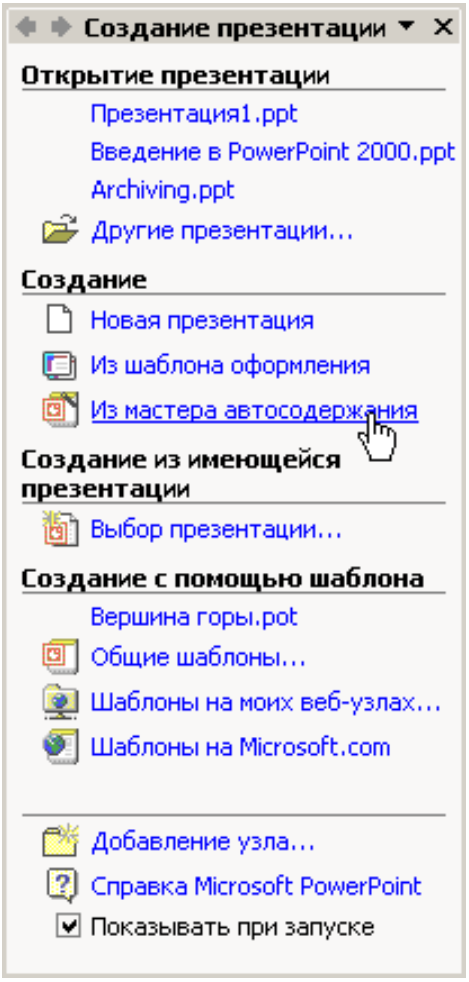


Рис. 67. Область задач ***Создание презентации***

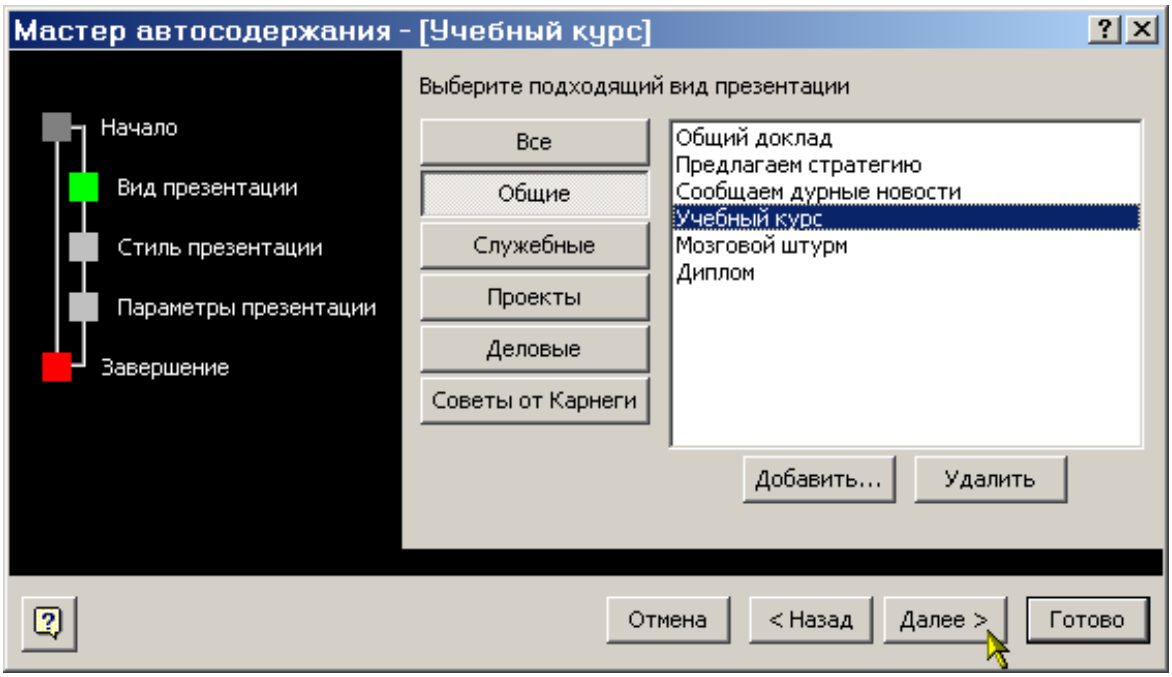


Рис. 68. Диалог мастера автосодержания – этап 2

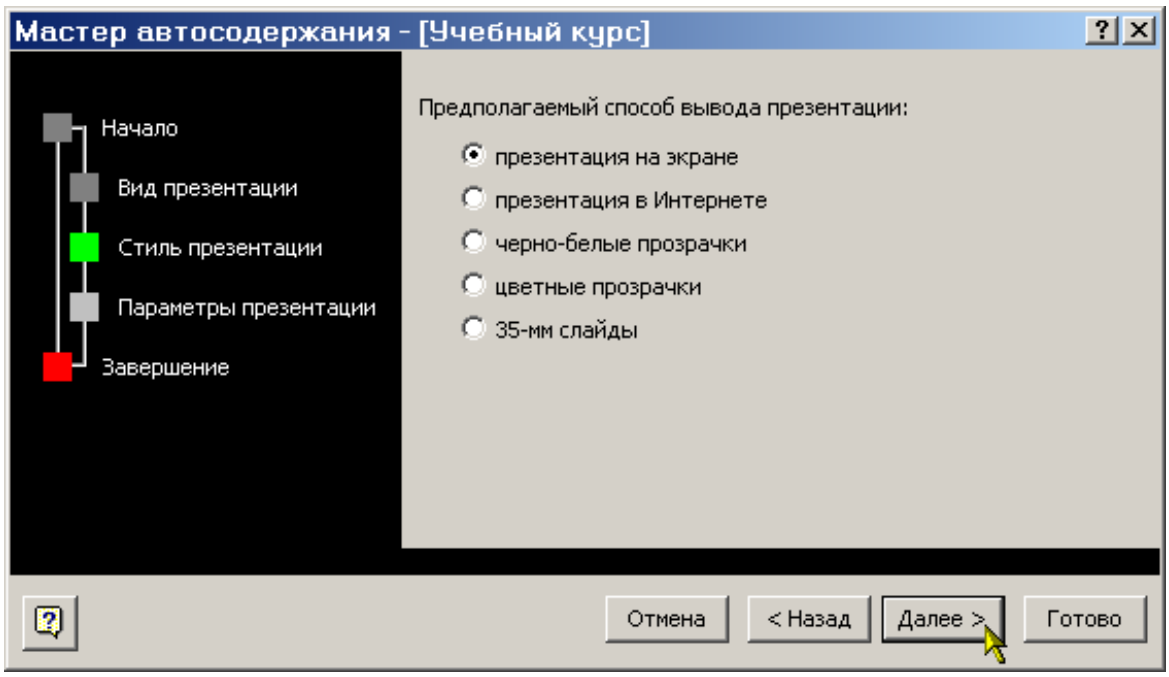


Рис. 69. Диалог мастера автосодержания – этап 3

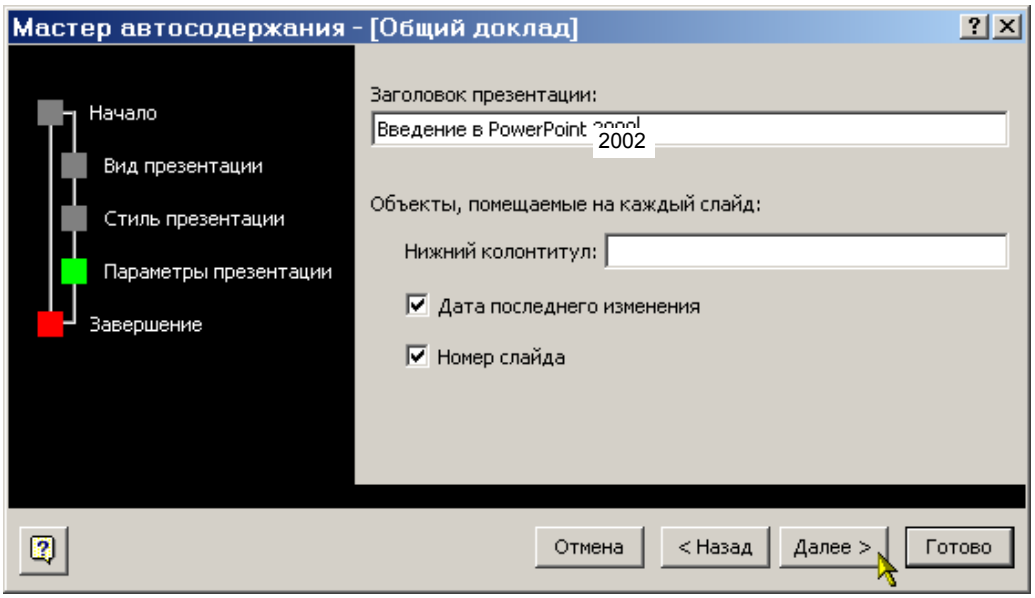


Рис. 70. Диалог мастера автосодержания – этап 4

После завершения работы с мастером автосодержания PowerPoint создаст заготовку будущей презентации в виде набора слайдов (рис. 71), которую можно редактировать для создания законченной презентации.

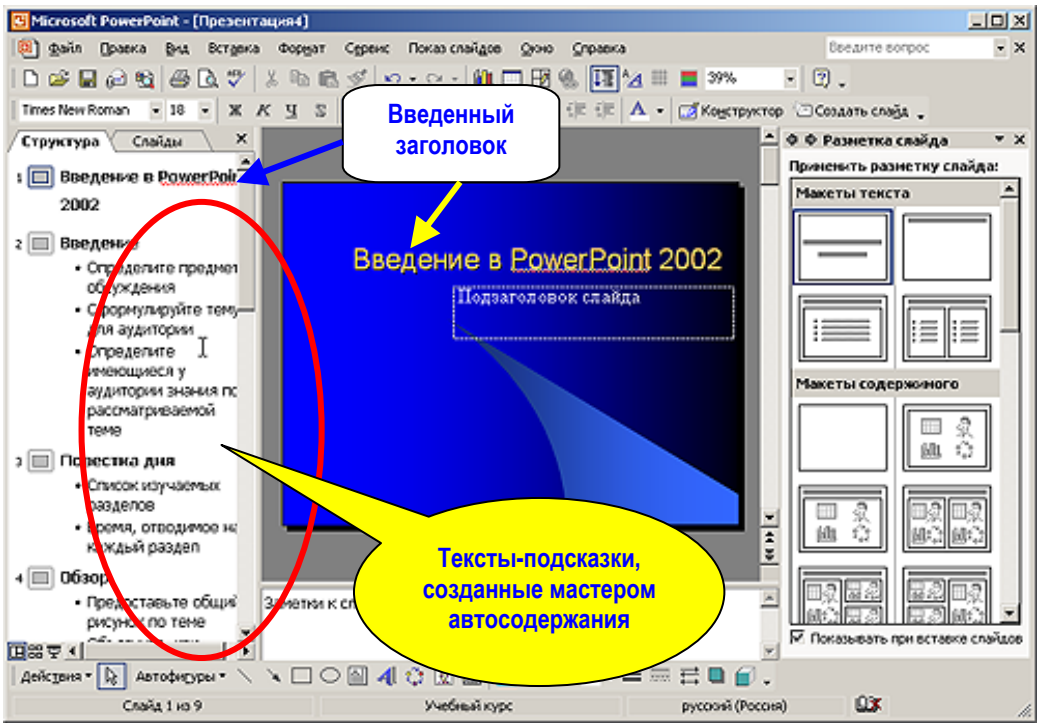


Рис. 71. Окно PowerPoint после завершения работы с мастером автосодержания

Мастер автосодержания позволяет лишь создать первоначальный набор слайдов с единым оформлением, соответствующим выбранному шаблону, и некоторыми настройками, которые PowerPoint создает «по

умолчанию». Текстовые области на заготовках слайдов являются, скорее, наводящими подсказками, помогающими пользователю, создающему презентацию, ввести на их месте нужный текст. Этим подсказкам можно следовать, но можно и вводить любой текст, соответствующий замыслу презентации.

Редактирование слайдов

Слайды, созданные мастером автосодержания, не содержат ничего, кроме фонового рисунка и текстовых областей с наводящими подсказками типа: «Определите предмет обсуждения» и «Сформулируйте тему для аудитории». Все остальное наполнение слайдов (изменение содержания текстовых областей, добавление новых, вставка нетекстовых объектов) осуществляется на этапе редактирования. Рассмотрим основные действия на этом этапе на примере создания нашей презентации.

Редактирование текстовых областей

Сначала займемся текстовыми областями. Редактировать их можно в области структуры или непосредственно в области слайда. Второй вариант удобнее тем, что позволяет выполнять дополнительные действия, такие как изменение пространственного положения и размеров текстовых областей. Заголовок на первом слайде уже есть. Текстовая область подзаголовка, выделенная пунктиром, пока пустая. Если щелкнуть мышкой на текстовой области, она будет выделена рамкой с маркерами. После выделения в текстовой области начинает мигать вертикальный курсор и текст можно вводить с помощью клавиатуры.

Приемы и средства работы с текстом полностью аналогичны приемам работы в текстовом процессоре Microsoft Word. Для работы со шрифтами и установки метода выравнивания абзацев используйте пункт меню **Формат/Шрифт** или знакомые по работе с Microsoft Word кнопки панелей инструментов.

Выделяющую рамку можно использовать для перетаскивания области в новое положение на слайде, а маркеры позволяют изменять размеры области.

Приемы перетаскивания и изменения размеров с помощью захвата мышкой – стандартные для Windows. Под захватом мышкой понимаются такие действия: сначала указатель мышки устанавливают на выделенный объект, на его рамку или выделяющие его маркеры (при этом указатель мышки часто меняет форму), нажимают левую кнопку мышки и, не отпуская ее, передвигают мышку в нужном направлении. Эти приемы применяют к любым объектам, нанесенным на слайд.

На рис. 72 изображен титульный слайд с текстовой областью подзаголовка в процессе ее перетаскивания.

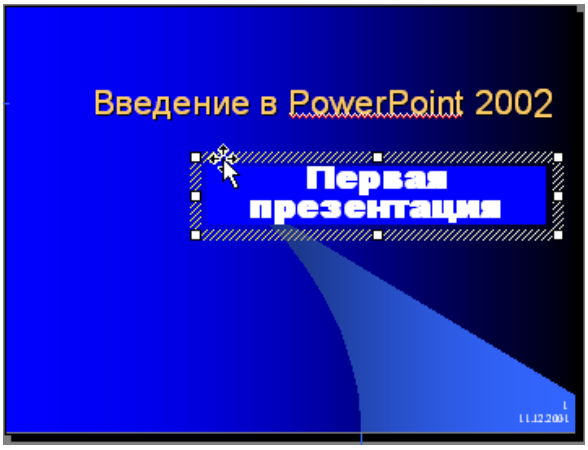


Рис. 72. Перетаскивание текстовой области на слайде

Точно так же редактируются текстовые области на остальных слайдах.

Вставка нетекстовых объектов

После редактирования текстовых областей можно вставлять нетекстовые объекты. Чтобы украсить титульный слайд, вставим вместо строки заголовка объект *Фигурный текст* (объект *WordArt*) и добавим рисунок.

Перед вставкой фигурного текста нужно выделить текст заголовка и удалить его. Для вставки фигурного текста нужно выполнить команду меню **Вставка/Рисунок/Объект WordArt…**. В результате выполнения команды появится окно диалога *Коллекция WordArt*, в котором следует выбрать нужный стиль надписи (рис. 73) и щелкнуть по кнопке *OK*.

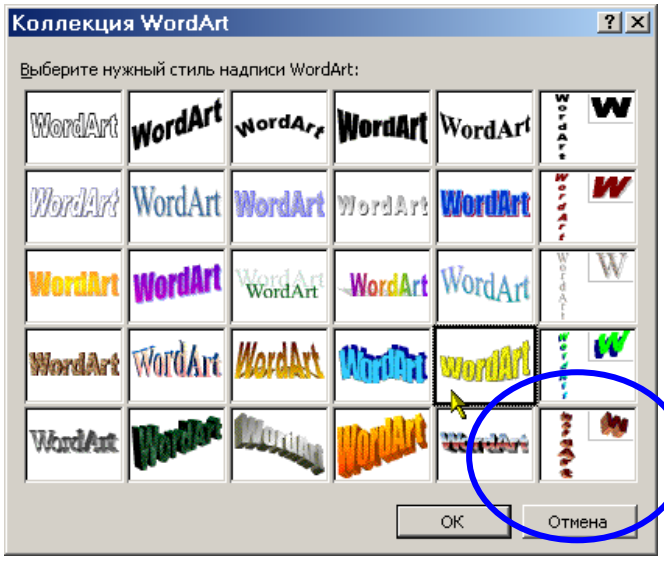


Рис. 73. Окно диалога для выбора стиля фигурного текста

На следующем этапе открывается окно для ввода текста, отображаемого с помощью объекта WordArt (рис. 74).

Результат вставки на слайд объекта WordArt изображен на рис. 75. После вставки объекта его перетаскивают мышкой на нужное место и изменяют его размеры до требуемых.

Обратите внимание, что на рис. 75 под вставленным фигурным текстом видна строка **Заголовок слайда**, хотя текст заголовка был предварительно удален. Эта строка является принадлежностью не самого слайда, а так называемого образца слайда, о котором речь пойдет ниже. При демонстрации презентации эта строка не будет видна.

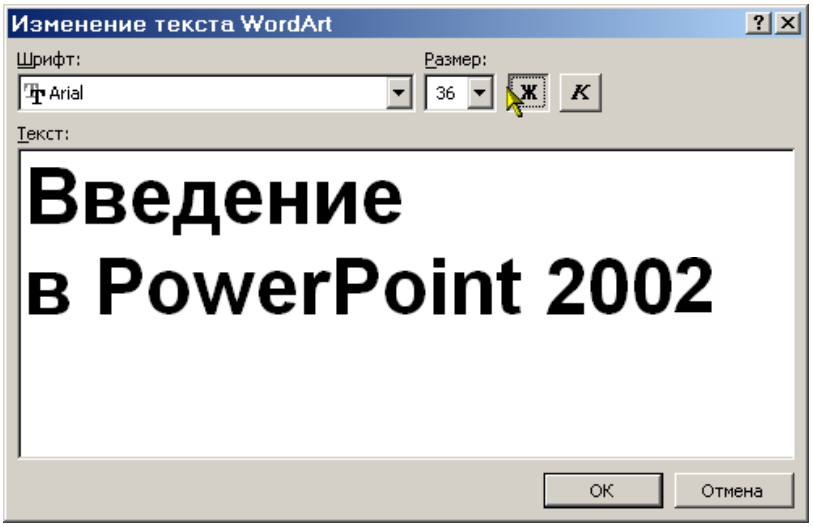
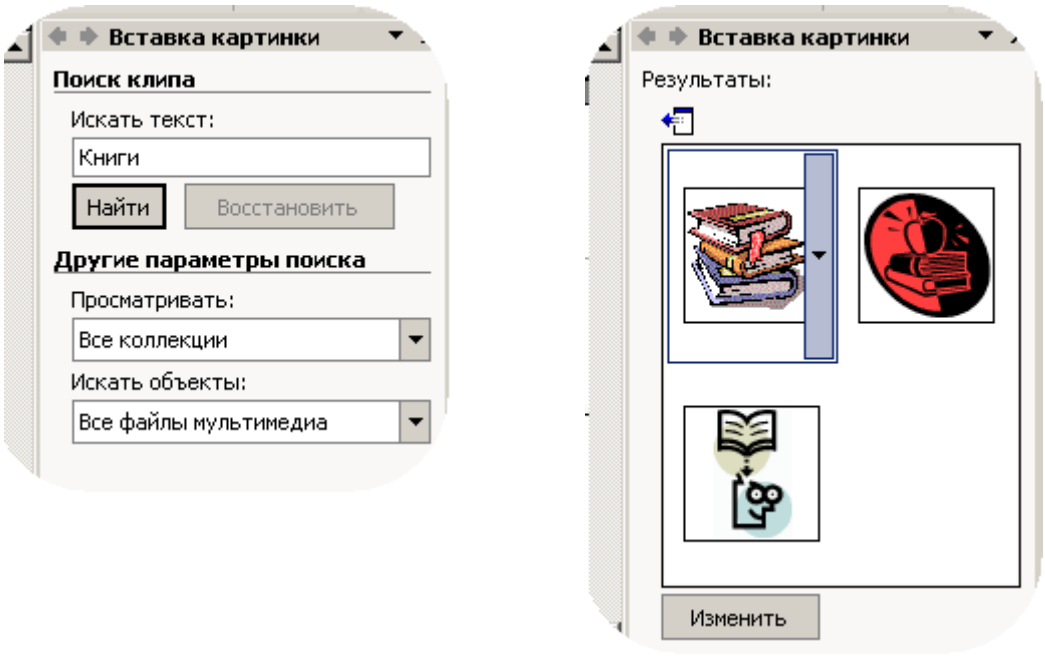


Рис. 74. Окно диалога для ввода текста надписи

Рис. 75. Слайд со вставленным и перемещенным в область заголовка объектом WordArt

Для реализации задуманного осталось вставить рисунок. Допустим, для презентации учебного характера решено вставить рисунок, изображающий стопку книг. Такой рисунок есть в одной из категорий коллекции рисунков, предоставляемых пакетом Microsoft Office. Для вставки рисунка из коллекции нужно выполнить команду меню **Вставка/Рисунок/Картинки…**, после выполнения которой появится область задач *Вставка картинки*. Технология вставки картинок из коллекций готовых изображений, одинаковая для всех приложений MS Office, уже была описана в разделе *Встраивание иллюстраций* главы «Текстовые процессоры». К описанному в этом разделе можно добавить, что в MS Office XP появилась удобная возможность поиска картинки по ключевому слову прямо из области задач. Не просматривая предварительно все коллекции картинок, в разделе *Поиск клипа* области задач *Вставка картинки* можно ввести в поле *Искать текст* ключевое слово, определяющее содержание рисунка, например, *Книги*, и щелкнуть по кнопке *Найти* (рис. 76, *а*). Все рисунки относящиеся к указанной категории, сразу появятся в области задач (рис. 76, *б*), откуда их можно копировать и вставлять в слайд описанным ранее способом.

После выполнения команды меню **Правка/Вставить** рисунок сразу появится на слайде, его можно перетащить в нужное место и изменить размеры (рис. 77).



а) *б)*

Рис. 76. Поиск рисунка по ключевому слову

Если рисунки из коллекции не подходят для оформления слайдов, но известно расположение файлов любого графического формата, содержащих нужные рисунки, то выполняют команду меню **Вставка/Рисунок/Из файла…**. В появившемся диалоге *Добавить рисунок* необходимо добраться до места расположения такого файла и, выделив файл рисунка, щелкнуть по кнопке *Вставить*.

Точно так же вставляются рисунки и любые другие нетекстовые объекты на все остальные слайды, где их присутствие необходимо.

Рис. 77. Вставленный на слайд рисунок в процессе изменения размера

Редактирование образца слайдов

Часто возникает ситуация, когда сделанные при редактировании изменения должны появиться одновременно на всех слайдах презентации. Например, необходимо на каждый слайд поместить логотип фирмы, проводящей презентацию, или поместить специальные кнопки, которые можно использовать для навигации по презентации. В этом случае изменения выполняются не на каждом слайде, а на образце слайдов.

Образец слайдов – совокупность элементов оформления (шрифт текста, его размер и цвет, цвет и рисунок фона и др.). Каждый шаблон оформления связан со своим образцом слайдов. Сделанные на образце изменения сразу же отражаются на всех слайдах презентации.

Для редактирования образца нужно выполнить команду меню **Вид/Образец/Образец слайдов**. В результате откроется образец слайдов (рис. 78), на котором выполняются нужные изменения. После завершения редактирования образца для возвращения к обычному режиму работы следует выполнить команду меню **Вид/Обычный**.

Для примера рассмотрим вставку управляющих кнопок на образец слайдов. По умолчанию в PowerPoint переход на следующий слайд пре-

зентации при ее показе происходит по щелчку левой кнопкой мышки. Управляющие кнопки позволяют более гибко осуществлять навигацию по слайдам презентации. Для их вставки нужно выполнить команду меню **Показ слайдов/Управляющие кнопки** и в выпадающей панели щелкнуть по требующейся кнопке (выпадающая подсказка и рисунок на кнопке поясняют ее назначение). После щелчка указатель мышки сменяется крестообразным курсором, с помощью которого указывают на слайде положение и размеры кнопки. Сразу после этого автоматически открывается окно диалога *Настройка действия* для настройки действия, происходящего при щелчке мышкой на кнопке. По умолчанию PowerPoint предлагает действие, связанное с типом кнопки. Например, для кнопки с изображением стрелки вправо предлагается переход на следующий слайд. Можно выбрать и любое другое действие из выпадающего списка.

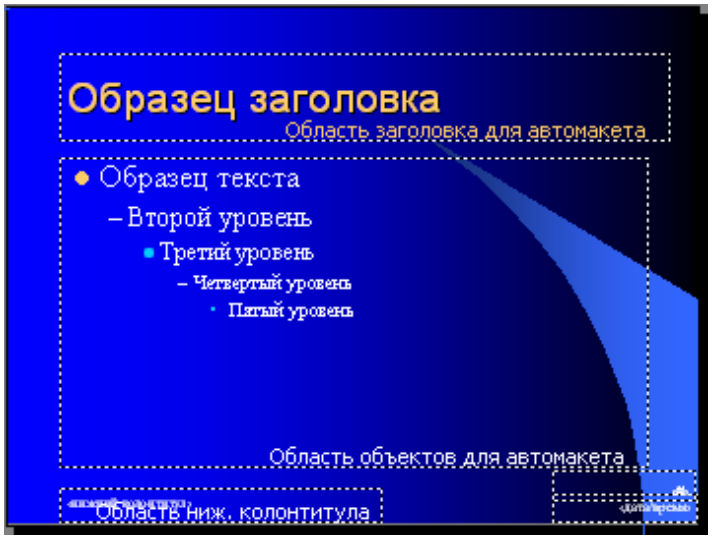


Рис. 78. Образец слайда

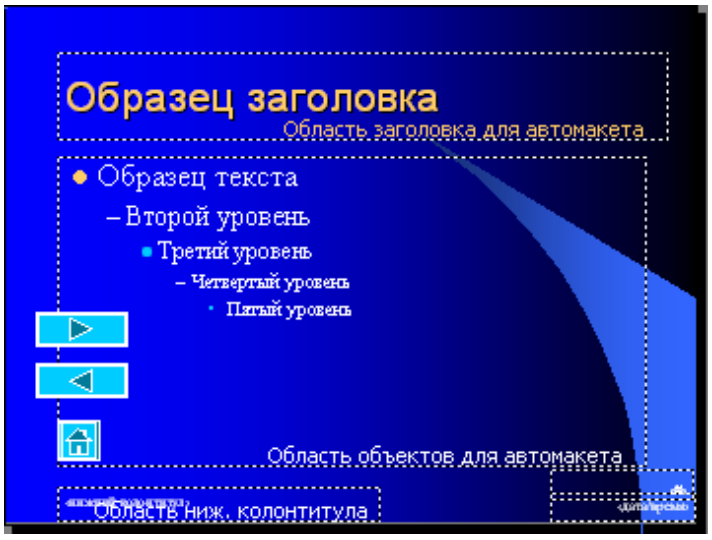


Рис. 79. Образец слайда с вставленными управляющими кнопками.

Для создаваемой презентации вставим кнопки перехода на следующий, предыдущий и первый слайды. После вставки всех трех кнопок (рис. 79) и возврата в обычный режим, кнопки для навигации появятся на всех слайдах презентации, кроме титульного.

Титульный слайд имеет отдельный образец. Доступ к нему открывается командой меню **Вид/Образец/Образец заголовков**.

Выполнение настроек

Превращение отдельных слайдов в презентацию начинается на этапе настроек. Основные из них – настройка смены слайдов и настройка визуальных и сопровождающих их звуковых эффектов появления элементов слайда. Все эти настройки могут сделать презентацию эффектной и зрелищной. При этом важно следовать правилу: эффекты должны способствовать главной цели презентации, а не отвлекать от нее. Обилие всевозможных эффектов так велико, что главным советчиком должно быть чувство меры.

Настройка смены слайдов

Для настройки смены слайдов нужно выполнить команду меню **Показ слайдов/Смена слайдов…**. В появившееся области задач *Смена слайдов* (рис. 80) в поле *Применить к выделенным слайдам* из прокручиваемого списка выбирают подходящий эффект (в нашем случае *Прямоугольник наружу*), в поле *Скорость* выбирают темп эффекта (медленно, средне или быстро). Если установлен флажок *Автопросмотр* внизу области задач, выбранный эффект и темп сразу иллюстрируется прямо на слайде.

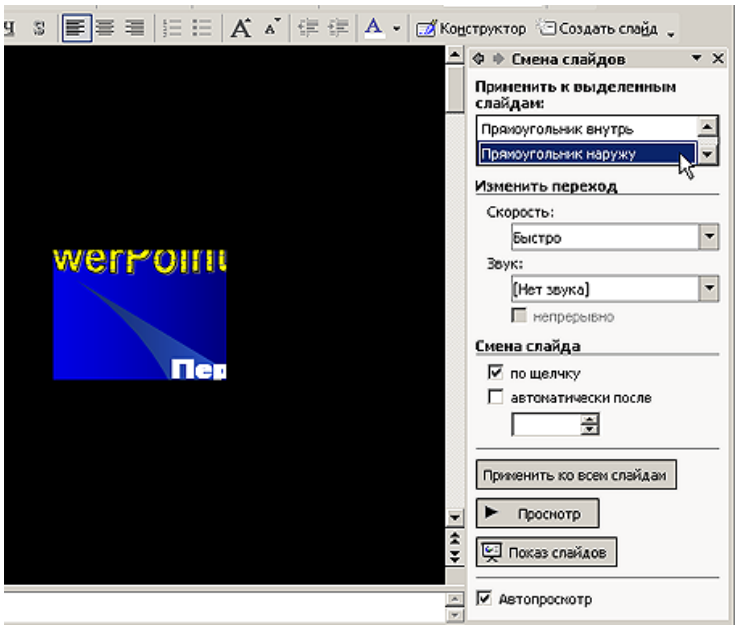


Рис. 80. Настройка смены слайдов

Кнопка *Просмотр* предназначена для просмотра выбранной настройки в области слайда, а кнопка *Показ слайдов* позволяет сделать то же самое в полноэкранном режиме.

Настройка анимации

Настройка анимации элементов слайда наиболее кропотливый и творческий процесс. Обилие эффектов, необходимость выбора анимируемых элементов, предварительного просмотра произведенных настроек – все это занимает немалое время, но и способствует (при развитом чувстве меры) максимальному достижению цели презентации. Рассмотрим этот процесс на примере настройки элементов титульного слайда создаваемой презентации.

Выбрав настраиваемый слайд, необходимо выполнить команду меню **Показ слайдов/Настройка анимации…**. Появляющаяся после выполнения команды область задач *Настройка анимации* (рис. 81) содержит несколько интуитивно понятных элементов для выбора и настройки эффектов анимации и подсказку в центре области, поясняющую, с чего начать настройку.

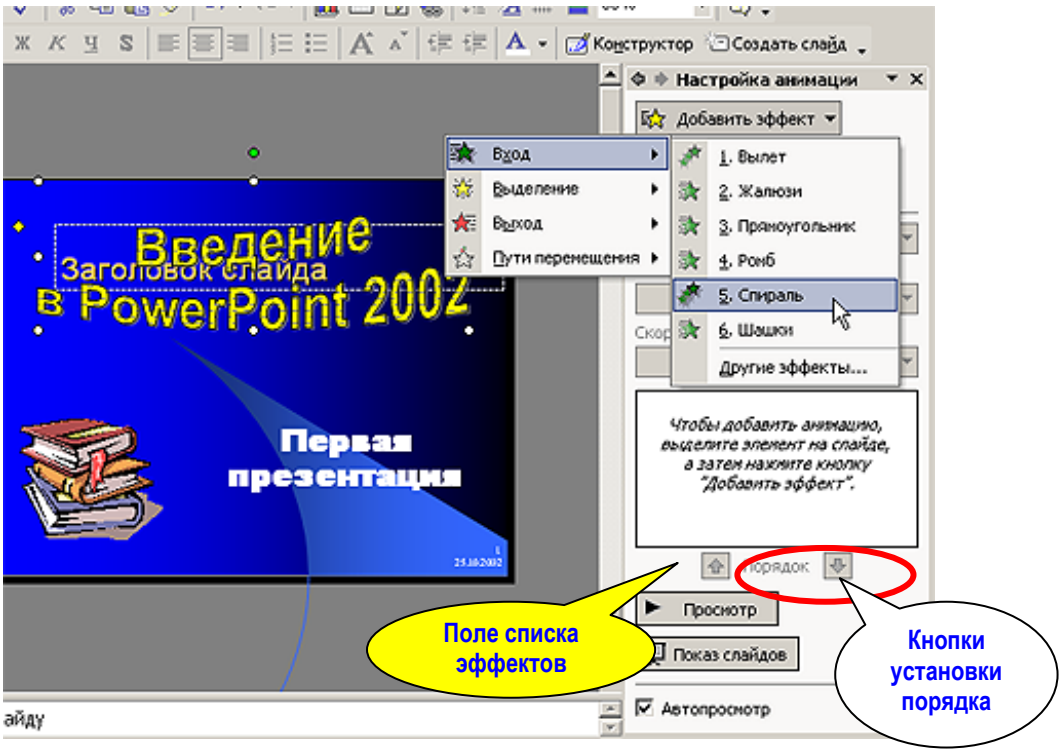


Рис. 81. Добавление эффекта анимации для выделенного элемента слайда

Как следует из подсказки, для добавления анимации к элементу слайда вначале этот элемент нужно выделить в области слайда, а затем щелкнуть по кнопке *Добавить эффект*.

Допустим, принято решение (после предварительного знакомства с набором всевозможных эффектов), что после появления слайда по спиралеобразной траектории вылетит объект WordArt, представляющий заголовок титульного слайда, после него появится подзаголовок **Первая презентация**, вылетающий буква за буквой справа, и следом, с эффектом растворения, появится рисунок.

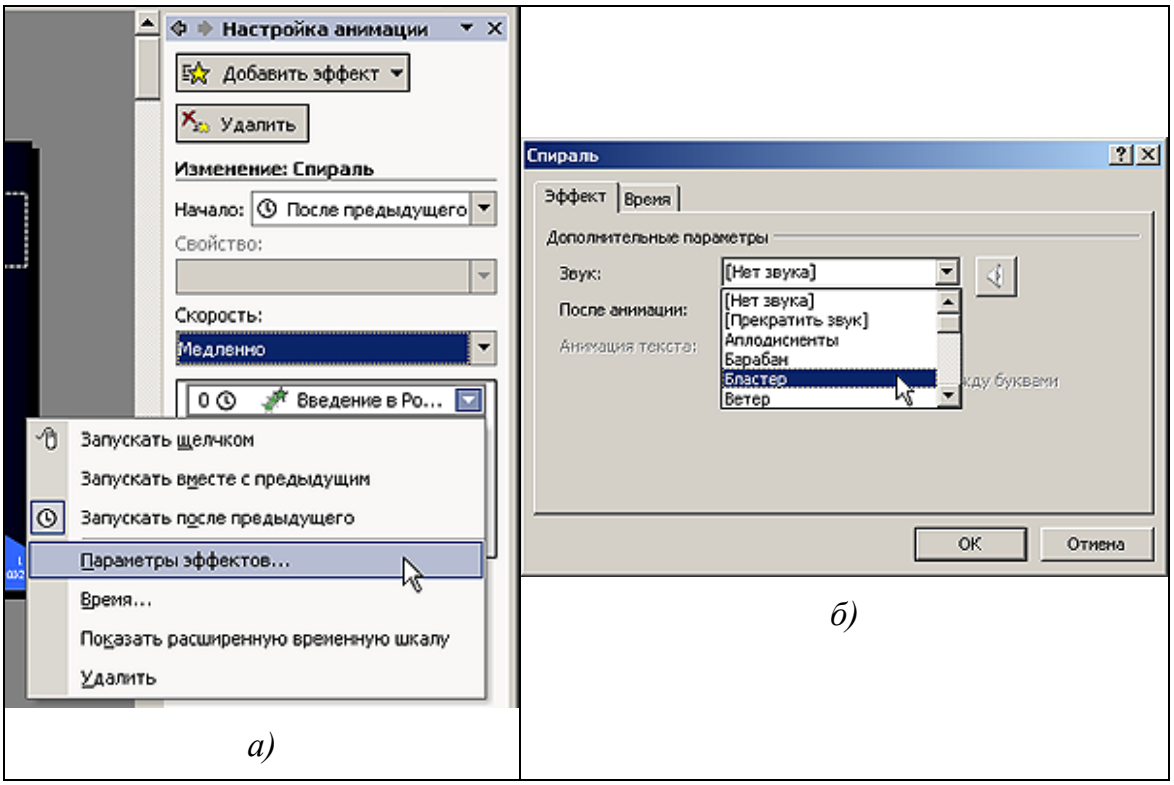
Выделив объект WordArt и щелкнув по кнопке *Добавить эффект*, нужно выбрать задуманный эффект. На месте первоначальной подсказки в области задач появится поле списка эффектов, в котором пока один выбранный нами эффект. Сразу после выбора эффекта он будет продемонстрирован в области слайда, если в области задач установлен флажок *Автопросмотр*. В поле *Начало* области задач из выпадающего списка выбирается событие, по которому происходит анимация (*По щелчку*, *С предыдущим* или *После предыдущего* эффекта). Указанная совокупность приемов настройки применяется для каждого выделяемого элемента слайда из тех, которые вы собираетесь анимировать.

Каждый эффект добавляется в поле списка эффектов области задач. Если в этом поле щелкнут мышкой на названии эффекта, то справа от названия появляется кнопка выпадающего списка (рис. 82, *а*). Список содержит возможности дополнительной настройки или удаления выбранного эффекта. Например, выбор команды **Параметры эффектов…** приводит к появлению окна диалога (рис. 82, *б*), используя которое можно выбрать сопровождающий анимацию звуковой эффект, а на вкладке *Время* установить точные временные характеристики настраиваемого эффекта (задержка, темп, возможность повторения). При анимации текста здесь же можно установить характер его появления (*все вместе*, *по словам* или *по буквам*).

К одному и тому же элементу слайда можно добавлять несколько различных эффектов анимации. После добавления всех эффектов анимации можно установить порядок их появления по времени с помощью кнопок со стрелками, которые расположены в области задач под списком эффектов (рис. 83).

Результат всех настроек можно немедленно просмотреть после щелчка по кнопке *Просмотр*. При необходимости настройки можно изменить, добиваясь задуманного эффекта.

Описанную процедуру настройки повторяют с каждым слайдом, объекты которого решено анимировать.



а) *б)*

Рис. 82. Настройка дополнительных параметров эффектов

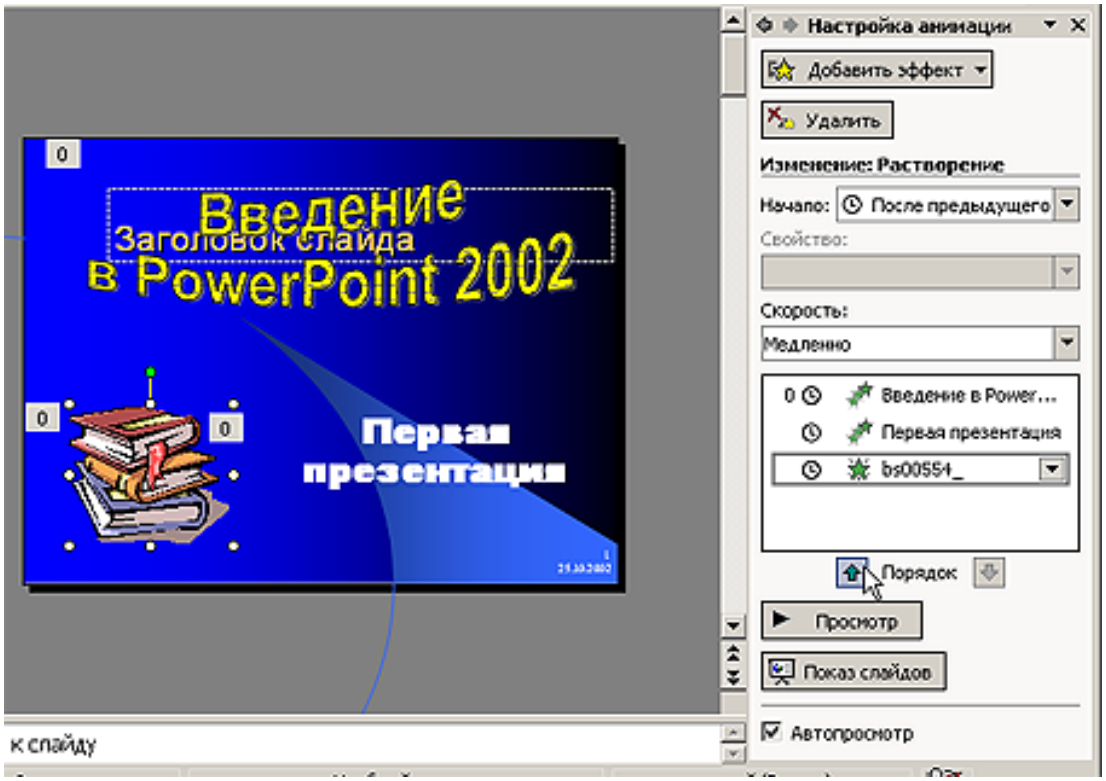


Рис. 83. Изменение порядка эффектов анимации

Завершение работы с презентацией

Закончив все настройки, можно провести репетицию презентации, выполнив команду меню **Показ слайдов/Начать показ** или щелкнув по кнопке режима *Показ слайдов* ниже панели структуры (рис. 66). Репетиция может выявить необходимость доработки, повторного редактирования отдельных слайдов, проведения дополнительных настроек и т. д. Завершив работу, необходимо сохранить презентацию. Для этого выполняют команду меню **Файл/Сохранить как…**. Работа с диалогом сохранения документа такая же, как и в других приложениях, поэтому здесь не описывается.

Математические пакеты

Компьютер является мощным средством автоматизации инженерно-технической и научной работы. Для решения сложных вычислительных задач используют специально созданные прикладные проблемно-ориентированные программы. В то же время существует множество не слишком сложных задач, для решения которых можно использовать универсальные средства. К таким задачам относятся:

- подготовка научно-технической документации, содержащей текст и формулы в привычной для специалиста форме;
- вычисления результатов математических операций с константами, переменными и размерными физическими величинами;
- векторные и матричные операции;
- решение уравнений и систем уравнений;
- статистические расчеты и анализ данных;
- построение графиков;
- аналитические преобразования и аналитическое решение уравнений и систем;
- аналитическое и численное дифференцирование и интегрирование;
- решение дифференциальных уравнений.

К универсальным средствам решения таких задач относятся математические пакеты, такие как Mathematica, MatLab, MathCad и другие. Предметом нашего рассмотрения будет один из самых распространенных среди технических специалистов пакет MathCad.

О системе MathCad

MathCad – это интегрированная система, которая предоставляет в распоряжение пользователя разнообразный набор средств как для проведения математических расчетов и преобразований, так и для создания документов, оформляющих результаты их выполнения. В наборе реали-

зованы численные математические методы, позволяющие решать алгебраические и дифференциальные уравнения, системы таких уравнений; методы численного дифференцирования и интегрирования; дискретные интегральные преобразования; представлены разнообразные матричные, векторные, статистические, трансцендентные, кусочно-непрерывные и другие функции. Кроме того, предоставлены некоторые возможности символьной математики: символьная алгебра, решение уравнений, действия с матрицами, интегральные преобразования, символьные вычисления и упрощения. Система имеет богатые графические возможности, позволяет строить графики в декартовых и полярных координатах, трехмерные гистограммы, векторные поля и карты линий уровня, возможно создание анимационных клипов и импорт графики из других приложений. Система предоставляет также возможность динамического обмена данными с другими программами.

Основные сведения по возможностям системы

Некоторые средства создания и редактирования формул и текста в системе

Главное окно MathCad представлено на рис. 84. В рабочей области окна набираются математические выражения в привычном для пользователя виде, т. е. с применением общепринятых математических обозначений и символов.

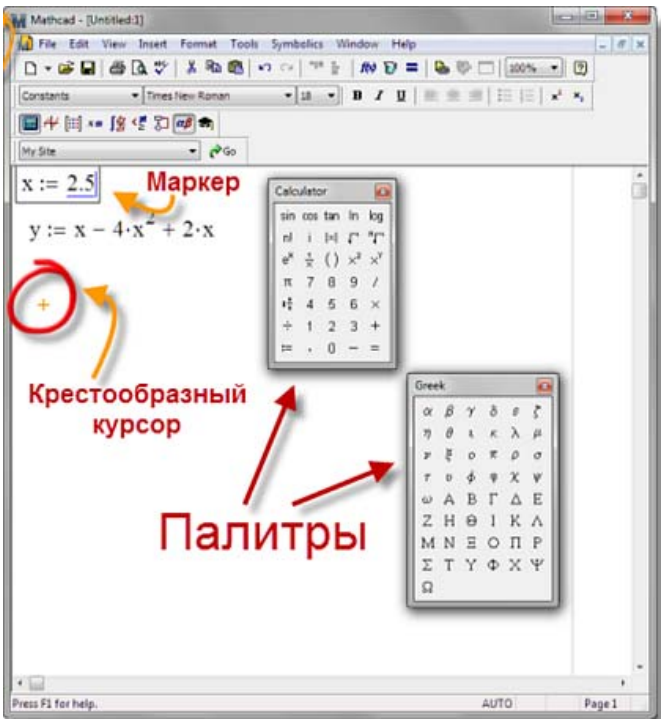


Рис. 84. Внешний вид окна MathCad

В MathCad при создании и редактировании используются следующие средства:

- **крестообразный курсор (визир)**;
- **маркер ввода.** Он вводится щелчком мыши в нужном месте. Используется в математических выражениях, текстах, графиках для указания места удаления или вставки символа. Вставка одного или нескольких символов происходит во время их набора, место вставки – левее курсора;
- **выделяющая рамка.** Используется для выделения частей математических выражений или графиков при их удалении или вставке. Наименьшее выделение в выражении – это имя переменной или число.

Простые вычисления

MathCad позволяет производить простые вычисления подобно калькулятору. Чтобы это сделать, достаточно набрать вычисляемое выражение со знаком **=** в конце его и нажать клавишу ввода. После этого MathCad вычислит и выведет результат на экран. Более сложные вычисления выполняются с помощью переменных, определяемых специальным образом.

Комплексные числа и действия с ними

Комплексные числа возникают в результате некоторых вычислений или вводятся пользователем для своих расчетов. MathCad воспринимает комплексные числа в алгебраической форме: $\mathbf{a + b}i$, где a и b – обычные числа, а i – мнимая единица.

Предостережение

При вводе комплексных чисел следует иметь в виду, что нельзя использовать ***i*** или ***j*** сами по себе для ввода комплексной единицы. Нужно всегда набирать ***1i*** или ***1j,*** в противном случае MathCad воспримет i или ***j,*** как обычную переменную. Когда курсор покидает выражение, содержащее ***1i*** или ***1j,*** MathCad скрывает лишний множитель ***1***.

В MathCad существует несколько специальных функций и операторов, реализующих общепринятые в математике действия с комплексными числами:

Re(z)	вещественная часть z;
Im(z)	Мнимая часть z;
arg(z)	Угол в комплексной плоскости между вещественной осью и z. Возвращает результат между -π и π радиан;
|z|	Модуль *z*. Чтобы записать модуль от выражения, нужно заключить его в выделяющую рамку и нажать клавишу с вертикальной чертой.

Повторяющиеся вычисления

Для многократного вычисления выражения с рядом изменяющихся значений в MathCad используется специальный тип переменных – *дискретные аргументы.* Переменная этого типа принимает значения в выбранном диапазоне с заданным шагом изменения значений. Если в выражении есть переменная такого типа, то MathCad вычисляет выражение столько раз, сколько значений содержит дискретный аргумент.

Дальнейшее расширение возможностей расчетов связано с определением **функций**. Функция – есть правило, согласно которому проводятся некоторые вычисления с её аргументами и вырабатывается её числовое значение. Чтобы ***определить функцию***, нужно сначала набрать ее имя с аргументом в круглых скобках, затем символ двоеточия. В процессе набора двоеточие отобразится знаком присваивания. В появившемся поле ввода следует набрать выражение, с помощью которого вычисляется значение функции. Для получения набора значений функции следует набрать ее имя с именем дискретного аргумента в круглых скобках и завершить ввод знаком равенства. Чтобы получить результат для одного конкретного аргумента в скобках, после имени функции указывают значение этого аргумента.

Все переменные, используемые в выражении, должны быть определены заранее. В противном случае переменные, не имеющие значения, будут отмечены в рабочей области негативным изображением.

Ввод текста

Текст в рабочих документах может присутствовать в форме текстовых областей.

Текстовые области могут иметь произвольную ширину, и могут быть расположены в любом месте документа.

Для создания текстовой области следует выполнить пункт **Create Region (Создать текстовую область)** из меню **Text.** В результате визир, который указывает на свободное место, заменится прямоугольником для размещения текста (текстовая рамка). По мере набора текста рамка расширяется, охватывая набранное. В случае необходимости текст можно отредактировать. Операции редактирования текста можно выполнить через пункт меню **Edit** (**Правка**), предварительно выделив нужную область мышкой. Чтобы покинуть текстовую область, щёлкните вне её. Не нажимайте **Enter**, это просто приведёт к переходу на новую строку внутри текста.

Графики

MathCad позволяет строить разнообразные графики с помощью пунктов меню **Graphics** (**Графика**). Чтобы *построить* график, нужно:

- определить дискретную переменную в заданном диапазоне с требуемым шагом;
- определить функцию этой переменной;
- щелкнуть кнопкой мыши на свободном месте, где предполагается разместить график;
- выбрать и выполнить пункт меню **Create X-Y Plot** (**Декартов график)** из меню **Graphics** (**Графика**); MathCad создаст пустой график. В этом графике в центральные поля ввода нужно ввести обозначения переменной и функции.

После заполнения полей нужно щелкнуть кнопкой мыши вне области графика. MathCad построит график автоматически.

Для *нанесения масштабной сетки, осей координат, делений на осях* нужно воспользоваться окном диалога **Formatting Currently Selected X-Y Plot** (**Форматирование текущего X-Y графика**). Это окно можно вызвать на экран, дважды щелкнув на графике кнопкой мыши. Далее следует выбрать нужные закладки и с помощью открывшихся окон выполнить операции форматирования.

Векторы и матрицы

Векторы и матрицы – это очень распространенные математические средства проведения научных и инженерных расчетов во многих областях. В частности, многие расчеты электрических цепей упрощаются с помощью матричного представления. MathCad представляет широкие возможности для использования векторов и матриц.

Создание и редактирование вектора или матрицы

MathCad использует следующие математические определения:

- *скаляр* – это одиночное число (вещественное или комплексное);
- *вектор* – это столбец чисел;
- *матрица* – это прямоугольная таблица чисел.

Кроме перечисленных терминов, в MathCad для обозначения вектора или матрицы может использоваться общий термин – *массив.* Имеется три способа создания массива:

- заполнение пустых полей ввода для небольших массивов;
- использование дискретного аргумента в тех случаях, когда есть формула вычисления значений элементов массива через их индексы;
- считывание значений из файла данных.

Чтобы *создать* вектор или матрицу, нужно набрать имя матрицы или вектора, затем символ присваивания и воспользоваться палитрой матричных и векторных операций на инструментальной панели (или командой меню **Math/ Matrices**). В появившемся окне диалога указать размерность матрицы, создать ее и заполнить пустые поля ввода нужными значениями.

Можно *изменить матрицу* с помощью следующих действий:

- указать место вставки строк или/и столбцов (выделить элемент матрицы). Чтобы вставить строку выше первой строки или столбец левее первого столбца, следует сначала заключить матрицу целиком в выделяющую рамку;
- воспользоваться палитрой матричных и векторных операций для вставки или удаления нужного количества строк и столбцов.

Вставка и удаление строк и столбцов происходит ниже и правее выделенного элемента.

Чтобы *удалить* или *скопировать* вектор или матрицу целиком, нужно заключить их в выделяющую рамку.

Нужно иметь в виду, что в MathCad существуют *ограничения на размеры* используемых массивов. ***Нельзя*** использовать пункт **Matrices (Матрицы)** из пункта меню **Math (Математика)**, если в массиве ***более 100*** элементов. Такие массивы ***можно создать*** либо чтением данных из файла, либо соединяя массивы **A** и **B** допустимых размеров в один массив с помощью функций **augment**(A, B) или **stak**(A, B). Функция **augment**(A, B) объединяет массивы A и B с одинаковым числом строк в один массив, располагая их рядом, бок о бок. Функция **stak**(A, B) делает это же для массивов A и B с одинаковым числом столбцов, располагая A над B.

Использование вектора или матрицы

При использовании созданных вектора или матрицы следует различать два случая:

- использование созданного массива, как единого целого, что соответствует понятию векторной или матричной переменной;
- использование отдельного элемента массива, что соответствует понятию элемента матрицы или отдельного компонента вектора.

После создания массива его именем можно пользоваться в расчетах, как соответствующей переменной.

В случаях, когда в расчетах нужно обращаться к отдельным элементам массива, используют понятие *нижнего индекса*. При этом следует иметь в виду, что по умолчанию нумерация элементов массивов в MathCad начинается с нуля.

MathCad дает возможность выделять из матрицы отдельный столбец. Для этого используется *верхний индекс,* с помощью которого указывается номер выделяемого столбца. Для выделения нужного столбца указывается имя матрицы и с помощью палитры матричных и векторных операций вводится верхний индекс. После нажатия знака равенства на экране появится столбец значений.

Если необходимо выделить значения строки матрицы, ее предварительно следует транспонировать, а затем использовать верхний индекс для выделения нужного столбца.

Векторные и матричные операторы

MathCad предлагает большой набор векторных и матричных операторов. Нужно сразу обратить внимание на то, что во всех векторных операторах всегда имеется в виду вектор-столбец. Некоторые из операторов можно вызвать, используя палитру матричных и векторных операций.

Ряд операций можно выполнить с помощью палитры арифметических операций и символов.

Решение системы линейных алгебраических уравнений

MathCad позволяет решать системы линейных уравнений $A \cdot x = b$, где A – матрица коэффициентов при неизвестном векторе x, а b – вектор-столбец, составленный из свободных членов уравнений, следующими способами:

- применением специальной функции lsolve(A, b); решение получается также в виде вектора-столбца;
- использованием обращения матрицы A; решение получается в виде вектора-столбца x и вычисляется по формуле $x = A^{-1} \cdot b$.

Использование функции lsolve(A, b) показано на рис. 85.

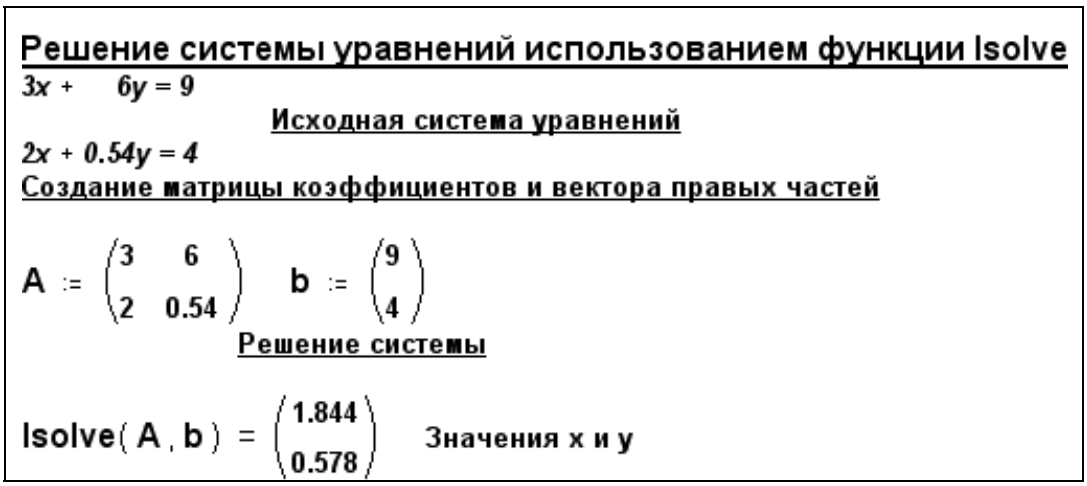


Рис. 85. Использование функции lsolve

Далее приведено решение этой же системы уравнений с помощью обращения матрицы коэффициентов (рис. 86).

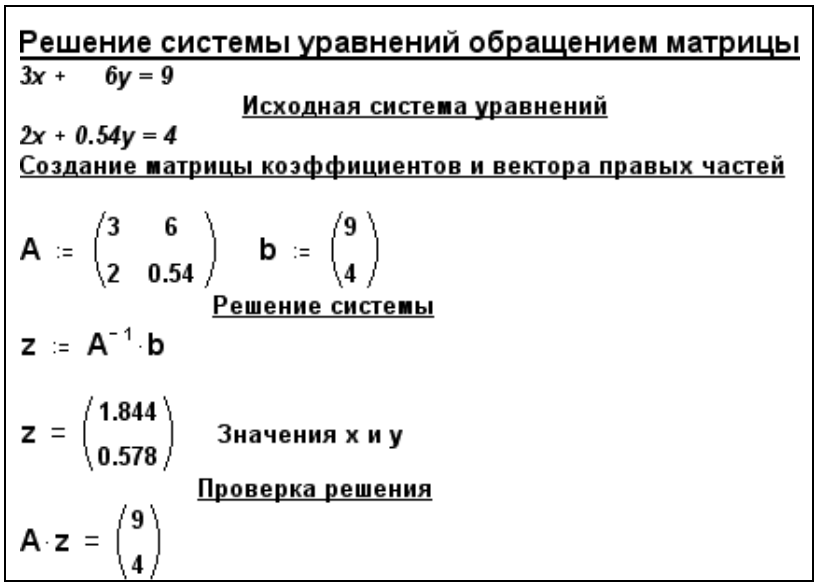


Рис. 86. Использование обращения матрицы

Векторные и матричные функции

MathCad содержит функции для обычных в линейной алгебре действий с массивами. Их аргументами являются вектора или матрицы, причем нужно использовать, как уже отмечалось, вектор-столбец. Ниже приводятся некоторые из наиболее употребительных функций, существующих в MathCad. Используются обозначения: A – матрица, v – вектор.

Функция	**Результат**
rows(A)	Число строк в массиве A;
cols(A)	Число столбцов в матрице A;
max(A)	Наибольший элемент в матрице A;
min(A)	Наименьший элемент в матрице A;
Re(A)	Матрица, состоящая из действительных частей комплексной матрицы (вектора) A;
Im(A)	Матрица, состоящая из мнимых частей комплексной матрицы (вектора) A;
length(v)	Число элементов в векторе v;
last(v)	Индекс последнего элемента в векторе v;
tr(A)	Сумма диагональных элементов матрицы A.

Кроме этого, существуют и другие векторные и матричные функции, сведения о которых можно получить, обратившись к помощи MathCad. Вызвать любую функцию можно, либо набрав ее имя, либо выбрав имя из списка при выполнении команды **Math/Choose Function… (Выбрать функцию)**.

Символьные вычисления

MathCad позволяет выполнять не только численные расчеты, но и символьные (аналитические). Это такие расчеты, где результатами преобразования выражений в ходе вычислений являются также выражения, представленные в символьном виде. При этом желаемая форма преобразованных выражений может быть задана. Первоначальное выражение можно разложить на множители, проинтегрировать, выполнить дифференцирование, разложить в ряд, реализовать некоторые интегральные преобразования и выполнить иные математические операции, например, матричные. Все символьные вычисления можно осуществить через пункт меню **Simbolic.**

При выполнении символьных вычислений нужно иметь ввиду, что:

- многие вычисления могут быть выполнены только численно;
- в ряде случаев получаются такие длинные ответы, что удобнее использовать соответствующие численные расчеты с последующим графическим представлением этих ответов.

Чтобы продифференцировать функцию один или несколько раз, нужно вызвать соответствующий оператор производной с помощью палитры *Производные и интегралы*, заполнить пустые поля ввода, заключить выражение в выделяющую рамку и нажать клавиши **Shift**/**F9**.

Получить первую производную от заданной функции можно и не используя оператор производной. Для этого нужно набрать функцию, выделить переменную, по которой выполняется дифференцирование и выполнить пункт меню **Simbolic/Differentiat on Variable**.

Для вычисления неопределенного или определенного интеграла следует вызвать соответствующий оператор интегрирования с помощью палитры *Производные и интегралы*, заполнить пустые поля ввода для подинтегрального выражения и переменной интегрирования, заключить выражение в выделяющую рамку и нажать клавиши **Shift**/**F9**.

Вычислить неопределенный интеграл от заданной функции можно и не используя оператор интегрирования. Для этого нужно набрать функцию, выделить переменную, по которой выполняется интегрирование и выполнить пункт меню **Simbolic/Integrate on Variable**.

Для того чтобы найти решение системы уравнений в символьном виде, нужно:

- ввести слово Given;
- ввести уравнения и неравенства ниже слова Given;
- ввести функцию Find с аргументами-неизвестными;
- убедиться в том, что команда **Math/Live Symbolics** (**Математика/ Использовать символику)** в меню отмечена галочкой, в противном случае выберите эту команду;

- удерживая клавишу **Ctrl**, нажать клавишу с точкой. MathCad выводит стрелку вправо;
- щёлкнуть вне области действия функции **Find**. MathCad вырабатывает результат в виде вектора. Результаты размещаются в том же порядке, в котором были перечислены неизвестные – аргументы функции **Find**.

Пример использования приводится ниже:

Given

$x + 2\cdot\pi\cdot y = a$

$4\cdot x + y = b$

$$\text{find}(x,y) \rightarrow \begin{bmatrix} \frac{1}{(-1 + 8\cdot\pi)}\cdot(2\cdot\pi\cdot b - a) \\ \frac{-(-4\cdot a + b)}{(-1 + 8\cdot\pi)} \end{bmatrix}$$

Базы данных

Развитие методов организации и обработки данных

Вначале следует сказать об одном часто встречающемся заблуждении, которое является следствием путаницы понятий. Принято отождествлять понятия «данные» и «информация». Однако это вовсе не одно и то же, что легко увидеть из приводимых ниже определений.

Данные – описание сведений о реальном мире в формализованном виде, например, в виде числа или строки символов.

Информация – смысл, приписываемый данным посредством принятых людьми соглашений.

Далее при изложении постараемся следовать принятым определениям.

С самого начала развития вычислительной техники образовалось два основных направления ее использования: вычислительное и информационное.

Первое направление – применение вычислительной техники для выполнения численных расчетов.

Второе направление – это использование средств вычислительной техники в автоматизированных информационных системах. В самом широком смысле информационная система представляет собой программный комплекс, функции которого состоят в поддержке надежного

хранения данных в памяти компьютера, выполнении специфических для данного приложения преобразований данных (или вычислений), предоставлении пользователям удобного и легко осваиваемого интерфейса. Обычно объемы данных, с которыми приходится иметь дело таким системам, достаточно велики, а сами данные имеют достаточно сложную структуру. Классическими примерами информационных систем являются банковские системы, системы резервирования авиационных или железнодорожных билетов, мест в гостиницах и т. д.

По мере увеличения объемов данных все насущнее требуется автоматизация процессов их обработки и хранения. На первых порах эту задачу решали путем автоматизации отдельных решаемых подзадач, что привело к ряду противоречий и трудностей:

- возникла проблема контроля избыточности данных, поскольку данные в разных задачах часто дублировались;
- возникла взаимосвязь между данными и прикладными программами. Любое изменение в организации данных приводило к необходимости изменения программы со всеми вытекающими последствиями;
- выборочная автоматизация информационных процессов нарушала сложившиеся в системе (предприятии) взаимосвязи.

Проиллюстрировать, например, вторую проблему можно следующим образом. На раннем этапе развития информационных систем использовалась общепринятая в то время для вычислительных систем технология работы с большими массивами данных. Каждая прикладная программа, которой требовалось хранить данные во внешней памяти, сама определяла расположение каждой порции данных на магнитной ленте или барабане и выполняла обмены между оперативной и внешней памятью с помощью программно-аппаратных средств низкого уровня (машинных команд или вызовов соответствующих программ операционной системы). Такой режим работы не позволяет или очень затрудняет поддержание на одном внешнем носителе нескольких архивов долговременно хранимых данных (и связанной с ними информации). Кроме того, каждой прикладной программе приходилось решать проблемы именования частей данных и структуризации данных во внешней памяти. То есть большой объем данных мог использоваться только одной этой программой.

Однако для информационных систем ситуация коренным образом отличается. Эти системы главным образом ориентированы на хранение, выбор и модификацию постоянно существующих данных. Структура данных зачастую очень сложна, и хотя структуры данных различны в разных информационных системах, между ними часто бывает много общего. На начальном этапе использования вычислительной техники для управления информацией проблемы структуризации данных реша-

лись индивидуально в каждой информационной системе. Производились необходимые надстройки над файловыми системами (библиотеки программ).

Проблемы использования неструктурированных данных легко пояснить следующим примером. Данные, содержащие сведения о студентах (номер личного дела, фамилию, имя, отчество и год рождения), записанные в текстовом файле являются неструктурированными.

Личное дало N 16493, Сергеев Петр Михайлович, дата рождения 1 января 1876 г; Л/д. N 16593. Петрова Анна Владимировна, дата рожд. 15 марта 1975 г; N личн. дела 16693, д.р. 14.04,78, Анохин Андрей Борисович.

Легко убедиться, что сложно организовать поиск необходимых данных, хранящихся в таком виде, а упорядочить подобную информацию практически не представляется реальным.

Цель любой информационной системы – обработка данных об объектах реального мира. При создании информационных систем требуется упорядочить информацию по различным признакам и быстро извлекать выборку с произвольным сочетанием признаков. Сделать это возможно, только если данные структурированы.

Структуризация – это представление данных в соответствии с принятыми соглашениями о способах их представления.

Чтобы автоматизировать поиск и систематизировать эти данные, необходимо выработать определенные соглашения о способах представления данных, т. е. дату рождения нужно записывать одинаково для каждого студента, она должна иметь одинаковую длину и определенное место среди остальной информации. Эти же замечания справедливы и для остальных данных (номер личного дела, фамилия, имя, отчество).

N личного дела	Фамилия	Имя	Отчество	Дата рождения
16493	Сергеев	Петр	Михайлович	01.01.76
16393	Петрова	Анна	Владимировна	15.03.75
16693	Анохин	Андрей	Борисович	14.04.76

Поскольку информационные системы требуют сложных структур данных, эти дополнительные индивидуальные средства управления данными являлись существенной частью информационных систем и практически повторялись от одной системы к другой. Стремление выделить и обобщить общую часть информационных систем, ответственную за управление сложно структурированными данными, явилось одной из побудительных причин создания СУБД – систем управления базами данных. Очень скоро стало понятно, что невозможно обойтись

общей библиотекой программ, реализующей над стандартной базовой файловой системой более сложные методы хранения данных.

Эти и другие трудности привели к пониманию того, что при переходе к новой ИТ, способной обеспечить работу с большими объемами данных, требуется как взаимная увязка отдельных задач, так и новый подход к организации данных. Были сформулированы стандартные требования к организации данных в новой ИТ. Основными из них являются:

- концентрация данных в одном месте и создание постоянно обновляемой модели предметной области. Предметная область (ПО) – это часть реального мира, моделируемая какими-либо средствами;
- максимально возможная независимость прикладных программ от данных.

Чтобы удовлетворить этим требованиям, необходимо создать единый блок данных для каждой ПО, который называется *базой данных* (БД). Кроме того, необходимо разработать для данной ПО единую управляющую программу для манипулирования данными на физическом уровне (*система управления базой данными – СУБД).*

Модели данных

Любая БД строится в соответствии с некоторым набором данных, в рамках которого представляется информация о реальном мире. Основные понятия и способы организации таких наборов данных, используемые при моделировании объектов реального мира и их взаимосвязей, можно назвать *моделью данных.*

В процессе развития технологии баз данных в качестве основных рассматривались три модели данных: сетевая, иерархическая и реляционная. В настоящее время наиболее распространенной является реляционная модель, которая поддерживается подавляющим большинством СУБД.

Реляционная модель данных

Предварительно рассмотрим несколько важных для реляционной модели данных понятий.

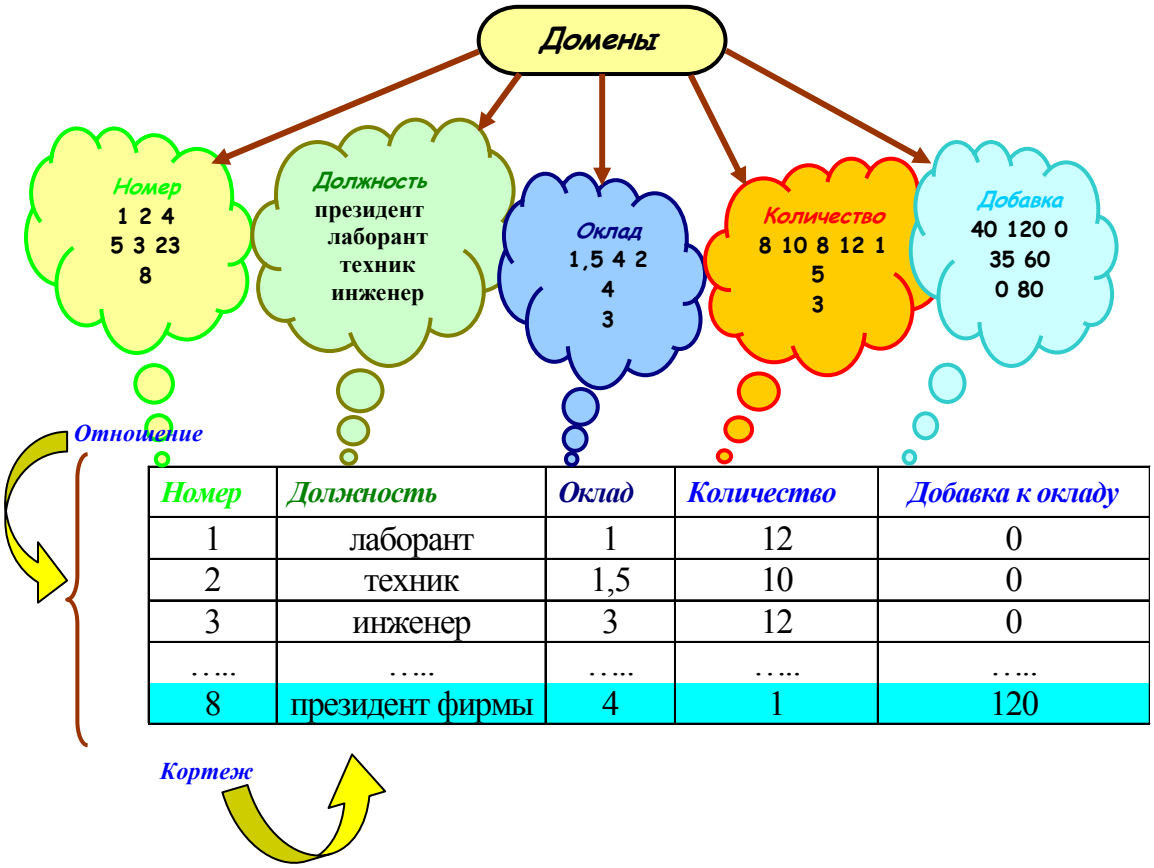
Понятие *тип данных* в реляционной модели данных полностью эквивалентно понятию типа данных в языках программирования. Это понятие позволяет разграничить данные по их смысловому назначению. Под *типом данных* понимается *множество допустимых значений* данных, относящихся к этому типу, *и набор операций*, разрешенных над данными этого типа. Обычно в современных реляционных БД используются данные символьных, числовых и специализированных типов данных (денежные типы, типы даты, времени и др.).

Понятие *домена* более специфично для баз данных и соответствует понятию множества в математике. Наиболее правильной интуитивной трактовкой понятия домена является понимание домена как допустимого множества значений данного типа.

Кортеж – это последовательность, составленная из элементов доменов. Кортеж содержит по одному элементу из каждого домена, причем порядок расположения этих элементов строго задан.

В основе реляционной модели данных лежит понятие отношения. В общем случае, *отношение* можно определить как множество, элементами которого являются кортежи. На рис. 87 приведено графическое пояснение этого определения.

В приведенном рисунке домены *Номер, Количество* и *Добавка к окладу* – множества целых чисел, домен *Должность* – множество строк и домен *Оклад* – множество вещественных чисел. Множество кортежей и является *отношением*, а вся полученная таблица является графическим представлением нового множества. Чаще всего эта таблица отождествляется с отношением, т. е. о ней говорят, что это и есть отношение.



Номер	Должность	Оклад	Количество	Добавка к окладу
1	лаборант	1	12	0
2	техник	1,5	10	0
3	инженер	3	12	0
…..	…..	…..	…..	…..
8	президент фирмы	4	1	120

Рис. 87. Графическая иллюстрация понятий реляционной модели данных

Имена доменов, на базе которых получают отношение, называются его *атрибутами*. В нашем примере атрибутами являются имена *Номер, Должность, Оклад, Добавка к окладу*. Соответственно, значения элементов домена – это *значения атрибутов*. В примере – это значения целых чисел и строки. В табличном представлении атрибутам соответствуют имена столбцов, а кортежам соответствуют строки таблицы. Содержимое одной ячейки таблицы – это значение соответствующего атрибута.

Таким образом, теперь можно дать следующее **определение реляционной модели данных**: это совокупность основных понятий и способов организации данных, используемых для моделирования ПО, которая основана на отношениях.

Концепция реляционной модели данных была предложена Е.Ф. Коддом в 1970 году в связи с решением задачи обеспечения независимости представления и описания данных от прикладных программ. Он показал, что набор отношений может быть использован для хранения данных об объектах реального мира и моделирования связей между ними.

Реляционная база данных – это совокупность взаимосвязанных отношений, содержащих всю информацию о ПО. Каждое отношение отображается таблицей и в компьютере хранится в виде файла записей. Взаимосвязь отношения, таблицы и файла может быть представлена следующими соответствиями:

Отношение	**Таблица**	**Файл**
Кортеж	*Строка*	*Запись*
Атрибут	*Столбец*	*Поле записи*

Реляционная БД – это не просто набор таблиц (отношений). Каждая таблица отображает отношение, полученное по определенным правилам из других отношений. Существует строгая система операций (реляционная алгебра), которая позволяет выводить одни отношения из других подобно тому, как выполняются арифметические операции. Применение такой системы дает возможность делить информацию на хранимую и нехранимую (вычисляемую) части. И при необходимости вычислять нужную информацию из хранимой части, что экономит память.

Проектирование базы данных

Любая база данных является информационной моделью предметной области. Предметная область представляется множеством *фрагментов*, например, предприятие – цехами, дирекцией, бухгалтерией и т. д. Каждый фрагмент предметной области характеризуется множеством *объектов* и *процессов*, использующих объекты, а также множеством *пользователей* с различными взглядами на предметную область. Для того

чтобы представить ПО в базе данных, нужно прежде всего выделить те понятия, которые представляют интерес только с точки зрения создания БД. Иными словами, нужно выполнить классификацию понятий ПО. Далее эти понятия будут представляться в *БД*. Проведение такой классификации называется *концептуальным проектированием данных*.

При концептуальном проектировании технические характеристики компьютера и особенности его программного обеспечения в расчет не берутся. Главное на этом этапе – правильно представить многочисленные связи объектов реального мира, т. е. промоделировать объекты и их связи. Описание ПО в терминах некоторой модели данных принято называть *концептуальной схемой.* Концептуальная схема может быть представлена в виде предложений естественного языка, а возможно представление в виде набора таблиц. Какие именно термины естественного языка будут использованы или что будет внесено в таблицы – определяется конкретной ПО. В любом случае на этапе создания концептуальной схемы объекты ПО, их связи, свойства объектов представляются именами.

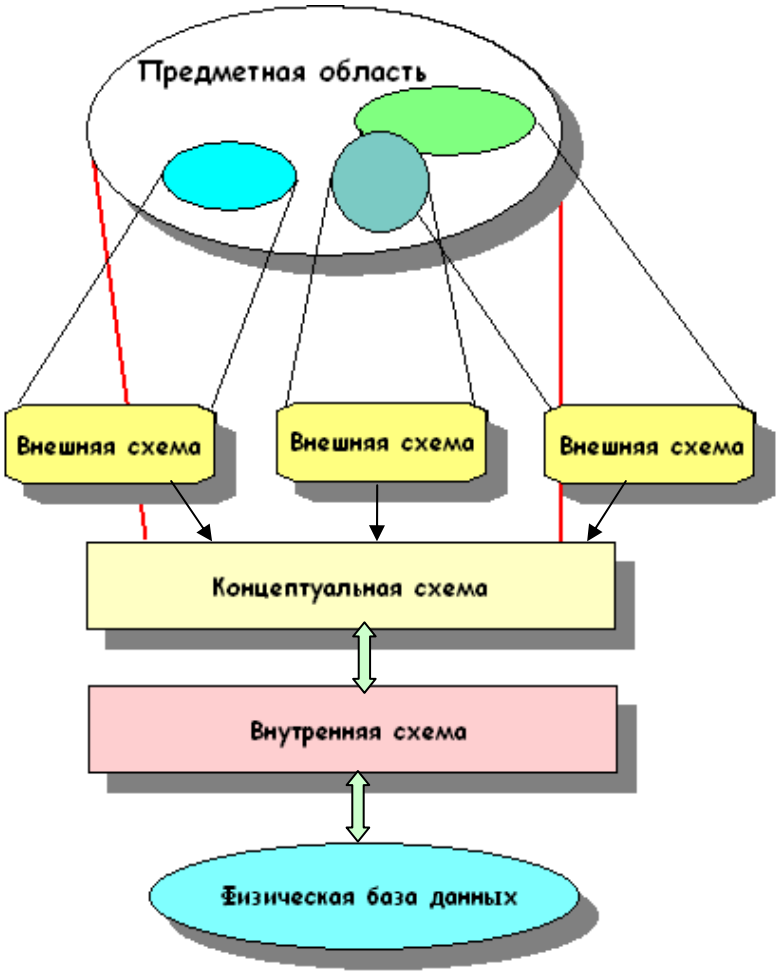


Рис. 88. Уровни предсавления данных

Обычно для отдельных пользователей базы данных интерес представляет только часть ПО. Это означает, что в общей концептуальной схеме для отдельного пользователя необходимо выделить некоторую подобласть понятий и, возможно, даже преобразовать ее для конкретного применения. Такая выделенная часть концептуальной схемы называется *внешней схемой*. Одной ПО соответствует одна концептуальная схема и большое количество внешних схем.

Описание концептуальной схемы в терминах данных, представляемых в памяти компьютера, называется *внутренней схемой*.

Соответствие между понятиями, введенными при моделировании ПО, иллюстрирует рис. 88

Пример разработки базы данных

Рассмотрим условную предметную область «Учебный процесс», включающую следующие основные понятия: «студент», «преподаватель», «изучаемая дисциплина», «оценки». Пусть требуется создать базу данных для хранения данных о студентах, преподавателях и изучаемых дисциплинах, используя которую можно, например, получить сведения о студентах, имеющих право получать стипендию.

На этапе логического проектирования необходимо выделить основные объекты предметной области, которые требуется моделировать, дать им имена и описать их атрибуты. Приведем эти объекты в виде списка имен, за которыми в скобках следует список атрибутов.

Можно выделить четыре объекта − *Студенты, Дисциплины, Оценки* и *Преподаватели:*

- *Студенты* (код студента, фамилия, имя, отчество, номер группы, дата рождения, стипендия, оценки).
- *Дисциплины* (код дисциплины, название дисциплины),
- *Оценки (*код студента, код дисциплины, оценка),
- *Преподаватели* (код преподавателя, код дисциплины, фамилия, имя, отчество, дата рождения, телефон, название дисциплины).

В реляционной базе данных в качестве объектов рассматриваются отношения, которые можно представить в виде таблиц. Таблицы между собой связываются посредством общих полей, т. е. одинаковых по форматам и, как правило, по названию, имеющихся в обеих таблицах.

Рассмотрим, какие общие поля надо ввести в таблицы для обеспечения связности данных. В таблицах Студенты и Оценки таким полем будет «Код студента», в таблицах Дисциплины и Оценки – «Код дисциплины», в таблицах Преподаватели и Дисциплины – «Код дисциплины». Выбор цифровых кодов вместо фамилий или названий дисциплин обу-

словлен меньшим размером данных в таких полях: например, число «2» по объему занимаемой памяти значительно меньше слова «математика». На рис. 89 представлена схема базы данных, где жирными буквами выделены ключевые поля.

Рис. 89. Схема базы данных

Ключом называют любую функцию от атрибутов отношения, с помощью которой можно однозначно определить конкретный кортеж. Такая функция может быть значением одного из атрибутов (простой ключ), задаваться алгебраическим выражением, включающим значения нескольких атрибутов (составной ключ). Это означает, что данные в строках каждого из столбцов составного ключа могут повторяться, но комбинация данных каждой строки этих столбцов является уникальной. Например, в таблице Студенты есть столбцы Фамилии и Год рождения. В каждом из столбцов есть некоторые повторяющиеся данные, т. е. одинаковые фамилии и одинаковые года рождения. Но если студенты, имеющие одинаковые фамилии, имеют разные года рождения, то эти столбцы можно использовать в качестве составного ключа. Как правило, ключ является уникальным, т. е. каждый кортеж определяется значением ключа однозначно, но иногда используют и неуникальные ключи (ключи с повторениями). Обозначение 1 ∞ на схеме соответствует типу связи между таблицами «один-ко-многим». При таком типе связи одной строке таблицы, например, Студенты с уникальным значением ключа «Код студента» может соответствовать множество строк таблицы Оценки с таким же значением поля «Код студента».

В приведенной схеме каждое отношение представляет собой таблицу, каждая строка которой состоит из полей, значения которых являются значениями атрибутов отношения. Например, таблица С*туденты* может иметь следующий вид:

Код студента	Фамилия	Имя	Отчество	Номер группы	Дата рождения	Стипендия
16493	Сергеев	Петр	Михайлович	9а51	01.01.87	750
16393	Петрова	Анна	Владимировна	9а52	15.03.86	750
16693	Анохин	Андрей	Борисович	9а52	14.04.87	800
–	–	–	–	–	–	–

Для работы с базой данных используются специальные программные средства манипулирования данными – *СУБД* (*системы управления базами данных)*. Широко известны СУБД, предназначенные для пользователей различного уровня – от локальных СУБД для работы с небольшими базами данных (например, СУБД MS Access) до мощных СУБД для профессионалов и больших организаций (СУБД Oracle, Interbase и др.). Рассмотрим основные понятия СУБД.

Основные понятия СУБД

СУБД – это комплекс программных средств, предназначенных для создания новой базы данных, наполнения ее содержимым, редактирования и отображения данных в соответствии с заданным критерием.

Основные режимы работы СУБД

С базами данных работают две категории людей: проектировщики и пользователи. В связи с этим СУБД имеет два режима работы: проектировочный и пользовательский. В проектировочном режиме создатель БД работает с ее структурой и имеет полный доступ к базе. Пользовательский режим используется для наполнения базы данными с помощью форм, обработки данных с помощью запросов и получения результатов в виде таблиц или отчетов. Доступ к структуре БД для рядовых пользователей закрыт.

Основные средства СУБД

Основными средствами СУБД являются:

- средства описания структуры БД;
- средства конструирования экранных форм для ввода данных;
- средства создания запросов для выборки данных при заданных условиях и выполнения операций по их обработке;
- средства создания отчетов;
- языковые средства (макросы, встроенный алгоритмический язык, язык запросов) для реализации нестандартных алгоритмов обработки данных.

Основные объекты СУБД

Основными объектами любой СУБД являются таблицы, запросы, формы, отчеты, макросы и модули.

Таблицы служат для хранения всех данных, имеющихся в БД, и ее структуры (полей, их типов и свойств).

Запросы используются для извлечения данных из таблиц и предоставления их пользователю в удобном виде. С помощью запросов

данные обрабатывают (упорядочивают, фильтруют, отбирают, изменяют, объединяют, выполняют простейшие вычисления в таблицах). Запросы обеспечивают сохранность данных в таблицах БД и разграничение доступа к различным данным для разных категорий пользователей. Например, для базы данных «Учебный процесс» можно создать запрос для получения списка отличников по результатам сессии. Результат использования запроса приведен на рис. 90.

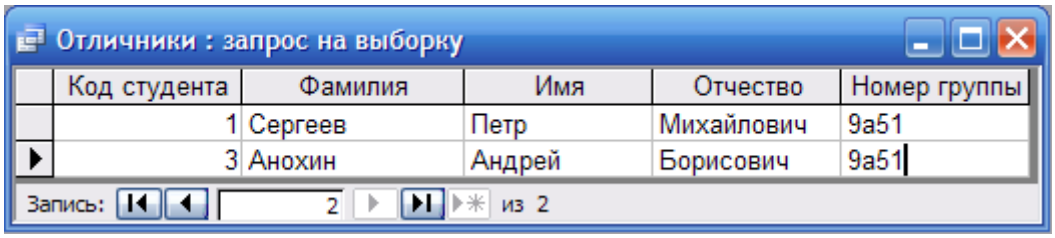


Рис. 90. Результат использования запроса

Формы – это средство для ввода данных. Они используются для заполнения тех полей таблицы, к которым есть доступ пользователям данной категории. В форме можно разместить специальные элементы управления для автоматизации ввода (раскрывающиеся списки, переключатели, флажки и т. п.). Формы особенно удобны для ввода данных с заполненных бланков.

Отчеты предназначены для вывода данных на принтер в удобном и наглядном виде. В отчетах данные таблиц и запросов преобразуются в документы.

Макросы и **модули** предназначены для автоматизации повторяющихся операций при работе с СУБД и создания новых функций путем программирования. Макросы состоят из последовательностей внутренних команд СУБД, модули создаются средствами внешнего языка программирования.

ГЛАВА 4. КОМПЬЮТЕРНЫЕ СЕТИ

Сетью *называют объединение нескольких компьютеров для совместного использования информации и ресурсов*. Для создания сети используется специальное аппаратное и программное обеспечение. Сети бывают *локальные и глобальные*.

Локальная сеть – это сеть, которая объединяет компьютеры, находящиеся в одной комнате, в одном или нескольких близко расположенных зданиях. При этом для соединения компьютеров используются выделенные линии связи, принадлежащие той же организации, что и компьютеры.

Глобальные сети объединяют компьютеры в пределах региона, страны и даже континентов. В качестве примера можно привести глобальную сеть Internet, покрывающую своей паутиной узлов весь мир. Для создания глобальных сетей приходится брать в аренду телефонные и спутниковые линии связи.

Для обеспечения работы любой сети необходимо соблюдение трех основных требований:

- обязательно наличие соединений, т. е. промежуточной аппаратуры (сетевой интерфейс) для связи компьютеров и передающей среды. Обычно подсоединение к сети осуществляется специальной съемной платой, называемой сетевой интерфейсной платой;
- должны быть установлены правила (протоколы), по которым компьютеры общаются друг с другом, что связано с возможностью установки на них разного программного обеспечения;
- должны быть определены услуги (сервис), т. е. перечень тех операций, которые один компьютер может делать для другого.

Эволюция компьютерных систем

Появление компьютерных сетей является логическим результатом эволюции компьютерной технологии. Можно выделить несколько этапов такой эволюции.

Первые компьютеры – большие, громоздкие и дорогие – предназначались для небольшого числа пользователей и использовались в режиме *пакетной обработки*. В этом режиме в вычислительном центре устанавливался мощный и надежный компьютер универсального назначения – мэйнфрейм. Пользователи сдавали в центр перфокарты с данными и командами программ, операторы центра вводили эти карты в компьютер. Обычно результаты работы программы становились известными пользователю на следующий день. Интересы пользователей на этом этапе развития практически не учитывались.

По мере удешевления процессоров начали развиваться *интерактивные многотерминальные системы разделения времени*. В таких системах компьютер становился доступным сразу нескольким пользователям. Индивидуальные терминалы пользователей обеспечивали удобство диалога с компьютером, а малое время реакции компьютера в диалоге делало незаметным его параллельную работу с другими пользователями. На этом этапе ввод и вывод данных стали распределенными, а вычислительные ресурсы компьютера оставались централизованными.

Использование индивидуальных терминалов привело к постановке задачи доступа к компьютеру с мест, удаленных от него на сотни, а то и тысячи километров. Для решения этой поставленной задачи терминалы соединялись с компьютерами через телефонные сети с помощью *модемов*. В дальнейшем для совместной работы стали связывать компьютер с компьютером. *Компьютеры получили возможность обмениваться данными в автоматическом режиме, что является базовым механизмом любой компьютерной сети*. Так появились первые прообразы глобальных компьютерных сетей. В процессе их становления были предложены и отработаны многие основные идеи и концепции современных сетей.

В начале 70-х годов 20-го столетия появились большие интегральные схемы. Их сравнительно невысокая стоимость и высокие функциональные возможности привели к созданию относительно дешевых мини-компьютеров, ставшими реальными конкурентами мэйнфреймов. Создание мини-компьютеров привело к тому, что *предприятия и организации стали соединять принадлежащие им мини-компьютеры и разрабатывать программное обеспечение, необходимое для их взаимодействия. В результате появились первые локальные вычислительные сети.* Для соединения компьютеров использовались разнообразные *нестандартные* устройства со своим способом представления данных, своими типами кабелей и т. п. Такие устройства могли соединять только те типы компьютеров, для которых были разработаны.

В середине 80-х годов положение дел в локальных сетях стало кардинально меняться. Утвердились стандартные технологии объединения компьютеров в сеть – Ethernet, Arcnet, Token Ring. *Для создания сети теперь достаточно было приобрести и установить стандартное коммуникационное оборудование и программное обеспечение для компьютера и линии связи.* После этого сеть начинала работать и присоединение каждого нового компьютера не вызывало никаких проблем.

Сегодня компьютерные сети продолжают быстро развиваться. Разрыв между локальными и глобальными сетями постоянно сокращается. Появились высокоскоростные территориальные каналы связи, не уступающие по качеству кабельным системам локальных сетей. В гло-

бальных сетях появляются службы доступа к ресурсам, такие же удобные, как и службы локальных сетей. Подобные примеры в большом количестве демонстрирует самая популярная глобальная сеть – Internet.

Изменяются и локальные сети. Для них создано разнообразное коммуникационное оборудование (коммутаторы, маршрутизаторы, шлюзы), благодаря чему появилась возможность построения больших корпоративных сетей со сложной структурой и тысячами компьютеров.

Возродился интерес к крупным компьютерам. На новом витке эволюционной спирали мэйнфреймы стали возвращаться в компьютерные системы, но уже как полноправные сетевые узлы, поддерживающие стандартные технологии объединения и правила (протоколы) обмена информацией компьютеров в сети.

Проявилась еще одна очень важная тенденция, затрагивающая все сети. Изменилась их информационное наполнение. В них стала обрабатываться несвойственная ранее вычислительным сетям информация − голос, видеоизображения, рисунки. Сети перестали быть только вычислительными. Скорее их можно назвать информационными.

Что дает предприятию использование сетей

Этот вопрос можно уточнить следующим образом: в каких случаях развертывание на предприятии компьютерных сетей предпочтительнее использования автономных компьютеров или многомашинных систем? Какие новые возможности появляются на предприятии с появлением там компьютерной сети? И, наконец, всегда ли предприятию нужна сеть?

Если не вдаваться в частности, то *конечной целью использования компьютерных сетей на предприятии является повышение эффективности его работы, которое может выражаться, например, в увеличении прибыли предприятия.*

Использование вычислительных сетей дает предприятию следующие возможности:

- разделение дорогостоящих ресурсов, таких, как принтеры для печати документов; базы данных с коллективным доступом; совместное использование файлов
- совершенствование коммуникаций, например, передача сообщений и электронная почта;
- улучшение доступа к информации;
- быстрое и качественное принятие решений;
- свобода в территориальном размещении компьютеров.

Основные программные и аппаратные компоненты сети

Даже при поверхностном рассмотрении работы в сети становится ясно, что вычислительная сеть – это сложный комплекс взаимосвязанных и согласованно функционирующих программных и аппаратных компонентов. Изучение сети в целом предполагает знание принципов работы ее отдельных элементов:

- компьютеров;
- коммуникационного оборудования;
- операционных систем;
- сетевых приложений.

Весь комплекс программно-аппаратных средств сети может быть описан многослойной моделью.

В основе любой сети лежит *аппаратный слой (стандартная компьютерная аппаратура).* В настоящее время в сетях широко и успешно применяются компьютеры различных классов – от персональных компьютеров до мэйнфреймов и суперкомпьютеров. Набор компьютеров в сети должен соответствовать набору разнообразных задач, решаемых сетью.

Второй слой – это коммуникационное оборудование. Хотя компьютеры и являются центральными элементами обработки данных в сетях, в последнее время не менее важную роль стали играть коммуникационные устройства. Сегодня коммуникационное устройство может представлять собой сложный специализированный мультипроцессор, который нужно конфигурировать, оптимизировать и администрировать. Коммуникационные устройства из вспомогательных компонентов сети превратились в основные и наряду с компьютерами и системным программным обеспечением влияют как на характеристики сети, так и на её стоимость.

Третьим слоем, образующим программную платформу сети, являются операционные системы (ОС). При проектировании сети важно учитывать, насколько просто данная операционная система может взаимодействовать с другими ОС сети, насколько она обеспечивает безопасность и защищенность данных, до какой степени она позволяет наращивать число пользователей, можно ли перенести ее на компьютер другого типа и многие другие соображения.

Самым верхним слоем сетевых средств являются различные сетевые приложения, такие как сетевые базы данных, почтовые системы, средства архивирования данных, системы автоматизации коллективной работы и др. Очень важно представлять диапазон возможностей, предоставляемых приложениями для различных областей применения, а также знать, насколько они совместимы с другими сетевыми приложениями и операционными системами.

Подключение компьютера к сети

Речь пойдет о том, как выполнить объединение компьютеров в сеть. Способы объединения зависят от того, какая создается сеть – локальная или глобальная.

Подключение к локальной сети

Подключение компьютера к ***локальной сети*** выполняется при помощи специального устройства – сетевого адаптера (сетевой карты), который приобретается отдельно. Существует большое количество сетевых адаптеров, выбор адаптера зависит от топологии сети. Топология – раздел математики, изучающий свойства геометрических фигур. Для компьютерных сетей – это способ организации физических связей между компьютерами (узлами) сети. Простейшая локальная сеть имеет или *шинную* или *звездообразную* топологию.

В случае ***шинной топологии*** (рис. 91) используется длинный кабель, в разрывы которого подключаются сетевые адаптеры рабочих станций. Это более дешевый способ объединения компьютеров в сеть, но он обладает существенным недостатком – при обрыве кабеля вся сеть выходит из строя.

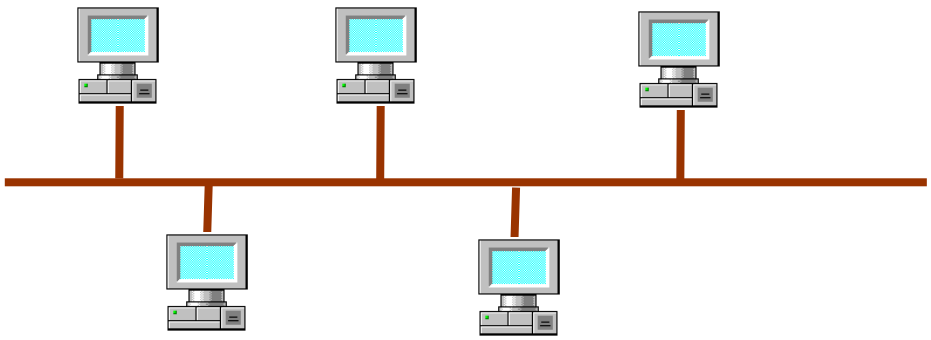


Рис. 91. Сеть с шинной топологией

Если в шинной топологии компьютер-отправитель и получатель расположены не на одном кабеле, то для их связи используется специальное устройство пересылки сообщений – ***маршрутизатор***. Он копирует посылаемое по одному кабелю сообщение и передает его по другому.

В случае ***звездообразной топологии*** (рис. 92) каждый компьютер подключается своим кабелем к другому специальному устройству – ***концентратору*** (разветвителю). При обрыве одного из кабелей сеть продолжает функционировать.

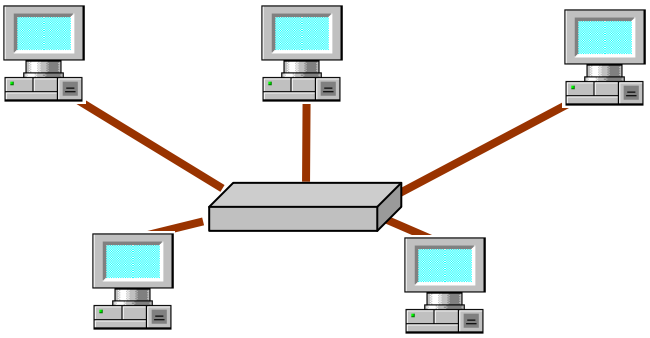


Рис. 92. Сеть со звездообразной топологией

Возможны и другие варианты топологии сети (например, кольцевая топология), а также более сложные варианты топологии (например, смешанная топология).

Подключение к глобальной сети

Для физического подключения компьютера к ***глобальной сети*** *нужно подключить к компьютеру модем и соединить его с обычной телефонной линией*. Существуют и другие способы физического подключения к глобальной сети. Например, можно подключиться к оптоволоконной линии связи, идущей от включенного в сеть узла. Конечно, этот способ более дорог, так как требует прокладки дорогостоящего оптоволоконного кабеля. В мире существует много глобальных сетей. Наиболее известны среди них ***Internet***, ***Compuserve***, ***Relcom***, ***FIDONET*** и др. Некоторые глобальные сети охватывают отдельные страны или регионы, другие – весь мир. В глобальной сети есть компьютеры, специально выделенные для работы в качестве почтовых серверов. Такие компьютеры подключены сразу к нескольким телефонным линиям или к другим подобным каналам.

Адресация компьютеров

При объединении компьютеров в сеть появляется проблема их адресации. На практике обычно используется сразу несколько схем, Каждый адрес используется в той ситуации, когда соответствующий вид адресации наиболее удобен. Наибольшее распространение получили три схемы адресации узлов.

Аппаратные (hardware) адреса. Эти адреса предназначены для сети небольшого или среднего размера, поэтому они не имеют иерархической структуры. Типичным представителем адреса такого типа является адрес сетевого адаптера локальной сети. Такой адрес обычно используется только аппаратурой, поэтому его стараются сделать по воз-

можности компактным и записывают в виде двоичного или шестнадцатеричного значения. При задании аппаратных адресов обычно не требуется выполнение ручной работы, так как они либо встраиваются в аппаратуру компанией-изготовителем, либо генерируются автоматически при каждом новом запуске оборудования. Помимо отсутствия иерархии, использование аппаратных адресов связано еще с одним недостатком – при замене аппаратуры, например, сетевого адаптера, изменяется и адрес компьютера

Символьные адреса или *имена.* Эти адреса предназначены для запоминания людьми и поэтому обычно несут смысловую нагрузку. Символьные адреса легко использовать как в небольших, так и крупных сетях. Для работы в больших сетях символьное имя может иметь сложную иерархическую структуру, например, ftp://archl.ucl.ac.uk.

Числовые составные адреса. Символьные имена удобны для людей, но из-за переменного формата и потенциально большой длины их передача по сети не очень экономична. Поэтому во многих случаях для работы в больших сетях в качестве адресов узлов используют числовые составные адреса.

В современных сетях для адресации узлов применяются, как правило, одновременно все три приведенные выше схемы. Пользователи адресуют компьютеры символьными именами, которые автоматически заменяются в сообщениях, передаваемых по сети, на числовые номера. С помощью этих числовых номеров сообщения передаются из одной сети в другую, а после доставки сообщения в сеть назначения вместо числового номера используется аппаратный адрес компьютера. Обычно такая схема характерна даже для небольших автономных сетей.

Проблема установления соответствия между адресами различных типов, которой занимается *служба разрешения имен,* может решаться как полностью централизованными, так и распределенными средствами.

Организация работы в сети

Для работы в глобальной сети одного физического подключения мало. После физического подключения к глобальной сети необходимо позвонить в организацию, которая предоставляет услуги в определенной сети (такие организации называются *провайдерами* – поставщиками услуг). Телефоны и адреса таких организаций известны. Существуют и другие формы приобретения услуг провайдера.

При выборе сети нужно знать, что сети бывают платные и бесплатные. Платные сети обеспечивают высокий уровень сервиса и быстроту работы в сети.

Можно подключиться к глобальной сети, используя специальное программное обеспечение. В этом случае можно говорить об интеграции локальной и глобальной сети, все *пользователи локальной сети работают с глобальной сетью через сервер, который является зарегистрированным узлом глобальной сети.*

Организация работы в локальной сети зависит от сетевого программного обеспечения. По типу используемого программного обеспечения локальные сети можно разделить на две группы – *сети с выделенными файл-серверами* и *одноранговые сети без выделенного файл-сервера.*

В сетях *с выделенным файл-сервером* на сервере устанавливается специальная сетевая операционная система, которая организует работу в сети, предоставляет в распоряжение пользователей сетевые ресурсы (дисковые устройства, сетевые принтеры и т. д.) и выполняет ряд других специальных задач.

В *одноранговых сетях* нет специально выделенных компьютеров. Пользователь любой рабочей станции легко может превратить свой компьютер в сервер, обеспечив доступ к его ресурсам для других пользователей. Для организации одноранговых сетей можно использовать такие операционные системы, как любые операционные системы семейства Windows. Одноранговые сети удобны в тех случаях, когда пользователи в процессе работы интенсивно обмениваются данными или работают над одним проектом. Обычно за работу сети отвечает специально подготовленный человек – администратор сети.

В крупных организациях в сети могут работать несколько групп пользователей. В этом случае администратор может ограничить доступ некоторых групп или отдельных пользователей к некоторым ресурсам сети. Причем в одноранговых сетях эти возможности ограничены.

В сетях же с выделенным файл-сервером администратор может запретить доступ к отдельным дискам или каталогам файл-сервера (в этом случае пользователь их даже не видит) или ограничить возможности работы с файлами на таких дисках, разрешив, например, только чтение файла.

Модель OSI

Функционирование глобальных сетей основано на модели *взаимодействия открытых систем (Open System Interconnection, OSI)*. Термин «взаимодействие открытых систем» относится к процедурам передачи данных между системами, которые «*открыты*» друг другу благодаря совместному использованию ими соответствующих *стандартов*. Назначение этой модели состоит также в определении областей разработки и усовершенствования стандартов, выработке общих принципов их согласованности.

Если две сети построены с соблюдением принципов открытости, то это дает следующие преимущества:

- возможность построения сети из аппаратных и программных средств различных производителей, придерживающихся одного и того же стандарта;
- возможность безболезненной замены отдельных компонентов сети другими, более совершенными, что позволяет сети развиваться с минимальными затратами;
- возможность легкого сопряжения одной сети с другой;
- простота освоения и обслуживания сети.

Ярким примером открытой системы является международная сеть Internet. Эта сеть развивалась в полном соответствии с требованиями, предъявляемыми к открытым системам.

В модели OSI средства взаимодействия делятся на семь уровней: прикладной, представительный, сеансовый, транспортный, сетевой, канальный и физический. Каждый уровень имеет дело с одним определенным аспектом взаимодействия сетевых устройств. Любой уровень состоит из объектов. Объекты одного и того же уровня для обеспечения взаимодействия могут связываться друг с другом. *Взаимодействие объектов, расположенных на одном и том же уровне, определяется стандартами, называемыми протоколами.* Кроме этого, объекты N-уровня взаимодействуют с объектами соседних уровней. От объектов N-1 уровня объекты N-уровня получают разного вида сервис. В свою очередь объекты N-уровня предоставляют сервис объектам N+1-уровня. Протоколы реализуются не только компьютерами, но и коммуникационным оборудованием. Действительно, в общем случае связь компьютеров в сети осуществляется не напрямую, а через различные коммуникационные устройства. В зависимости от типа устройства в нем должны быть встроенные средства, реализующие тот или иной набор протоколов.

Итак, пусть приложение обращается с запросом к прикладному уровню. На основании этого запроса *программное обеспечение прикладного уровня формирует сообщение стандартного формата.* Обычное *сообщение состоит из заголовка и поля данных*. Заголовок содержит служебную информацию, которую необходимо передать через сеть прикладному уровню машины-адресата, чтобы сообщить ему, какую работу надо выполнить. Поле данных сообщения может быть пустым или содержать какие-либо данные, например, те, которые необходимо записать в удаленный файл.

После формирования сообщения прикладной уровень направляет его вниз представительному уровню. *Протокол представительного уров-*

ня на основании информации, полученной из заголовка прикладного уровня, выполняет требуемые действия и добавляет к сообщению собственную служебную информацию – заголовок представительного уровня, в котором содержатся указания для протокола представительного уровня машины-адресата. Полученное в результате сообщение передается вниз сеансовому уровню, который в свою очередь добавляет свой заголовок, и т. д. Наконец, сообщение достигает нижнего, физического уровня, который собственно и передает его по линиям связи машине-адресату. К этому моменту сообщение «обрастает» заголовками всех уровней.

Когда сообщение по сети поступает на машину-адресат, оно принимается ее физическим уровнем и последовательно перемещается вверх с уровня на уровень. Каждый уровень анализирует и обрабатывает заголовок своего уровня, выполняя соответствующие данному уровню функции, а затем удаляет этот заголовок и передает сообщение вышележащему уровню.

Уровни модели OSI

Физический уровень

Физический уровень имеет дело с передачей битов по физическим каналам связи, таким, например, как коаксиальный кабель, витая пара, оптоволоконный кабель или цифровой территориальный канал.

Канальный уровень

Одной из задач канального уровня является *проверка доступности среды передачи.* Другой задачей канального уровня является *реализация механизмов обнаружения и коррекции ошибок.* Для этого на канальном уровне биты группируются в наборы, называемые *кадрами (frames).* Иными словами, доставка данных внутри сети обеспечивается соответствующим канальным уровнем

Сетевой уровень

Сетевой уровень служит *для образования единой транспортной системы, объединяющей несколько сетей*, причем эти сети могут использовать различные принципы передачи сообщений между конечными узлами и обладать произвольной структурой связей. В отличие от канального уровня сетевой уровень занимается доставкой данных между сетями. Сообщения сетевого уровня принято называть **пакетами** (packets).

Примерами протоколов сетевого уровня являются протокол межсетевого взаимодействия IP стека TCP/IP и протокол межсетевого обмена пакетами IPX стека Novell.

Транспортный уровень

На пути от отправителя к получателю пакеты могут быть искажены или утеряны. *Транспортный уровень обеспечивает приложениям или верхним уровням стека – прикладному и сеансовому – передачу данных с той степенью надежности, которая им требуется.*

Сеансовый уровень

Сеансовый уровень обеспечивает управление диалогом: *фиксирует, какая из сторон является активной в настоящий момент, предоставляет средства синхронизации.* Последние позволяют вставлять контрольные точки в длинные передачи, чтобы в случае отказа можно было вернуться назад к последней контрольной точке, а не начинать все с начала.

Представительный уровень

Представительный уровень имеет дело с формой представления передаваемой по сети информации, не меняя при этом ее содержания. За счет уровня представления информация, передаваемая прикладным уровнем одной системы, всегда понятна прикладному уровню другой системы. На этом уровне может выполняться шифрование и дешифрование данных, благодаря которому секретность обмена данными обеспечивается сразу для всех прикладных служб.

Прикладной уровень

Прикладной уровень – это в действительности просто *набор разнообразных протоколов, с помощью которых пользователи сети получают доступ к разделяемым ресурсам*, таким как файлы, принтеры или гипертекстовые Web-страницы, а также организуют свою совместную работу, например, с помощью протокола электронной почты. Единица данных, которой оперирует прикладной уровень, обычно называется сообщением (message).

Тенденция к сближению локальных и глобальных сетей

Одним из проявлений сближения локальных и глобальных сетей является появление *сетей масштаба большого города (MAN),* занимающих промежуточное положение между локальными и глобальными сетями. При достаточно больших расстояниях между узлами они обладают качественными линиями связи и высокими скоростями обмена, даже более высокими, чем в классических локальных сетях. Как и в случае локальных сетей, при построении MAN уже существующие линии связи не используются, а прокладываются заново.

За счет новых сетевых технологий и, соответственно, нового оборудования, рассчитанного на более качественные линии связи, скорости передачи данных в уже существующих коммерческих глобальных сетях нового поколения приближаются к традиционным скоростям локальных сетей.

Кроме того, процесс переноса служб и технологий из глобальных сетей в локальные приобрел такой массовый характер, что появился даже специальный термин – intranet-технологии (intra – внутренний), обозначающий применение служб внешних (глобальных) сетей во внутренних – локальных.

Глобальная сеть Интернет

История Интернет

Попытки передачи информации между компьютерами по специальным или телефонным линиям предпринимались еще на заре всеобщей компьютеризации. История Интернет в некотором смысле началась с 1958 года, когда в ответ на запуск первого спутника, США создали организацию под названием ARPA (Advanced Research Projects Agency – Бюро перспективных исследований Министерства обороны США). До 68-го года внутри ARPA и в других организациях велась работа по соединению компьютеров. В конце 60-х – начале 70-х годов Министерство обороны США приступило к разработке системы связи, которая должна была соединить между собой компьютеры всех центров противоракетной обороны страны. К надежности системы предъявлялись высокие требования: система должна быть децентрализованной. Выход из строя любых ее составляющих (например, в случае точечного ядерного удара) не должен был приводить к разрушению системы и ухудшать качество и скорость связи между участниками информационного обмена. Целью ARPA было создание надежной и гибкой компьютерной сети, способной функционировать в экстремальных условиях военного времени. Требовалась гибкость и надежность при минимальных затратах. ARPA успешно решила эту задачу, разработав технологию *коммутации пакетов*. Следуя этой технологии, передаваемые *данные разбивались на «пакеты»,* каждый из которых *имел свой «адрес назначения», и передавались в линию связи*. Наличие «адреса назначения» позволяло легко отыскать «адресата» и передать ему все необходимые данные. Система получила название ARPAnet и использовалась для передачи данных между компьютерами и электронной почты.

В начале 70-х годов при поддержке ARPA была разработана новая технология передачи данных – правила передачи различных типов данных между разнородными компьютерными сетями. Эти правила

(протоколы), получившие общее название INTERNET, сделали возможным создание всемирной сети, которая объединяет компьютеры любых типов и позволяет передавать практически любую информацию. Она и сыграла огромную роль в создании этой новой системы связи.

К 1978-му году были выработаны все базовые протоколы, которые и сейчас используются в Интернет. В 1982 году была основана Европейская UNIX Сеть (EUnet). До этого в Сеть входили только США, Канада и Великобритания. Система доменных имен (символьные адреса) появилась в 1984 году. В 1991-м была разработана технология WWW. К этому моменту ARPAnet уже перестала официально существовать. Пришло время современного Интернета. В 1991-м году Россия присоединилась к Интернет. К 1992 году в Сети было более миллиона компьютеров. За 1993-й год Веб вырос в три с половиной тысячи раз. В 1994-м году Интернет исполнилось 25 лет. С тех пор Сеть достигла глобального распространения, но принципиально не изменилась. Были придуманы множество новых технологий, улучшились каналы связи, увеличилось до десятков миллионов количество компьютеров, а количество пользователей – до сотен миллионов. В результате Интернет обрел общеизвестность, и он стал коммерчески выгодным не только для тех, кто предоставляет доступ в Сеть. Уже в 94-м году можно было заказать пиццу на дом через Интернет. На август 2000-го года насчитывалось более двадцати миллионов Web-сайтов, т. е. в десять раз больше, чем в 96-м году.

В настоящее время Интернет не является отдельной сетью – на самом деле это сообщество сетей.

Как организована работа Интернета?

Для того чтобы компьютеры разных моделей, работающие под управлением различных программ, могли общаться друг с другом в Интернет, надо было разработать единый для всех язык. Точнее, не язык, а несколько наборов правил, каждый из которых служит определенной цели. Эти наборы правил называются протоколами.

Кроме того, понадобилось установить способ различения одного компьютера в Сети от другого, а также – способ определения их места нахождения. Этой цели служат IP адреса (числовые адреса, о которых речь шла ранее).

Сервис в Интернет

Интернет позволяет человеку расширить свои познания в любой, даже самой немыслимой сфере деятельности или исследований. А так как развитию Интернет, с одной стороны, способствовали коммерческие организации, фирмы, использующие Сеть для обмена деловой ин-

формацией и публикации рекламы, а с другой – студенты, помещающие на всеобщее обозрение множество материалов развлекательного характера, то здесь много полезного для себя найдут все – от бизнесменов до любителей со вкусом отдохнуть.

Спектр услуг в Интернет к настоящему времени стал просто необозрим, поэтому начнем с самых известных и доступных.

Электронная почта

Этот сетевой сервис многим уже знаком. Электронная почта в Интернет практически ничем не отличается от электронной почты в локальной сети, кроме того, что ваш адресат может находиться в любой точке света, доставка же корреспонденции по-прежнему осуществляется в считанные минуты.

При пользовании *e-mail* из-за ее оперативности может сложиться ощущение телефонной связи, но всегда следует осознавать, что это все же почта. Однако нужно помнить, что *e-mail* не обладает той степенью приватности, как обычная почта. Анонимность также исключена: источник прослеживается без труда.

File Transfer Protocol (FTP)

С помощью FTP осуществляется обмен файлами между компьютерами. На множестве FTP-серверов можно найти полезные утилиты, демонстрационные версии программ, мультимедийные ролики, картинки и т. д. Доступ к большинству таких FTP-серверов свободный – в качестве входного пароля вам достаточно набрать ваш адрес электронной почты.

Видеоконференции реального времени (RTVC)

Видеоконференция – это взаимное общение двух и более лиц с использованием мультимедийных возможностей компьютеров по сети Интернет. Компьютер должен иметь высокоскоростной канал выхода в Интернет, видеокамеру, микрофон и звуковые колонки. Технические устройства, обеспечивающие проведение видеоконференций, включают мониторы высокого разрешения, сканеры, средства представления и сжатия изображений, аппаратуру связи. Все эти устройства работают в режиме реального времени.

Основные понятия World Wide Web

Сегодня Интернет используется как источник разносторонней информации по различным областям знаний. Большинство документов, доступных на серверах Интернета, имеют *гипертекстовый формат.* World Wide Web, Всемирная паутина, WWW, Web, Веб – это все назва-

ния одного и того же сервиса, который был придуман в 1991 году и использует протокол HTTP для передачи гипертекстовых документов от Web-сервера к клиентам. От обычного гипертекста WWW отличается главным образом тем, что позволяет устанавливать ссылки не только на соседний файл, но и на файл, находящийся на компьютере в другом полушарии Земли. От пользователя не требуется никаких усилий – компьютер установит связь самостоятельно. Популярность WWW обусловлена тем, что он был первым способом получения информации любого рода из Интернета в интуитивно понятной и красивой форме.

На сегодняшний день это наиболее продвинутый и интересный ресурс – гипертекстовая система навигации в Интернет.

Среда WWW не имеет централизованной структуры. Она пополняется теми, желает разместить в Интернете свои материалы, и может рассматриваться как *информационное пространство.* Как правило, документы WWW хранятся на постоянно подключенных к Интернету компьютерах − *Web-серверах.* Обычно на Web-сервере размещают не отдельный документ, а группу взаимосвязанных документов. Такая группа представляет собой *Web-узел* (жаргонный термин − *Web-сайт).* Размещение подготовленных материалов на Web-узле называется *Web-издание,* или *WWW публикацией.*

Web-страница. Отдельный документ World Wide Web называют *Web-страницей.* Обычно это комбинированный документ, который может содержать текст, графические иллюстрации, мультимедийные и другие вставные объекты. Для создания *Веб* страниц используется *язык HTML (HyperText Markup Language* − язык разметки гипертекста), который при помощи вставленных в документ *тегов* описывает логическую структуру документа, управляет форматированием текста и размещением вставных объектов. *Интерактивные* Web-узлы получают информацию от пользователя через *формы* и генерируют запрошенную Web-страницу с помощью специальных программ *(сценариев CGI), динамического HTML* и других средств.

Гиперссылки. Отличительной особенностью среды World Wide Web является наличие средств перехода от одного документа к другому, тематически с ним связанному, без явного указания адреса. Связь между документами осуществляется при помощи *гипертекстовых ссылок* (или просто *гиперссылок).* Гиперссылка − это выделенный фрагмент документа (текст или иллюстрация), с которым ассоциирован адрес другого Web-документа. При использовании гиперссылки (обычно для этого требуется навести на нее указатель мыши и один раз щелкнуть) происходит *переход по гиперссылке* − открытие Web-страницы, на которую указывает ссылка. Механизм гиперссылок позволяет организовать тема-

тическое путешествие по World Wide Web без использования (и даже знания) адресов конкретных страниц.

Адресация документов. Для записи адресов документов Интернета (Web-страниц) используется форма, называемая *адресом URL* **(**Uniform Resourse Locator). Обозначение URL состоит из трех частей: *первая* указывает тип связи, который следует установить с нужным вам источником (протокол), *вторая* – имя требуемого сервера, и *третья* – полное имя ресурса, т. е. имя файла на сервере, включающее путь к нему. Например, **http**://***www.gismeteo.ru***/towns/29430.htm. Здесь разными шрифтами выделены три части URL.

Преобразование адреса URL в цифровую форму *IP-адреса* производит *служба имен доменов (Domain Name Service, DNS)*. В качестве разделителя в пути поиска документа Интернета всегда используется символ косой черты.

Навигация в World Wide Web

Документы Интернета предназначены для отображения в *электронной форме,* причем автор документа не знает, каковы возможности компьютера на котором документ будет отображаться. Поэтому язык *HTML* обеспечивает не столько форматирование документа, сколько описание его логической структуры. Форматирование и отображение документа на конкретном компьютере производится специальной программой − *браузером* (от английского слова *browser)*.

Основные функции браузеров следующие:

- установление связи с Web-сервером, на котором хранится документ, и загрузка всех компонентов комбинированного документа;
- интерпретация тегов языка *HTML*, форматирование и отображение Web-страницы в соответствии с возможностями компьютера, на котором браузер работает;
- предоставление средств для отображения мультимедийных и других объектов, входящих в состав Web-страниц, а также механизма расширения, позволяющего настраивать программу на работу с новыми типами объектов;
- обеспечение автоматизации поиска Web-страниц и упрощение доступа, к Webстраницам, посещавшимся ранее.
- предоставление доступа к встроенным или автономным средствам для работы с другими службами Интернета.

В настоящее время широкой известностью в мире пользуются несколько браузеров. Фактически это мощные системы, обеспечивающие не только просмотр информации, но и ее быстрый и целенаправленный

поиск, все сервисы Интернет и даже средства создания Web-страниц. Среди всех браузеров наибольшее распространение получил браузер Microsoft Internet Explorer (рис. 93), который входит во все последние версии операционных систем семейства Windows в качестве стандартного компонента системы.

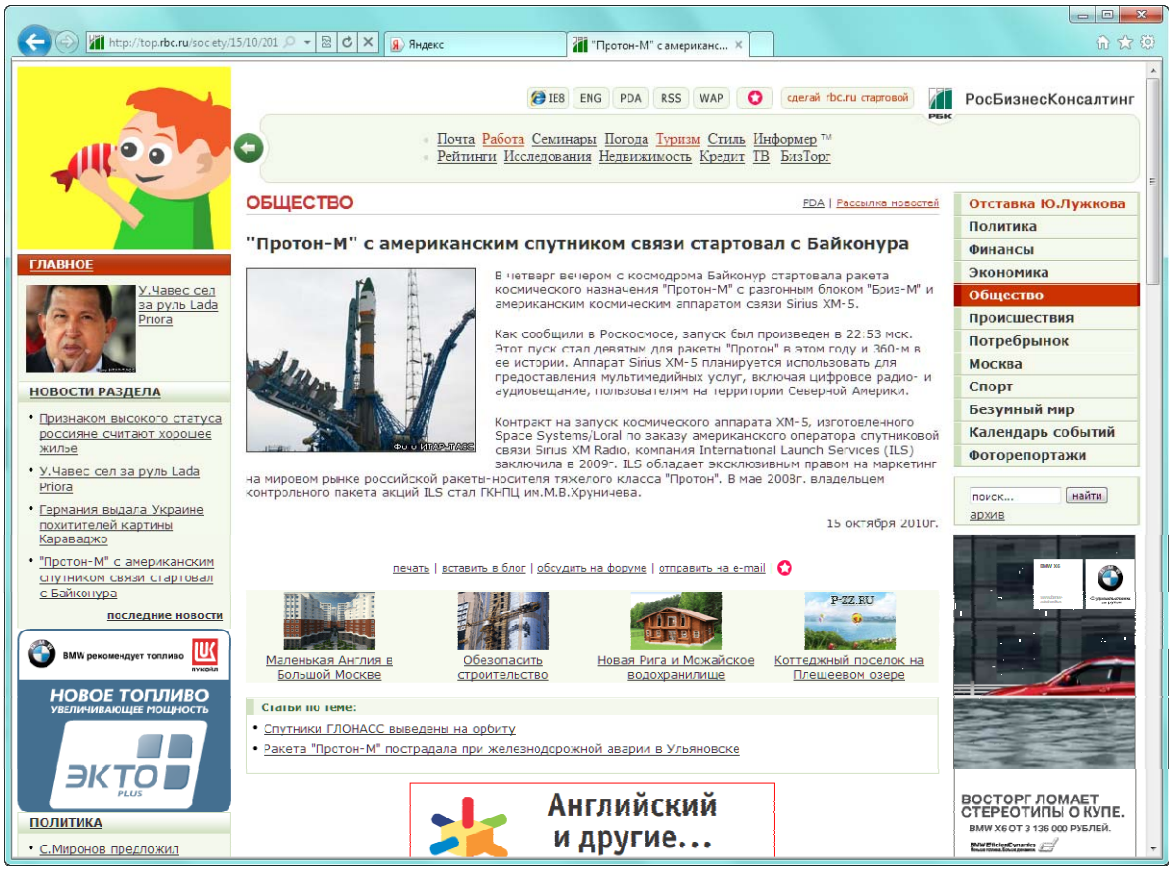


Рис. 93. Окно браузера Microsoft Internet Explorer

Браузер – это программа, которая понимает протокол HTTP и специальный язык HTML, на котором создается содержимое web-страниц, доступных в WWW. Принцип работы браузера следующий. Пользователь набирает адрес web-страницы, находящейся на www-сервере. Браузер обращается к серверу с запросом загрузки документа, расположенного по этому адресу. После загрузки документа браузер обрабатывает его и, если в нем есть картинки, также обращается к серверу с запросом об их загрузке, как и о загрузке других материалов документа. После этого браузер обрабатывает все пришедшие данные и отображает готовую страницу на экране. Некоторые элементы страницы (текст, картинки, кнопки) могут быть ссылками. Если нажать на них, то браузер пошлет запрос серверу, указанному в ссылке, чтобы загрузить с него документ, который в ссылке обозначен. Таким образом, можно передвигаться от документа к документу, от сервера к серверу, что превращает весь Ин-

тернет в одну гигантскую Сеть, связывающую документы и сервера друг с другом нитями гиперссылок. Кроме того, на странице могут быть места для ввода какой-либо информации и ссылки на программу на сервере, которая должна обрабатывать эту информацию.

Как и в любом приложении Windows, в меню и панели инструментов браузера сосредоточен доступ к функциям, обеспечивающим эффективную работу пользователя. Например, можно сохранить загруженную из Интернета Web-страницу на компьютере локально для последующего доступа к ней без выхода в Интернет. Можно сохранить ссылку на понравившуюся страницу в специальной папке «Избранное», распечатать страницу, отправить страницу или ссылку на нее по электронной почте и многое другое.

Поиск информации в Интернет

В Интернет обращаются за определенной информацией. Чтобы открыть нужную Web-страницу, надо иметь либо ее адрес, либо другую страницу со ссылкой на нее. **Если** нет ни того, ни другого, обращаются к ***поисковым системам***. Поисковая система представляет собой специализированный Web-узел. Пользователь сообщает поисковой системе данные о содержании искомой Web-страницы, а поисковая система выдает список гиперссылок на страницы, на которых упоминаются соответствующие сведения. Поисковые системы 'классифицируют по методам поиска.

Поисковые каталоги предназначены для поиска по темам. Пользователь «погружается» в иерархическую структуру разделов и подразделов, на нижнем уровне которой располагается относительно небольшое число ссылок, заслуживающих внимания. Поисковый каталог обеспечивает высокое качество поиска.

Поисковый индекс обеспечивает поиск по заданным ключевым словам. В результате поиска формируется набор гиперссылок на Web-страницы, содержащие указанные термины. Поисковые индексы предоставляют грандиозную широту поиска.

Структурированием данных, входящих в базу поисковых каталогов, занимаются люди, а создание баз для поисковых индексов выполняется автоматическими средствами. Соответственно, в среднем, поисковые каталоги предоставляют доступ к меньшему числу Web-ресурсов, чем поисковые индексы, но они точнее указывают **на *основные*** ресурсы Сети. Таким образом, при проведении первичного поиска по конкретной теме целесообразно использовать поисковые каталоги. Для специалистов, хорошо знакомых с ресурсами Интернета по своей специально-

сти, более полезны поисковые индексы. Они позволяют разыскивать малоизвестные и узкоспециализированные ресурсы.

Многие современные поисковые системы сочетают в себе оба вышеуказанных поиска и позволяют использовать наиболее подходящий. Для многих поисковая система превращается в отправную точку для работы в Интернете, средство, через которое пользователь получает доступ к нужной ему информации. Это привело к появлению ***Web-порталов,*** специализированных страниц, обеспечивающих удобный интерфейс доступа к поисковым системам, а также к другим Web-узлам, представляющим всеобщий интерес. Web-портал можно рассматривать как «окно» в World Wide Web.

Тематические порталы могут предлагать возможность ***поиска с классификацией.*** Они содержат относительно неизменный тематический список Web-странin в виде гиперссылок и учитывают число пользователей, которые воспользовались каждой из ссылок. Это число носит характер рейтинга, позволяющего оценить популярность соответствующей страницы.

Обзор русскоязычных поисковых систем

Продолжается информационный бум, растут количество и объемы серверов в WWW, увеличиваются мощности локальных сетей. Многим людям ежедневно приходится иметь дело с большими объемами текстов – это и новости, и официальные документы, и подшивки газет в электронном виде, и электронная почта, и Web-страницы, и документация. Очень важно уметь быстро искать и находить в этом море действительно нужную информацию. Как и во всем мире, в России разрабатывают всевозможные средства интеллектуализации поиска, но русскоязычные поисковые системы должны учитывать особенности русского языка.

Учет особенностей конкретного языка проводится на основе *морфологии языка*. Морфология языка – это область лингвистики, изучающая законы образования различных форм слов (словоформ).

Поисковая система ***с*** *учетом морфологии языка* умеет для всех слов этого языка делать анализ, то есть понимать, формой какого слова они являются. Каждому слову языка в начальной форме соответствует совокупность всех его словоформ, полученных при изменении слова по числам, падежам, родам и временам.

Среди широко известных поисковых систем (порталов) с учетом морфологии русского языка можно назвать следующие: Yandex (yandex.ru), Rambler (rambler.ru), Google (google.ru), и другие.

В каждый из поисковых систем существуют свои соглашения о синтаксисе запроса на поиск. Но чаще всего они сводятся к требованию

задавать набор ключевых слов для поиска, разделяя их между собой специальными знаками. Например, «математическая логика»+программа. Ключевые элементы, состоящие более чем из одного слова, ограничиваются двойными кавычками, а знак + соединяет отдельные элементы запроса в единое целое.

Результаты поиска выдаются в виде списка гиперссылок на ресурсы сети Интернет. Выбор любого элемента списка приводит к переходу на соответствующий сервер в сети. Если все найденные ссылки не помещаются на одной станице, они автоматически разбиваются на необходимое число страниц. Переход между этими страницами возможен с использованием соответствующих элементов навигации.

По мере работы в сети Интернет каждый пользователь накапливает список полезных адресов, на которых располагается интересующая его информация.

ПРОГРАММИРОВАНИЕ ДЛЯ ИНЖЕНЕРОВ

ГЛАВА 5. ПОНЯТИЯ И ПРИНЦИПЫ ПРОГРАММИРОВАНИЯ

Программирование – это совокупность процессов, связанных с разработкой программ и их реализацией.

В широком смысле к указанным процессам относят все технические операции, необходимые для создания программ, включая анализ требований, все стадии разработки, а также реализации в виде готового программного продукта.

В узком смысле под программированием часто понимают только процессы выбора структуры, кодирования и тестирования программ.

Основные понятия

Данные − описание сведений о реальном мире в формализованном виде, пригодном для передачи и обработки аппаратурой компьютера.

Алгоритм − предписание, задающее последовательность действий по переработке данных в результат решения задачи. Существует несколько способов записи алгоритмов (словесное описание, описание с помощью блок-схем и т. д.). Мы будем использовать описание алгоритма на языке, близком к естественному. Такое описание называется **псевдокодом**.

Язык программирования − средство описания данных и алгоритма их переработки.

Программа − описание данных задачи и последовательность инструкций по их обработке, которую должен выполнить компьютер.

Модуль − законченная часть программы, оформленная по определенным правилам. Модуль − своего рода упаковка, контейнер для процедур, скрывающий от внешнего мира (в том числе от других модулей) как их внутреннее устройство, так и вспомогательные процедуры.

Объект − это понятие, объединяющие в логически единое целое данные и средства манипулирования ими. Эти данные и средства могут относиться как к реально существующим, так и к придуманным объектам с их свойствами и поведением.

Событие

Под событием понимается любое действие со стороны пользователя или операционной системы, которое может требовать ответной реакции программы.

С содержанием этих понятий можно познакомиться в лабораторном практикуме.

Управление программой

В программировании различают два вида управления программой. Первый из них − программа управляется данными, второй − программа управляется событиями.

Программа, управляемая данными − это программы с фиксированной последовательностью выполнения. Разработчик программы задаёт последовательность выполнения операторов, и система строго ее соблюдает. Программа сама контролирует процесс своего выполнения от начала до конца и операционная система способна лишь досрочно прекратить выполнение программы, а также приостановить ее выполнение или перенаправить потоки ввода/вывода.

В случае программ, управляемых событиями, разработчик не может заранее предсказать последовательность выполнения операторов своего приложения, так как эта последовательность определяется на этапе выполнения кода. Алгоритм обработки данных в этом случае зависит от того, какое событие случилось в текущий или предыдущий момент времени, Большую часть времени приложение, управляемое событиями, находится в состоянии ожидания сообщений о произошедших событиях. Сообщения могут поступать от различных источников, но все они попадают в одну очередь системных сообщений. Только некоторые из них система передаст в очередь сообщений вашего приложения

Принципиально важным отличием Windows-приложений от приложений DOS является то, что все они − программы, управляемые событиями (event-driven applications).

Основные этапы развития технологии программирования

Технологией программирования называют совокупность методов и средств, используемых в процессе разработки программного обеспечения.

Рассмотрим этапы развития технологии программирования [5].

Первый этап − «стихийное» программирование. Этот этап охватывает период от момента появления первых вычислительных машин до середины 60-х годов XX века. В этот период практически отсутствовали сформулированные технологии, и программирование фактически было искусством. Первые программы имели простейшую структуру. Они состояли из собственно программы на машинном языке и обрабатываемых ею данных (рис. 94). Сложность программ в машинных кодах ограничивалась способностью программиста одновременно мысленно отслеживать последовательность выполняемых операций и местонахождение данных при программировании.

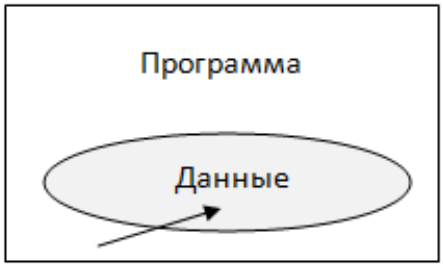


Рис. 94. Структура первых программ

Создание языков программирования высокого уровня, таких, как FORTRAN и ALGOL, существенно упростило программирование вычислений, снизив уровень детализации операций. Это, в свою очередь, позволило увеличить сложность программ.

Революционным было появление в языках средств, позволяющих оперировать подпрограммами. Подпрограмма – это часть общего кода программы, оформленная по специальным правилам, которая выполняет определенную подзадачу основной задачи.

Подпрограммы можно было сохранять и использовать в других программах. В результате были созданы огромные библиотеки расчетных и служебных подпрограмм, которые по мере надобности вызывались из разрабатываемой программы.

Типичная программа того времени состояла из основной программы, области глобальных данных и набора подпрограмм (в основном библиотечных), выполняющих обработку всех данных или их части (рис. 95).

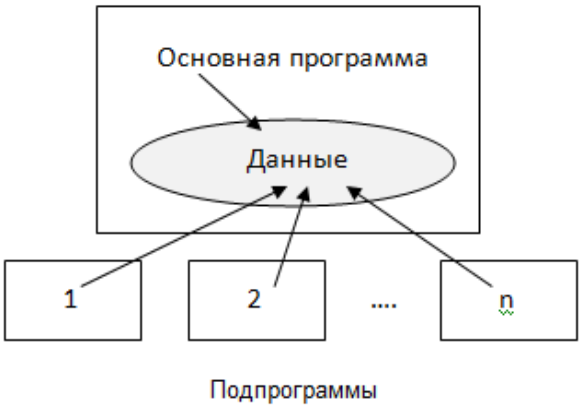


Рис. 95. Архитектура программы с глобальной областью данных

Слабым местом такой архитектуры было то, что при увеличении количества подпрограмм возрастала вероятность искажения части глобальных данных какой-либо подпрограммой. Чтобы сократить количество таких ошибок, было предложено в подпрограммах размещать локальные данные (рис. 96).

Сложность разрабатываемого программного обеспечения при использовании подпрограмм с локальными данными по-прежнему ограничивалась возможностью программиста отслеживать процессы обработки данных, но уже на новом уровне.

В начале 60-х годов XX в. разразился «кризис программирования».

Объективно все это было вызвано несовершенством технологии программирования. Прежде всего, стихийно использовалась разработка «снизу-вверх» − подход, при котором вначале проектировали и реализовывали сравнительно простые подпрограммы, из которых затем пытались построить сложную программу. В конечном итоге процесс тестирования и отладки программ занимал более 80% времени разработки, если вообще когда-нибудь заканчивался.

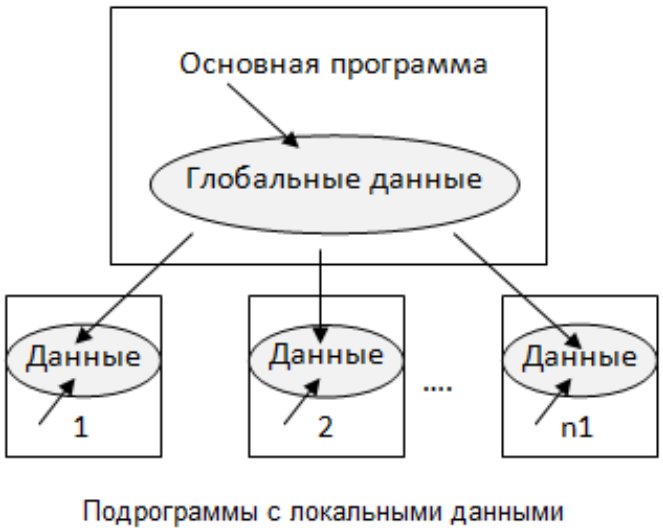


Рис. 96. Архитектура программы, использующей подпрограммы с локальными данными

Анализ причин возникновения большинства ошибок позволил сформулировать новый подход к программированию, который был назван «структурным».

Второй этап − структурный подход к программированию (60-70-е годы XX в.). В основе структурного подхода лежит декомпозиция (разбиение на части) сложных систем с целью последующей реализации в виде отдельных небольших (до 40−50 операторов) подпрограмм. Этот способ получил название процедурной декомпозиции.

В отличие от используемого ранее процедурного подхода к декомпозиции, структурный подход требовал представления задачи в виде иерархии подзадач простейшей структуры. Проектирование, таким образом, осуществлялось «сверху вниз». Тогда же появился *метод пошаговой детализации* проектирования алгоритмов.

Дальнейший рост сложности и размеров разрабатываемого программного обеспечения потребовал развития нового подхода к данным. Появились понятия простых и сложных (или структурных) данных. Одновременно усилилось стремление разграничить доступ к глобальным данным программы, чтобы уменьшить количество ошибок, возникающих при работе с глобальными данными. В результате появилась технология модульного программирования.

Модульное программирование предполагает выделение групп подпрограмм, использующих одни и те же глобальные данные в отдельно компилируемые модули (библиотеки подпрограмм) (рис. 97). Из таких модулей и собирается программа.

Практика показала, что структурный подход в сочетании с модульным программированием позволяет получать достаточно надежные программы, размер которых не превышает 100 000 операторов. Узким местом модульного программирования является то, что при увеличении размера программы с некоторого момента предусмотреть взаимовлияние отдельных частей программы становится практически невозможно. Для разработки программного обеспечения большого объема было предложено использовать объектный подход.

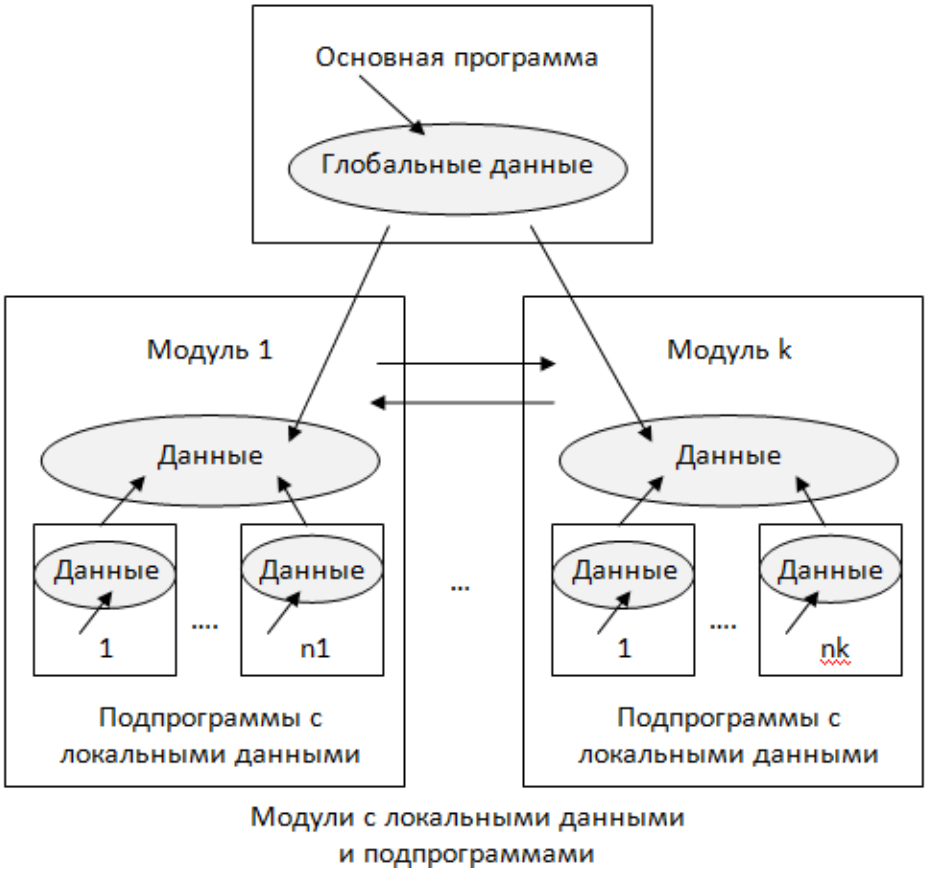


Рис. 97. Архитектура программы, состоящей из модулей

Третий этап − объектный подход к программированию (с середины 80-х до конца 90-х годов XX в.). *Объектно-ориентированное программирование* определяется как технология создания сложного программного обеспечения, основанная на представлении программы в виде совокупности объектов с присущими им свойствами.

Взаимодействие программных объектов в такой системе осуществляется путем передачи сообщений (рис. 98).

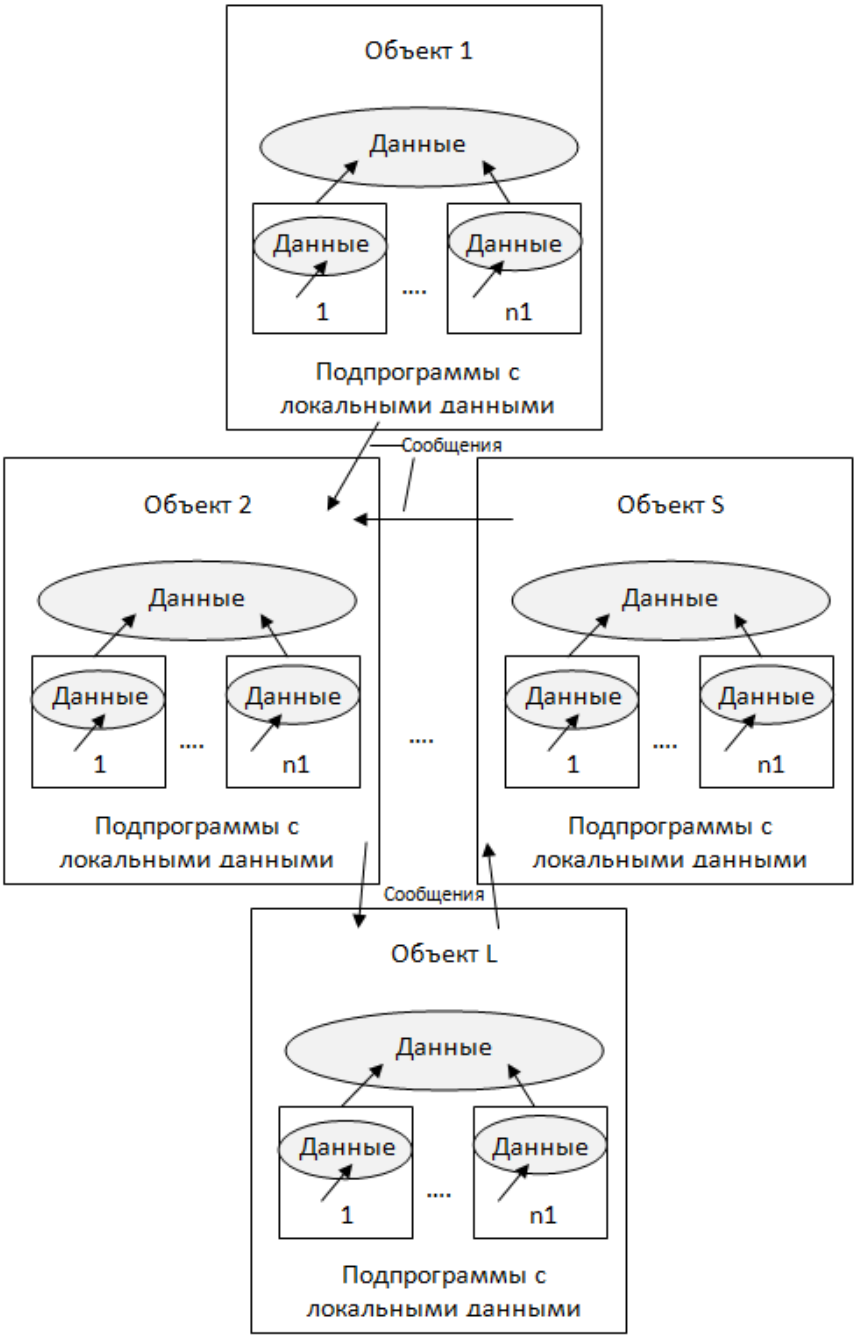


Рис. 98. Архитектура программы при объектно-ориентированном программировании

Основным достоинством объектно-ориентированного программирования по сравнению с модульным программированием является «более естественное» описание программой того процесса или явления, которое она моделирует. В конечном итоге, объектно-ориентированное программирование позволяет моделировать сложные процессы, для которых процедурное программирование малоэффективно.

С середины 90-х годов XX в. и до нашего времени развивается компонентный подход, который является дальнейшим развитием объектно-ориентированного подхода. Он предполагает построение программного обеспечения из отдельных компонентов физически отдельно существующих частей программного обеспечения, которые взаимодействуют между собой через *стандартизованные двоичные интерфейсы*. В отличие от обычных объектов объекты-компоненты можно собрать в динамически вызываемые библиотеки или исполняемые файлы, распространять в двоичном виде (без исходных текстов) и использовать в любом языке программирования, поддерживающем соответствующую технологию. Это позволяет программистам создавать продукты,

Появление нового подхода не означает, что отныне все программное обеспечение будет создаваться из программных компонентов, но анализ существующих проблем разработки сложного программного обеспечения показывает, что он будет применяться достаточно широко.

Основные принципы программирования

Какую программу можно считать «хорошей»? С точки зрения, как конечного пользователя, так и программиста-профессионала такая программа должна обладать следующими свойствами:

- надежность;
- ясность для понимания;
- удобство сопровождения и модификации.

Для создания хорошей программы необходимо иметь представление о современных технологиях программирования, хорошо знать и уметь использовать основные принципы программирования.

В ходе развития программирования были сформулированы принципы, которые являются базовыми для любых технологий современного программирования:

- структурность программного кода;
- модульность;
- развитая типизация и структурность данных.

С содержанием этих принципов можно познакомиться в лабораторном практикуме. Использование принципов обязательно при создании современных качественных программных продуктов.

Структурность программного кода

Принцип структурности программы состоит в том, что последовательность выполнения фрагментов программы полностью совпадает с последовательностью их расположения.

Отдельные фрагменты, из которых строится программа, содержат условия и порядок обработки данных. Эти фрагменты называют *управляющими структурами*. В теории программирования показано, что программа любой сложности может быть построена комбинацией всего трех базовых управляющих структур:

- следование;
- развилка (ветвление);
- повторение (цикл «пока»).

Запись основных управляющих структур на псевдокоде и реализация их на языке Pascal приведены в табл. 1.

Здесь условие − это утверждение, о котором всегда можно сказать, истинно оно или ложно в данный момент.

В структуре *Следование* действия последовательно выполняются в порядке их записи.

Таблица 1

Основные управляющие структуры

Структура	**Реализация на Паскале**
Следование	
<действие1>; <действие2>; <действие3>; ...	<оператор1>; <оператор2>; <оператор3>; ...
Развилка	

если (условие) **то**
  <действие1>
иначе
  <действие2>
	if (условие) **then**
	  <оператор1>
	else
	  <оператор2>

Повторение	
пока (условие) **повторять** <действие>	**while** (условие) **do** <оператор>

В структуре *Развилка* при истинности условия выполняется действие1, иначе действие2.

В структуре *Повторение* каждый шаг повторения начинается с проверки условия. Повторение завершается, когда условие ложно.

СТРУКТУРНАЯ ЗАПИСЬ ПРОГРАММНОГО КОДА (см. более подробно в приложении «Стандарт стилевого оформления исходного кода DELPHI»). Хорошим считают стиль оформления программы, облег-

чающий ее восприятие как самим автором, так и другими программистами, которым, возможно, придется ее проверять или модифицировать.

Стиль оформления программы включает:

- правила именования объектов программы (переменных, функций, типов, данных и т. п.);
- стиль оформления текстов модулей.

Правила именования объектов программы

При выборе имен программных объектов следует придерживаться следующих правил:

- имя объекта должно соответствовать его содержанию, например:

```
MaxItem – максимальный элемент;
NextItem – следующий элемент;
```

- если позволяет язык программирования, можно использовать символ «_» для визуального разделения имен, состоящих из нескольких слов, например:

```
Max_Item, Next_Itetm;
```

- необходимо избегать близких по написанию имен, например:

```
Index и InDec.
```

Стиль оформления текстов модулей

При написании программного кода его структура подчеркивается использованием отступов, пустых строк и комментариев. Отступ вправо на две позиции показывает внутреннюю часть (тело) управляющей структуры и вложенность одних структур в другие. Отступ влево означает возврат на предыдущий уровень вложенности. Пустыми строками выделяют отдельные важные фрагменты программы.

Комментировать следует цели выполнения тех или иных действий, а также группы операторов, связанные общим действием, т. е. комментарии должны содержать некоторую дополнительную (неочевидную) информацию.

Пример:

```
.............................
// Процедура регистрации типа метаданных в реестре
procedure GsvRegisterTypesInfo(aClasses:
 array of TGsvObjectInspectorTypeInfoClass);
var
 i: Integer;
begin
 for i := Low(AClasses) to High(AClasses) do
 GsvRegisterTypeInfo(AClasses[i]);
end;
```

```
// Функция поиска типа метаданных в реестре
function GsvFindTypeInfo(const ATypeName: String):
 TGsvObjectInspectorTypeInfoClass;
var
 i: Integer;
begin
 Result := nil;
 if Assigned(GsvTypesInfo) then
  if GsvTypesInfo.Find(ATypeName + '_INFO', i) then
  Result:=TGsvObjectInspectorTypeInfoClass
            (GsvTypesInfo.Objects[i]);
end;
```

Модульность

Этот принцип заключается в том, что программа строится из отдельных законченных частей – модулей, включающих в себя различные описания данных и подпрограммы. Подпрограмма – это часть общего кода программы, оформленная по специальным правилам, которая выполняет определенную подзадачу основной задачи. Существуют две разновидности подпрограмм – процедуры и функции.

Один из модулей – основная программа (главный модуль), остальные – модули с процедурами, функциями или объектами, которые используются в основной программе или в других модулях. Любые модули, кроме основной программы, могут отсутствовать. Каждый модуль должен иметь название (имя модуля), описание данных, которые им обрабатываются, и описание реализуемых им алгоритмов. Такие описания помещаются в определенном месте программы. В том месте, где нужно использовать модуль, например, в другом модуле, указывается только его имя.

Типизация и структурность данных

Современные языки программирования имеют развитую систему типов данных, основное назначение которой − выразить решение задачи с наибольшей ясностью и надежностью. *Типизация* − это способ разграничения данных по их смысловому назначению. *Структурность* − это возможность конструировать из простых данных более сложные.

Чтобы понять, почему типизация данных важна для обеспечения надежности, рассмотрим два класса ошибок, часто встречающихся в программах:

- присваивание данным неверных по смыслу значений
- использование неверных по смыслу операций над данными

Язык программирования должен иметь средства обнаружения таких ошибок на этапе трансляции или во время работы программы. Они обеспечиваются системой типов данных. В этой системе каждый тип определяет множество допустимых значений данных, относящихся к этому типу, а также набор операций, разрешенных для таких данных. Таким образом, компилятор имеет возможность обнаружить попытку присваивания не относящихся к данному типу значений, а также попытку выполнить недопустимую операцию над данными. Система типов данных в современных языках включает простые и сложные (структурные) типы. Данные простых типов не имеют внутренней структуры. Их значением является единственное значение из допустимого множества. Сложные данные конструируются из простых, значением их является совокупность значений их составляющих. Использование структурных данных позволяет более удобно и эффективно описывать алгоритм решения задачи. А иногда и вообще применить алгоритм, который при неструктурном представлении данных использовать невозможно.

ГЛАВА 6. СРЕДА ПРОГРАММИРОВАНИЯ DELPHI

Требования максимального упрощения и ускорения процесса разработки приложений и использования ранее реализованных программных фрагментов. привели к созданию многочисленных RAD-систем (Rapid Application Development − быстрая разработка приложений).

Одной из таких RAD-систем является Delphi [2, 6−10]. Среда Delphi − это объектно-ориентированная среда для визуального проектирования Windows-приложений, основанная на языке Object Pascal.

Общая характеристика Delphi

В основе среды Delphi лежит компонентная модель разработки программных продуктов. Суть модели заключается в поддержке системой широкого набора компонентов, из которых и строится программа. Компоненты − это объекты, с которыми можно работать визуально, и для этого у них есть необходимые свойства и методы.

Компонентная модель разработки дает возможность многократного повторного использования программного кода.

За счет использования готовых визуальных компонентов интерфейса теперь один программист получил возможность выполнять работу группы программистов за более короткий срок. Фактически, пользовательский интерфейс создаваемого приложения в среде Delphi программист собирает прямо на экране из предлагаемого набора готовых элементов. Текст программы, соответствующий этому интерфейсу, Delphi создает автоматически. Таким образом, для создания в Delphi несложных программных продуктов совершенно не обязательно понимать внутреннюю структуру Windows-приложения, получаемого после разработки в Delphi. Достаточно просто уметь работать с некоторыми компонентами, поставляемыми вместе со средой разработчика. При этом начать работу со средой можно практически без предварительного ознакомления, а написание первого приложения не потребует углубления в особенности системы. Этому отчасти способствует удобный интерфейс среды разработчика, не перегруженный излишними вопросами к разработчику.

Однако такой подход совершенно неприемлем для серьезного программирования, и, рано или поздно, придется освоить и основы программирования под ОС Windows, и серьезно изучить саму среду разработки Delphi, а также возможности, которые она предоставляет.

Состав среды Delphi

Среда состоит из четырех основных окон, появляющихся после запуска Delphi (рис. 99):

- главное окно
- окно **Object Inspector (инспектор объектов)**
- окно **Конструктора форм**
- окно **Редактора кода**

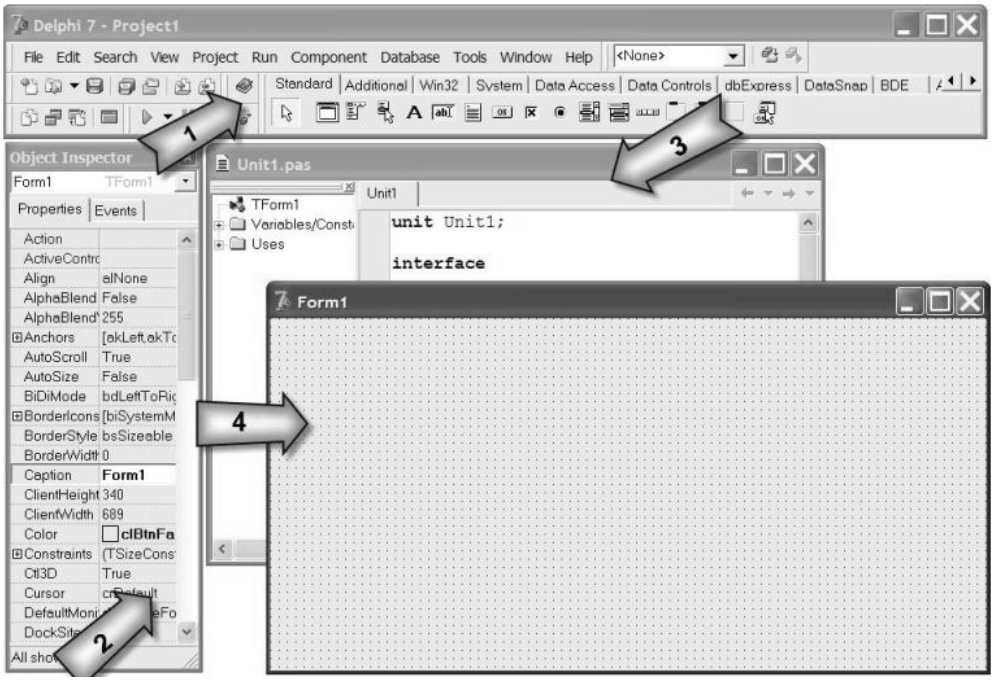


Рис. 99. Окна среды визуального программирования Delphi:
1) главное окно; 2) окно инспектора объектов; 3) окно редактора кода;
4) окно конструктора форм

Главное окно (1) является управляющим центром среды. В него входят три отдельных элемента: Панель меню, Панель инструментов и Палитра компонентов.

Панель меню обеспечивает управление всеми окнами среды, и при переключении от окна к окну она не меняется. Меню позволяет настраивать интерфейс окон Delphi, создавать нужную конфигурацию среды, получать контекстную подсказку.

Панель инструментов обеспечивает быстрый доступ ко всем основным операциям, представленным на Панели меню (операции с файлами, редактирование текста, запуск и отладка создаваемых программ из Delphi, выбор форм и модулей). Доступ обеспечивает набором кнопок с пиктограммами и всплывающими подсказками по назначению кнопки. Панель настраивается, и пользователь всегда может включить в нее необходимые или исключить ненужные ему кнопки.

Палитра компонентов содержит все визуальные и невизуальные компоненты, которые можно добавить к разрабатываемому приложению. Компоненты в Delphi разделены на группы (страницы) и позволяют Delphi использовать большинство присущих Windows возможностей. Желаемая группа компонентов выбирается нажатием соответствующей закладки в палитре. Чтобы поместить требуемый компонент на форму, достаточно щелкнуть на его значке в палитре и затем еще раз щелкнуть в нужном месте формы. Сама палитра может быть дополнена новыми компонентами из числа существующих, либо сокращена за счет удаления ненужных компонентов. Возможно также перемещение компонентов с одной страницы на другую или же изменение порядка компонентов в пределах одной страницы.

Object Inspector (2) предназначен для работы с объектами (размещенными на форме компонентами) и включает в себя:

- **комбинированный список**(Combo box), который отображает название и тип текущего объекта и позволяет выбрать другой объект;
- **страницу Свойства** (Properties), которая отображает два списка – названия свойств, доступных для текущего объекта, и значения этих свойств;
- **страницу События** (Events), которая содержит названия событий, доступных для текущего (выбранного) объекта, и строки для редактирования имени обработчика событий.

Окно Редактора кода (3) позволяет редактировать текст модулей (отдельных частей программы), написанных на языке Object Pascal, и предоставляет для этого широкий набор средств. Оно оформлено в виде многостраничного блокнота и позволяет переключаться между всеми открытыми файлами проекта с текстами модулей. По умолчанию вновь создаваемый модуль получает имя UnitN, где N – его порядковый номер, и отображается на новой странице Редактора кода. В дальнейшем это имя может быть изменено в соответствии с пожеланиями разработчика. Если модуль связан с формой, то в нем сразу автоматически создается описание всех элементов интерфейса, включенных и включаемых в эту форму. Модуль, соответствующий нужной форме, отображается в редакторе кода либо после двойного щелчка кнопкой мыши на поле формы, либо после выполнения пунктов меню View/Toggle Form/Unit. Всякие изменения в форме автоматически отображаются в тексте модуля в окне редактора.

Окно Конструктора форм (4) расположено в правой части экрана над окном редактора кода. Оно обеспечивает возможность создания интерфейса пользователя для разрабатываемого приложения и тем самым определяет его вид. На форме размещаются все визуальные и невизуальные компоненты разрабатываемого приложения. По умолчанию для ново-

го проекта это окно имеет заголовок «Form1». В дальнейшем это имя может быть изменено в соответствии с пожеланиями разработчика. Окно форм фактически является пустой заготовкой окна приложения, которую можно наполнить нужными элементами интерфейса. Для этого используется палитра компонентов из главного окна. Любые изменения, внесенные в форму при разработке, сохраняются и при работе приложения. Приложение может содержать несколько форм, каждая из которых запускается в определенное время в зависимости от потребностей пользователя.

Свойства, события и методы компонентов Delphi

В Delphi для создания программ используется объектно-ориентированная технология. Структурной единицей визуального программирования является *компонент*.

Компонент − это разновидность объекта, который:

- можно перенести в приложение из специальной Палитры компонентов;
- для обеспечения возможности реагировать на внешние события имеет набор свойств, которые можно определять, не изменяя исходный код программы.

Любой компонент содержит *методы-обработчики событий*. Эти обработчики содержат описание реакции компонента на события, связанные с ним. Событием может быть действие пользователя или же действия, связанные с работой операционной системы, например, нажатие клавиши мыши или запуск системного таймера

После добавления компонента в создаваемое приложение нужно задать его состояние и поведение. Делается это с помощью набора свойств, методов и событий, определённых и описанных в библиотеке визуальных компонентов. Такое описание компонента называется *классом*.

Свойства описывают состояние компонентов. Каждое свойство имеет свое значение, которое можно считывать или устанавливать. Для описания поведения используются методы и события.

Свойства

Свойства определяют состояние (например, расположение и внешний вид) объекта. Чтобы узнать или изменить свойство на этапе проектирования, можно использовать окно *Object Inspector*. В этом окне перечисляются только те свойства, которые доступны на этапе проектирования, свойства этапа выполнения отсутствуют. На рис. 100 приводится вид окна Object Inspector, в котором отображен список свойств для компонента Button (Кнопка), размещённого на форме в ходе проектирования интерфейса приложения.

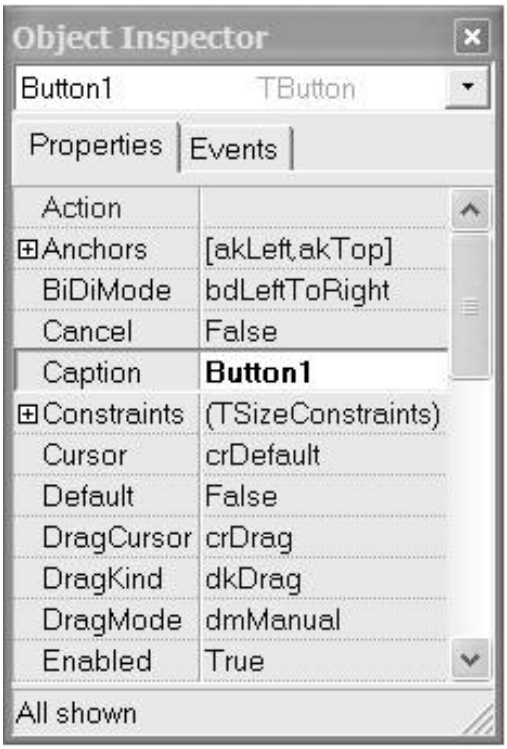


Рис. 100. Окно Object Inspecto r со списком свойств

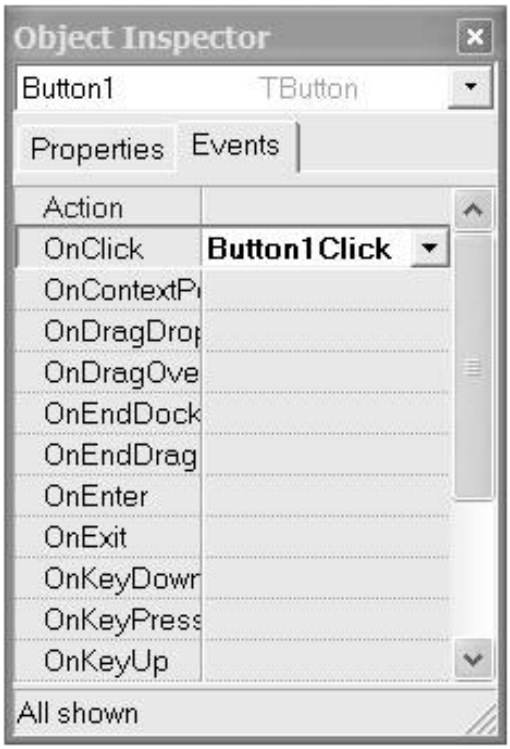


Рис. 101. Окно Object Inspector со списком событий

События

Когда пользователь или операционная система воздействует на компонент (допустим, пользователь щелкает на нем мышкой), тот генерирует *событие*. Чтобы программа управлялась этим событием, необходимо предусмотреть его обработку в программе, т. е. написать код, реализующий ответную реакцию программы. Этот код называется *обработчиком события,* и он оформляется специальным образом в виде специальной программной структуры – процедуры. Для создания пустой заготовки такой процедуры используется страница Events в окне Object Inspector. На рис. 101 приводится вид окна Object Inspector, в котором отображён список событий для компонента Button (Кнопка), размещённого на форме в ходе проектирования интерфейса приложения. Страница соответствует созданию процедуры-обработчика для события *OnClick* (Щелчок).

Методы

Методы компонентов − это самостоятельные программные единицы (подпрограммы: процедуры и функции), связанные с объектом, которые можно вызвать для выполнения соответствующих действий. Например, объект может иметь процедуру для вывода какого-то текста на экран. Эта процедура и есть метод, который принадлежит объекту.

Технология программирования в Delphi

Программирование в Delphi состоит из двух основных этапов:

1. Визуальное построение программы на основе компонентов и настройка их свойств, в результате чего можно быстро сформировать пользовательский интерфейс и обеспечить все его функции.

2. Написание программного кода на языке Object Pascal и его отладка для обеспечения тех функций приложения, которых невозможно достичь использованием визуального построения. На этом этапе выполняется компиляция, компоновка и выполнение программного кода с целью проверки правильности его работы. *Компиляция* позволяет получить из исходного текста программу в машинных кодах, которую может выполнить компьютер. *Компоновка* объединяет в единое целое откомпилированные модули программы. В среде Delphi компиляция и компоновка не разделяются и выполняются друг за другом. В результате компиляции и компоновки получается обычный исполняемый файл с расширением .exe.

Важным для Delphi является понятие проекта. *Проект* – это совокупность файлов, содержащих программный код и разного рода служебную информацию для создаваемого приложения.

Проект предназначен для повышения удобства разработки программных продуктов, особенно крупных.

Программа в Delphi состоит из совокупности модулей. Модулем в Delphi называют автономно компилируемую программную единицу.

Один из модулей − главный, создается системой Delphi автоматически. Его задача − запуск созданного приложения. Другие модули содержат описание требуемых для решения задачи данных, объектов, функций и процедур, выполняемых по определённым правилам.

Модуль хранится в отдельном файле. Заголовок модуля должен совпадать с именем файла, в котором модуль хранится.

Формирование проекта из составных частей

Как уже говорилось выше, проект с точки зрения программиста состоит из программы-проекта и ряда модулей различного назначения. В среде Delphi можно создавать новые проекты и корректировать уже имеющиеся, причем многие изменения визуально отображаются на экране монитора, что позволяет сразу же проверить правильность выполняемых действий.

Создание нового проекта

Как правило, проект должен содержать программу-проект как его основу и хотя бы один модуль формы. Кроме этого, обычно в проекте используются различные стандартные модули. Большинство стандарт-

ных подпрограмм находится в стандартном модуле System, который подключается к проекту автоматически. Другие стандартные модули следует подключать к проекту по мере необходимости в разделе uses. Многие стандартные модули, которые необходимо использовать практически в любой программе, подключаются автоматически средствами Delphi (например, при создании нового проекта в программе-проекте подключается модуль Forms, содержащий необходимые средства создания форм и работы с ними).

Для создания нового проекта можно воспользоваться следующими средствами:

- выбрать команду главного меню **File/New Application**;
- выполнить с помощью окна выбора элементов проекта операцию **File|New|New Application**; .

В каждом из этих случаев создается новый проект, состоящий из программы-проекта, имеющей имя Projectl, и заготовки модуля формы, имеющего имя Unitl. Форма модуля Unitl становится главной формой, и ее закрытие во время работы программы прекращает работу этой программы. При этом открытый проект закрывается, а если в него были внесены изменения, задается вопрос, нужно ли сохранить эти изменения. *При сохранении проекта рекомендуется имена Projectl и Unitl заменить на осмысленные имена.*

Новый проект может открываться также при запуске Delphi.

Открытие существующего проекта

Для того чтобы открыть существующий проект, можно воспользоваться следующими возможностями:

- использовать команду главного меню **File/Open**;
- на панели инструментов нажать кнопку *Open*.

В этих случаях выводится окно выбора имени файла, в котором нужно выбрать файл, содержащий требуемую программу-проект (такой файл стандартно имеет расширение **.dpr**). Открытый проект при этом закрывается, а если в его файлах были произведены изменения, выдается запрос, следует ли сохранить эти изменения.

Если проект, который следует открыть, был недавно закрыт, можно воспользоваться командой главного меню **File/Reopen**, позволяющей открыть один из нескольких последних закрытых проектов. Аналогичные действия можно выполнить с кнопкой *Open*, а точнее, со стрелкой, направленной вниз, находящейся в правой части этой кнопки.

Добавление к проекту нового файла

К проекту можно добавлять файлы, содержащие различные модули, включая модули формы, данные и т. д.

Наиболее полные возможности здесь при использовании команды главного меню **File/New**, которая открывает окно выбора элементов проекта.

Модуль, содержащий форму, можно подключить, используя команду главного меню **File/New Form** или кнопку панели инструментов **New Form**. При добавлении новой формы создается заготовка ее текста, изображение пустой формы, а сама она подключается к проекту в программе-проекте.

Добавление к проекту существующего файла

Для добавления к проекту существующего файла с тем или иным модулем можно воспользоваться следующими возможностями:

- использовать команду главного меню **Project / Add to Project**;
- нажать кнопку **Add file to project** панели инструментов

Во всех этих случаях открывается окно выбора имени файла для задания имени подключаемого файла (модули находятся в файлах с расширением **.pas**). После выбора имени файла его текст появляется в текстовом редакторе на новой странице, открывается его форма (если у модуля она имеется) и устанавливаются связи с этим модулем в программе-проекте.

Удаление файла из проекта

Для удаления файла из проекта можно использовать следующие возможности:

- использовать команду главного **меню Project/Remove from Project**;
- нажать кнопку **Remove file from project** панели инструментов;

Во всех этих случаях открывается окно выбора имени удаляемого модуля. Перед удалением файла из проекта задается вопрос, следует ли сохранить изменения, если они в нем были произведены. Для выбранного модуля с экрана удаляется его текст и форма, если таковая у него имеется. Удаляются связи с модулем в программе-проекте. Сам файл на диске не уничтожается, если он был туда записан. Если удаляется модуль главной формы, главной формой становится следующая подходящая форма в списке используемых в программе-проекте модулей (в разделе uses).

Замечание. Надо иметь в виду, что другие способы удаления модуля из проекта (например, просто удаление соответствующей строки в тексте программы-проекта) использовать не следует, так как это может привести к ошибке при компиляции.

Открытие файла

Можно открыть файл, не включая его в проект (например, для того, чтобы скопировать его какую-либо часть в файлы проекта, или для сравнения его с файлами проекта), или открыть файл, уже включенный в про-

ект, но по каким-либо причинам закрытый. Файл может содержать программу-проект, модуль формы, модуль или текст. Открыть файл можно:

- с помощью команды главного меню **File / 0pen**;
- нажатием кнопки *Open* панели инструментов.

В этих случаях открывается окно выбора имени файла, с помощью которого выбирается имя открываемого файла.

Если требуется открыть файл, включенный в проект, можно воспользоваться кнопками панели инструментов просмотра текста файла (**View Unit**) или просмотра формы (**View Form**), если она у модуля есть. Предварительно следует выделить мышью или с помощью клавиатуры соответствующую строку в списке модулей проекта. При этом на экране открывается как текст модуля, так и его форма. Эту же операцию можно выполнить либо активизировав мышью имя модуля или имя формы модуля в списке модулей проекта, либо активизировав клавиатурой соответствующую строку в этом списке. В любом случае на экране появляются и текст модуля, и форма.

Если открываемый файл содержит программу-проект (расширение файла **.dpr**), текущий проект закрывается (предварительно будет выведен запрос, следует ли сохранить внесенные в закрываемый проект изменения).

Если будет сделана попытка открыть несуществующий файл, появится сообщение, что такого файла не существует, и вопрос, следует ли открыть новый файл с таким именем и тем или иным типом (форму, модуль, текстовый файл).

Надо иметь в виду, что открытый файл к проекту не присоединяется, если он ранее не был включен в проект. Для подключения его к проекту следует воспользоваться средствами, описанными в разделе *Добавление к проекту существующего файла*.

Открытие программы-проекта

При открытии любого файла программа-проект на экран не выводится. Для того чтобы просмотреть ее текст, следует воспользоваться командой главного меню **View / Project Source**. Текст программы-проекта появляется на новой странице текстового редактора.

Задание связей между файлами

В проекте одни модули могут использовать другие модули проекта. Установить эти связи можно с помощью команды главного меню File|Use Unit. При этом предварительно надо активизировать модуль (текст или форму), в котором следует задать эту связь. При выполнении команды создается окно выбора имени открытого модуля, из которого выбирается имя, нужное для задания в активном модуле. Это имя помещается в раздел uses исполнительной части модуля.

При создании проекта первая созданная форма становится главной (ее закрытие приводит к прекращению работы запущенной программы). Чтобы переназначить главную форму, следует выполнить команду главного меню **Project / Options**, с помощью которой открывается окно задания параметров проекта, на странице *Forms* которого в комбинированной строке ввода *Main form* (главная форма) необходимо задать имя главной формы. Для задания имени формы нужно воспользоваться выпадающим списком строки ввода, в котором перечислены все формы проекта, способные выполнять функции главной формы.

Сохранение проекта

Сохранить проект можно следующими способами:

- с помощью команды главного меню **File/Save All**;
- с помощью кнопки *Save All* панели инструментов',

Во всех случаях сохраняются файлы проекта, в которых произошли изменения (в тексте или форме). В первых двух случаях сохраняются также измененные файлы, не входящие в проект.

Если какой-либо файл ни разу не сохранялся, выводится окно выбора имени файла для задания ему имени. В этом смысле первоначальные имена файлов (Projectl, Unitl, Unit2 и т. д.), задаваемые средствами Delphi, являются временными, и такие файлы целесообразно переименовать. Сам модуль также приобретает новое имя. При переименовании файла автоматически изменяются соответствующие связи в программе-проекте (и только в ней).

Можно сохранить проект и под другим именем. Это может потребоваться, если создается новый вариант программы. Для этого следует выбрать команду главного меню **File/Save Project As**. По этой команде выводится окно выбора имени файла, в котором надо задать новое имя. Если файл с таким именем уже имеется, будет выдано предупреждение об этом. Переименовывается только программа-проект (если только в проекте нет файлов, ни разу не сохранявшихся).

Сохранение файла

Чтобы сохранить открытый модуль в файле, можно воспользоваться командой главного меню **File / Save** или кнопкой *Save* панели инструментов. Предварительно следует сделать активным текст сохраняемого модуля или его форму, если она имеется.

Если сохраняемый файл еще ни разу не сохранялся, выводится окно выбора имени файла для задания ему вместо временного имени постоянного.

Тот или иной файл можно сохранить и под другим именем. Для этого достаточно воспользоваться командой главного меню **File/Save As**. При выполнении этой команды выводится окно выбора имени файла для задания нового имени. После выбора нового имени изменится на новое также имя модуля, а в программе-проекте также на новую изменится связь со старым файлом.

Если при сохранении файла с использованием этой команды уже имеется файл с заданным именем, появляется вопрос, заменить ли уже имеющийся файл. При утвердительном ответе старый файл заменяется на новый, при отрицательном ответе команда **Save A**s не выполняется.

Если сохраняется программа-проект, то также сохраняются и все измененные модули проекта.

Закрытие проекта

Закрыть проект можно следующими способами:

- использовать команду главного меню **File/Close All**;
- открыть другой (может быть, новый) проект;
- закрыть программу-проект.

Во всех случаях закрываются все открытые файлы (в том числе и не входящие в проект).

Закрытие файла

Для того чтобы закрыть файл, можно воспользоваться командой главного меню **FiIe/Close**. Предварительно следует активизировать тот модуль (или программу-проект), который нужно закрыть, сделав активным либо его текст, либо форму, если она имеется у модуля. При выполнении команды соответствующий модуль закрывается, при этом удаляются с экрана его текст и форма, если она имеется. Для измененного модуля предварительно выдается сообщение, следует ли сохранить его изменения. Сам модуль из проекта не удаляется, и его можно восстановить средствами, описанными в разделе *Открытие файла*. Для удаления модуля из проекта достаточно воспользоваться средствами, описанными в разделе *Удаление файла из проекта*.

Если закрывается программа-проект, то закрываются и все его модули.

Структура программных модулей

Структура главного модуля (основной программы) и любого другого модуля отличаются между собой. Описания различных программных объектов приводятся в соответствующих разделах программы и программных модулей.

Структура главного модуля

```
program <имя программы>;
uses
   <Подключаемые модули >
begin
   < Раздел операторов >
end.
```

Структура модуля

```
unit <ИмяМодуля>;          //Заголовок модуля
Interface                  //Интерфейсная часть
uses
    <Подключаемые модули >
Implementation             //Реализационная часть
uses
    <Подключаемые модули >
//Основной блок модуля
begin
    < Раздел операторов >
end.
```

Заголовок модуля должен совпадать с именем файла, в котором модуль хранится.

В интерфейсной части описываются все константы, типы данных, переменные, доступные для использования в других модулях или нужные для описываемых в этом модуле процедур и функций. Приводятся только заголовки процедур и функций.

В реализационной части описываются типы данных и структуры данных, доступные только в этом модуле. Здесь же приводится полное описание процедур и функций всех частей модуля.

В инициализационной части модуля помещаются те операторы, которые нужно выполнять перед операторами основного блока того модуля, в который включен данный модуль. С их помощью задаются (инициализируются) начальные значения необходимых данных.

Любая из этих частей может быть пустой, если в ней нет надобности.

ГЛАВА 7. ЯЗЫКИ ПРОГРАММИРОВАНИЯ PASCAL И OBJECT PASCAL

Понятия языка программирования

Всем современным языкам программирования присущи некоторые общие понятия, такие как:

- алфавит языка программирования
- комментарии
- идентификатор (имя)
- переменные, константы, выражения.

Далее приводится реализация этих понятий в языке Pascal и его расширении Object Pascal, лежащем в основе Delphi.

Алфавит языка Pascal

Алфавит языка Pascal включает в себя буквы латинского алфавита, цифры, специальные символы и служебные слова.

Комментарий

Комментарии используются для пояснения фрагментов программы. Комментарий − это последовательность любых символов, заключенная между фигурными скобками.

```
{ Это комментарий }
{ А это пример многострочного
 комментария
}
//комментарий в стиле языка C++, его можно исполь-
зовать в //конце строки.
```

Идентификаторы

Идентификатор или имя служит для обозначения программных объектов (констант, типов, переменных, процедур, функций). Имя является ссылкой на используемый в программе объект. Имя состоит из букв, цифр и может включать символ подчеркивания "_". Имя не может начинаться с цифры. Имена не должны совпадать со служебными словами.

Переменные, константы, выражения

Переменная − это именованный программный объект, который может изменять свое значение в ходе выполнения программы. Имя переменной является носителем ее значения.

Константа − это программный объект, который не изменяет своего значения в ходе выполнения программы. В языке Pascal допускаются неименованные и именованные константы, т. е. константа в программе может обозначаться именем, которому ставится в соответствие неизменяемое значение.

Выражение − описание действий, задающее правила вычисления некоторого значения. При этом такое описание ничего не говорит, что делать с этим значением (использование значения определяется другим типом описания действий − операторами). Общий вид выражения:

<операнд1> <знак операции> <операнд2>,

где операнд может быть константой, переменной, вызовом функции или, в свою очередь, выражением. Вызов функции – это обращение к особому виду подпрограммы, которая вычисляет значение.

Чтобы переменной придать значение, используют специальный оператор присваивания, который имеет вид:

```
<имя_переменной> := <выражение>.
```

Пример:

```
FullFileName := FileName + '.doc';
```

Средства описания данных

В современном программировании применяется *принцип: все программные объекты* (такие как константы, типы, переменные и др.) *должны быть описаны до их первого использования*. Это описание дает компилятору информацию о *свойствах* программных объектов, позволяя осуществлять контроль типов данных, о котором мы говорили при обсуждении принципа типизации и структурности данных в разделе «Основные принципы программирования». Описания различных программных объектов приводятся в соответствующих разделах программы и программных модулей.

Каждый раздел описаний начинается специальным служебным словом:

```
const          // начало раздела описаний констант
type           // начало раздела описаний типов
var            // начало раздела описаний переменных
```

Любой раздел описаний может отсутствовать, если в программе не используются соответствующие программные объекты.

Система типов данных в современных языках включает простые и сложные (структурные) типы. Данные простых типов не имеют внутренней структуры. Их значением является единственное значение из допустимого множества.

Простые типы данных и реализация их в Паскале

Мы будем рассматривать программирование в среде визуального программирования Delphi. Языком программирования в ней является Object Pascal – расширение классического языка Pascal.

Совокупность простых типов данных включает в себя стандартные (предопределенные) типы и определяемые программистом. Под

стандартными понимаются типы данных, реализованные практически во всех современных языках программирования. Для их использования достаточно указать имя типа, все остальное о них компилятор «знает». К ним относятся числовые типы − *целые* и *вещественные* (*действительные*) числа, которые являются переносом в программирование естественных математических понятий. Помимо числовых типов большинство современных языков программирования содержат еще два стандартных типа данных − *литерный* и *логический*.

Ниже приводится сводная таблица (табл. 2) основных сведений о простых стандартных типах данных в языке Pascal.

Таблица 2

Простые стандартные типы данных

Тип данных	**Целый**	**Вещественный**	**Литерный (символьный)**	**Логический**
Диапазон значений	множество целых чисел в диапазоне, определяемом разрядностью компьютера	Ограниченное множество вещественных чисел в диапазоне, определяемом разрядностью компьютера	набор литер (символов), определенных в вычислительной системе для связи с внешним миром	ложь истина
Имя типа	**integer**	**real**	**char**	**boolean**
Допустимые операции	сложение, вычитание, умножение, деление нацело (**div**) и взятие остатка от деления (**mod)**	сложение, вычитание, умножение, деление (/). Результат деления всегда вещественный	операции отношения: >, >=, <, <=, =, <>("неравно")	операции отношения; логические операции: **not** -логическое отрицание **and** – логическое И **or** − логическое ИЛИ
Константы	Обычная математическая запись целых чисел	С фиксированной точкой (обычная математическая запись, но дробная часть отделяется от целой точкой); с плавающей точкой	литера, заключенная в апострофы ‘L’ ‘9’ ‘+’ ‘?’	False True

Замечание: В настоящее время для описания вещественных чисел в Delphi, наряду с типом real, используется эквивалентный ему тип double.

Описания переменных стандартных типов приводятся в разделах описания переменных программы и программных модулей по следующему правилу:

```
var
  ИмяПеременной1, ИмяПеременной2, … ,
  ИмяПеременнойN: ИмяТипа;
```

Примеры описаний:

```
var
  nomer_kvartiry, den_nedely : integer;
  dlina, massa: real;
  Symvol, Ltera : char;
  EtoSymvol, eto_cifra : boolean;
```

Типы, определяемые программистом

Наряду со стандартными, в языках программирования имеется возможность вводить в употребление в программе и новые, нестандартные простые типы данных, а также сложные типы. Для ввода в употребление любых новых типов в программе существует *раздел описания типов*, для указания которого в языке используют специальное слово "**type**", а описание новых типов дается в следующей форме:

<имя типа> = <определение типа>

Далее <имя типа> используется для описания переменных этого типа.

В качестве нестандартных простых типов в языке Pascal определены два типа данных – *перечислимый* тип и *ограниченный* тип. В связи с большей распространенностью ограниченного типа данных рассмотрим именно его.

Ограниченный (диапазонный) тип

Ограниченный тип задается на основе любого простого упорядоченного типа указанием границ выбранного диапазона, разделенных двумя точками. Тип данных, лежащий в основе ограниченного, называется *базовым*.

```
type
  index= 1..20; // ограниченный тип на базе типа целый
var
  Nomer: index;
  Diapazon: 0..200;
```

Другие примеры описаний переменных простых типов:

```
var
   NomerElementa, Summa: integer;
   Simvol, Cifra : char;
   IsSimbol : Boolean;
   Height : real;
```

Структурные типы данных

Данные, обрабатываемые программой, являются некоторой моделью объектов реального мира. Обеспечить представление сложных объектов в программе связанной совокупностью отдельных данных позволяют *структурные (сложные)* типы данных. Такие новые типы данных создают по некоторым правилам из уже имеющихся типов, и их значением является совокупность значений их составляющих. Таким образом, достигается возможность создания данных произвольной сложности.

Регулярные типы (массивы)

Необходимость в массиве возникает тогда, когда при решении задачи приходится иметь дело с некоторым заранее известным количеством данных одного и того же типа. Массив в целом обозначается одним (групповым) именем. Отдельные данные, составляющие массив, называются элементами, и обозначаются этим общим именем и указанием его номера в наборе (индекса). Тип элементов называется базовым типом.

В языке Pascal задание типа массива имеет вид:

```
array [<тип_индекса>] of <тип_элементов>
```

Тип индекса обязательно должен быть простым типом (чаще всего используется ограниченный целый тип), тип элементов может быть любым (простым или структурным).

Если тип элементов простой, то массив называется *одномерным*. Во многих задачах возможны ситуации, когда элементами массива являются, в свою очередь, массивы. В этом случае описание примет вид:

```
array [<тип_индекса1>, <тип_индекса2>] of <тип_элементов>
```

Пример:

```
 var
    Temperatura: array [1..24] of real;
```

Сложную переменную типа массива, представляющую полную совокупность их элементов, будем называть полной переменной. Ее значением является вся совокупность значений ее элементов. Поэтому можно говорить о значении массива и значениях элементов массива. Для доступа к элементу массива необходимо записать общее имя массива и индекс элемента в квадратных скобках. Например, Temperatura [12].

Над массивами как полными переменными определена единственная операция − операция присваивания, т. е. значение массива (всю совокупность значений его элементов) можно присвоить другому массиву того же типа.

Строковые типы

Строковый тип – такой тип данных, значениями которого является произвольная последовательность символов алфавита. Каждая переменная такого типа может быть представлена фиксированным количеством байтов или иметь произвольную длину.

Для представления строк в Object Pascal используются следующие типы:

- короткая строка ShortString или String[N], где N <=255;
- длинная строка String;
- нуль-терминальная строка Pchar.

Общим для этих типов является то, что каждая строка трактуется как одномерный массив символов, но количество символов может быть разным. Так длина строки типа **String**[N] может меняться от 0 до N символов (**ShortString** – от 0 до 255 символов), длина строк типа **String** и **Pchar** ограничена только имеющейся оперативной памятью.

Нуль-терминальные строки **Pchar** введены для совместимости с некоторыми функциями операционной системы Windows, которые могут понадобиться программисту. Эти строки представляют собой массивы символов, которые заканчиваются специальным служебным символом #0 (этот символ ставится автоматически). Индекс первого элемента такой строки обязательно должен быть равен нулю.

Замечание: строки типа **PChar** совместимы с массивами символов, описанными как **array** [1..N] **of char**. Иными словами, при обращении к процедурам и функциям, которые требуют аргументов типа **Pchar**, вместо них можно подставлять аргументы, описанные как массивы символов. Иногда это удобно.

Над строками в целом определена операция присваивания. Разрешается присваивать строковой переменной значение строки того же строкового типа. Кроме того, для строк типа **String** определена операция конкатенации (слияния строк), которая обозначается знаком '+'. Например:

```
Var
    FileName, FileNameFull : string;
.............
    FileName := 'Report.doc';
    FileNameFull := 'C:\Мои документы\' + FileName;
```

После выполнения этого оператора значение строки FileNameFull будет 'C:\Мои документы\Report.doc'.

Подробности и особенности работы со строками, а также стандартные функции и процедуры для работы с ними хорошо описаны в литературе по Object Pascal.

Комбинированные типы (записи)

Записи (комбинированные типы данных) − совокупность разнородных, в общем случае, данных. Такая совокупность является единым программным объектом и имеет единое имя. Записи используются для представления в программе сложных объектов реального мира, обладающих совокупностью разнородных характеристик. Элементы записи называются полями записи. Каждое поле содержит данные о какой-либо характеристике объекта.

В Паскале задание записи имеет вид:

```
record
   <имя_поля_1> : <тип_поля_1>;
   <имя_поля_2> : <тип_поля_2>;
   <имя_поля_3> : <тип_поля_3>;
 ..............................
   <имя_поля_N> : <тип_поля_N>
end;
```

Каждое поле имеет свое уникальное (в пределах записи) имя и произвольный тип, в том числе им может быть и тип записи (такие записи называются иерархическими). Разные записи могут иметь поля с одинаковыми именами.

Пример описания записи, содержащей анкетные сведения о человеке:

```
type
  person = record
    FAM: string;
    God: integer;
    Adress: record
      Street: string;
      NomerDoma: integer
  end
 end;
```

На основе введенного типа могут быть описаны переменные, которые можно использовать для обработки разнородных данных :

```
var
  Student1, Student2: person;
```

Описанная таким образом запись содержит три поля, одно из которых (адрес), в свою очередь, также является записью.

Над записями в целом определена единственная операция − операция присваивания, т. е. значение записи (всю совокупность значений его полей) можно присвоить другой записи того же типа.

Чтобы получить доступ к соответствующему полю переменной типа запись, нужно записать селектор записи, который имеет вид:

< имя_переменной >.<имя_поля>

Тогда справедливы операторы, использующие следующие обращения к полям записи:

```
Student1.God := 1988 ;
Student1.Adress.Street:= 'Кузнечный взвоз'
```

Инструкция with

Инструкция **with** позволяет использовать в тексте программы имена полей без указания имени переменной-записи. В общем виде инструкция **with** выглядит следующим образом:

```
with Имя do
begin
// операторы программы
end;
```

где:

- Имя – имя переменной-записи;
- **with** – зарезервированное слово языка Delphi, означающее, что далее, до слова **end**, при обращении к полям записи Имя, имя записи можно не указывать.

Например, если в программе объявлена запись

```
student:record // информация о студенте
  f_name: string[30]; // фамилия
  l_name: string[20]; // имя
  address: string[50]; // адрес
end;
```

и данные о студенте находятся в полях Edit1, Edit2 и Edit3 диалогового окна, то вместо инструкций

```
student.f_name := Editl.Text;
student.l_name := Edit2.Text;
student.address := Edit3.Text;
```

можно записать:

```
with student do
begin
  f_name := Edit1.Text;
  f_name := Edit2.Text;
  address := Edit3.Text;
end;
```

Средства описания действий

Вспомним, что любой язык программирования включает в себя *средства описания данных* и *средства описания действий*. Средства описания действий − та часть языков программирования, которая непосредственно используется для выражения алгоритмов. Средства описания действий включают в себя три типа описаний:

- выражения;
- операторы;
- подпрограммы.

Выражения

Выражение − описание действий, задающее правила вычисления некоторого значения. При этом такое описание ничего не говорит, что делать с этим значением (использование значения определяется другим типом описания действий − операторами). Общий вид выражения:

```
<операнд1> <знак операции> <операнд2>,
```

где операнд может быть константой, переменной, вызовом функции или, в свою очередь, выражением. Тип выражения определяется типом результата вычисления. При этом значение имеет и тип операндов и операция. Например:

```
5 + 3 - арифметическое выражение целого типа (результат 8)
5 > 3 - логическое выражение (результат ИСТИНА).
```

Порядок вычисления выражения задается приоритетом операций. Сначала выполняются операции с самым высоким приоритетом, затем − со следующим по порядку приоритетом и т. д. Если необходимо изменить порядок вычисления, расставляют круглые скобки, так как действия в скобках выполняются в первую очередь. В порядке убывания приоритетов операции делятся на несколько групп:

```
операция NOT (логическое отрицание);
операции "типа умножения"
     *, /, div, mod,
     AND (логическое умножение)
операции "типа сложения"
     +, -,
    OR (логическое сложение)
операции отношения (сравнения)
     >, >=, <, <=, =, <>
     in (проверка принадлежности множеству).
```

ВАЖНОЕ ПРАКТИЧЕСКОЕ ПРАВИЛО! Всегда проще расставить скобки для указания нужного порядка вычислений, чем сомневаться, правильно ли вы учли все приоритеты.

Операторы

Оператор − описание действий, изменяющих состояние программных объектов (например, значения переменных) или управляющих ходом выполнения программы. Операторы делятся на простые и сложные, которые включают в себя некоторую совокупность простых операторов.

Простые операторы:

- оператор присваивания
- оператор перехода
- пустой оператор
- оператор процедуры, оператор вызова функции

Сложные операторы:

- составной оператор
- выбирающие операторы
- операторы цикла.

А теперь кратко рассмотрим содержание каждого из перечисленных операторов.

Простые операторы

Оператор присваивания − это описание действий, приводящих к изменению значений переменных. Он имеет следующий вид:

```
<имя_переменной> := <выражение>,
```

Пустой оператор − это просто "ничего", пустое место. Он не имеет специального обозначения. Его удобно использовать в случаях, когда по смыслу программных конструкций нужен оператор, но не требуется выполнять никаких действий.

Оператор перехода используется для нарушения естественного порядка выполнения операторов, совпадающего с порядком их записи в программе. Поскольку его использование не соответствует принципам структурного программирования, мы ограничимся лишь этим упоминанием о нем.

Оператор процедуры, оператор вызова функции − оба эти оператора используются для активизации (выполнения) подпрограмм, описание которых в виде функции или процедуры находится в разделе. Эти операторы имеют одинаковый внешний вид

<имя_подпрограммы> (<список_фактических параметров>)

и различаются только правилами использования: оператор процедуры используется самостоятельно, как отдельный оператор, а оператор вызова функции может использоваться только как часть выражения.

Пример оператора присваивания, в правой части которого выражение представляет собой оператор вызова функции:

ChisloRazryadov := NumberOfDigits(Chislo);

Составной оператор используется в тех случаях, когда синтаксис языка требует наличия одного оператора, а по смыслу задачи требуется выполнить целую последовательность действий. Составной оператор − это последовательность любого количества любых операторов, которая начинается служебным словом **begin** и заканчивается словом **end**

```
begin
   <оператор_1>;
   <оператор_2>;
   <оператор_3>;
   ...
   <оператор_N>
end
```

Пример:

В цикле **while** после слова **do** по правилам синтаксиса должен быть один оператор

```
while i <= 10 do
begin
  factorial := factorial * i;
  i := i + 1;
end;
```

Выбирающие операторы

Выбирающие операторы предназначены для записи многовариантных алгоритмов, когда выбор того или иного варианта зависит от выполнения некоторых условий. Под этим общим названием объединяются разновидности УСЛОВНОГО оператора и оператор ВЫБОРА (иногда его называют оператором варианта).

Условный оператор имеет две формы: полная может быть записана так:

```
if (<условие>) then <оператор1>
else                <оператор2>;
```

здесь <условие> записывается в форме логического выражения, которое вычисляется в момент выполнения оператора. Если условие истинно, то выполняется оператор1, а иначе − оператор2. Нетрудно видеть, что этот оператор реализует уже упоминавшуюся базовую программную структуру "развилка". Если оператор2 − пустой, то получается сокращенная форма, которая и записывается короче:

```
if (<условие>) then <оператор1>
```

Когда условие ложное, то оператор просто пропускается.

Оператор выбора используется, когда нужно выбрать одну из нескольких альтернативных возможностей. Выбор осуществляется по совпадению некоторого вычисленного значения с одним из заданного набора значений. Оператор выбора можно записать в виде

```
case <выбирающее Выражение> of
  <список вариантов_1>: <оператор_1>;
  <список вариантов_2>: <оператор_2>;
  <список вариантов_3>: <оператор_3>;
  ..............................
  <список вариантов_N>: <оператор_N>;
  else
     <оператор>;
end
```

Варианты – это значения некоторого простого типа, за исключением вещественного. Разделителем в списке вариантов является запятая. Тип результата вычисления выбирающего выражения должен совпадать с типом вариантов. Работа оператора выбора начинается с вычисления выражения. Если результат вычисления совпадает с одним из какого-то списка вариантов, выполняется оператор, соответствующий этому списку вариантов. Если полученного значения нет в наборе вариантов, то выполняется оператор после слова **else**. После этого работа оператора выбора закончена.

Пример: следующий оператор выводит на экран название предварительно введенного символа-цифры, при выводе используется оператор консольного вывода **writeln**

```
case Litera of
  '0' : writeln('ноль');
  '1' : writeln('один');
  '2' : writeln('два');
  ............................
  '9' : writeln('девять');
 else
   writeln('введенный символ – не цифра');
end
```

Операторы цикла

Известно, что многие алгоритмы требуют многократно повторяющихся действий (их называют циклическими). Для компактного задания таких действий в программировании используются операторы цикла. В программировании определены несколько разновидностей операторов цикла: **циклы, управляемые по условию,** и **циклы с заданным количеством повторений**. Смысл циклов, управляемых по условию, заключается в ПОВТОРЕНИИ действий, ПОКА выполняется некоторое УСЛОВИЕ.

Первый из этих операторов, называется оператором ЦИКЛА С ПРЕДУСЛОВИЕМ (другое название − цикл "пока"). Каждый шаг повторения начинается с проверки условия. Повторение прекращается тогда, когда условие станет ложным. Здесь <условие> записывается в форме логического выражения, которое вычисляется и проверяется перед выполнением действий.

Оператор цикла с предусловием записывается следующим образом:

```
while <условие> do
  <оператор>
```

Пример:

```
var
  factorial, i : integer;
  . . . . . .
  i:= 1;
  factorial:= 1;


  // Вычисление 10!
  while i <= 10 do
  begin
     factorial := factorial * i;
     i := i + 1;
  end;
  . . . . . .
```

Второй оператор называется оператором ЦИКЛА С ПОСТУСЛОВИЕМ. В отличие от цикла с предусловием в этой разновидности цикла сначала выполняются действия, а после этого проверяется условие. Если условие является истинным, циклические действия прекращаются, а иначе делается следующий шаг цикла. Отсюда ясно, что бывают ситуации, *когда цикл с предусловием не выполниться ни разу, а цикл с постусловием всегда выполняется хотя бы один раз.*

Оператор цикла с постусловием имеет следующий вид:

```
repeat
  <оператор1>;
  ..........
  <операторN>
until <условие>
```

Пример:

```
var
  factorial, i : integer;
  . . . . . .
```

```
  i:= 1;
  factorial:= 1;

  // Вычисление 10!
  repeat
    factorial := factorial * i;
    i := i + 1;
  until i >= 10;
. . . . . .
```

Важное замечание: при программировании следует обращать внимание на то, чтобы каждый шаг повторения цикла **содержал действия, меняющие значение логического выражения, используемого в качестве условия, иначе цикл будет продолжаться бесконечно!**

Следующая разновидность – ЦИКЛ С ЗАДАННЫМ КОЛИЧЕСТВОМ ПОВТОРЕНИЙ (цикл "для"). Этот оператор в общем виде имеет такую форму:

для <параметра_цикла> **от** <значения1> **до** <значения2> **выполнять** <действия>

Здесь <параметр_цикла > – переменная любого простого типа (кроме вещественного), <значение1> и <значение2> – границы диапазона изменения значений параметра цикла того же типа. Действия выполняются только после проверки текущего значения параметра. Параметр цикла на каждом шаге цикла последовательно меняет свое значение, начиная со <значения1> и кончая <значением2>. Выполнение действий прекращается, когда параметр цикла принимает значение вне заданного диапазона. Действия повторяются столько раз, сколько значений параметра цикла находится в заданном диапазоне.

Оператор цикла с заданным количеством повторений в Паскале имеет две формы записи. Форма записи зависит от правила, по которому вычисляется очередное значение параметра цикла. Если это значение определяется как следующее в заданном диапазоне за только что использованным, то оператор записывается следующим образом:

```
for параметр_цикла := значение1 to значение2 do
  <оператор>
```

При этом нужно следить за тем, чтобы задаваемые границы диапазона изменения значений параметра удовлетворяли неравенству значение1 <= значение2. В противном случае цикл не выполнится ни разу. Если очередное значение параметра определяется как предшествующее в заданном диапазоне только что использованному, то оператор записывается с заменой служебного слова **to** на слово **downto**

```
for параметр_цикла := значение1 downto значение2 do
  <оператор>
```

В этом случае также нужно следить за тем, чтобы задаваемые границы диапазона изменения значений параметра удовлетворяли неравенству, но уже обратному, т. е.значение1 > значение2.

Пример:

```
var
  factorial, i: integer;
  . . . . . .
  factorial:= 1;


  // Вычисление 10!
 for i := 1 to 10 do
    factorial := factorial * i;
. . . . . .
```

Функции и процедуры

Вспомните, что процедурное программирование связано с выделением из основной задачи более простых, достаточно независимых подзадач и повторением этого процесса (при необходимости) для каждой подзадачи. Такие подзадачи чаще всего удобно оформлять в виде подпрограмм – своеобразных микропрограмм, описывающих алгоритм решения выделенных подзадач и оформленных по определенным правилам, зависящим от языка программирования. При этом, описание алгоритма подзадачи (т. е. самой подпрограммы) помещается в определенном месте программы и делается один раз, а использование подпрограммы (АКТИВАЦИЯ, ВЫЗОВ) производится во всех местах программы, где это необходимо.

Подпрограмма может иметь разделы описаний данных и раздел описания действий по обработке данных. В подпрограмме следует описывать только те данные, которые связаны с алгоритмом подзадачи и не имеют отношения ко всей задаче в целом (ЛОКАЛЬНЫЕ параметры). Такое, "скрытое" от основной программы и других подпрограмм описание данных, делает их независимыми и повышает надежность программы, защищая эти данные от случайного изменения другими частями программы. Указанное свойство "упрятывания" данных внутри подпрограммы называется *инкапсуляцией*. В подпрограмме могут использоваться и неописанные в ней данные. Тогда они должны быть описаны на "внешнем" уровне (например, в вызывающей программе). В отличие от локальных их называют ГЛОБАЛЬНЫМИ по отношению к подпрограмме. В подпрограмме глобальные параметры имеют смысл, соответствующий их внешнему описанию. Любая подпрограмма имеет заголовок. Заголовок содержит имя подпрограммы, по которому и возможен

ее вызов в нужном месте, и некоторые другие сведения, необходимые для ее работы, например, список передаваемых подпрограмме данных. Для чего нужно передавать подпрограмме список данных? Вспомним, что подпрограмма содержит описание алгоритма обработки некоторых данных, соответствующего выделенной подзадаче. Можно обойтись без передачи данных (и такие подпрограммы встречаются), но тогда подпрограмма всегда будет связана с обработкой одних и тех же данных, имена которых использованы в ней. А ведь подпрограмма часто используется многократно в разных местах программы для обработки разных данных по одному алгоритму. Как обеспечить возможность такого использования подпрограммы? Такую возможность обеспечивает механизм соответствия ФОРМАЛЬНЫХ и ФАКТИЧЕСКИХ параметров. На стадии описания подпрограмме передается список обрабатываемых данных с их характеристиками − список формальных параметров (например, в скобках после имени подпрограммы). Имена формальных параметров используются внутри подпрограммы в описании алгоритма их обработки. Теперь подпрограмма "знает", как обрабатывать данные, представленные формальными именами. Остается сделать последний шаг − при вызове подпрограммы сообщить ей, какие фактические данные обрабатывать в этот раз. Для этого используется имя подпрограммы и следующий за ним список фактических параметров в скобках. Эти фактические параметры замещают формальные параметры при выполнении действий. Понятно, что должно быть обязательное соответствие между порядком следования формальных и фактических параметров. Описанный механизм обеспечивает подпрограммам качество, которое называют гибкостью, т. е. возможность настройки на разные данные.

Вызов подпрограммы приводит к выполнению некоторых действий или вычислению некоторого значения. Соответственно этому, различают два типа подпрограмм: подпрограммы-процедуры и подпрограммы-функции. Если говорить коротко, то различие между этими типами подпрограмм можно определить так. Функция, выполняя некоторые действия, вычисляет единственное значение, которое является основным результатом ее работы. Отработав, функция должна вернуть этот результат вызвавшей ее программе. Процедура просто выполняет какие-то действия, не возвращая никакого значения. Именно эти действия являются лавным результатом ее работы. При этом процедура может изменить, если это необходимо, значения некоторых объектов программы, к которым она имеет доступ. В качестве примера этих типов подпрограмм можно привести функцию, определяющую длину переданной ей строки (число символов в ней), и процедуру вывода информации на экран. Способ возврата значения, вычисленного функцией,

зависит от языка программирования. Разнообразие использованных понятий (фактические и формальные параметры, глобальные и локальные данные в подпрограммах) часто порождает путаницу у начинающих. Чтобы избежать ее, более четко поясним различие между ними.

Типы параметров

ФОРМАЛЬНЫЕ параметры подпрограммы – это не сами данные, передаваемые в подпрограмму, а только их описание, которое содержит информацию для подпрограммы о характеристиках этих данных и о действиях над ними. Образно можно представить, что формальные параметры – это "рамки", в которые потом, при использовании подпрограммы, будут вставлены "портреты" фактических "действующих лиц.

ФАКТИЧЕСКИЕ параметры – данные, фактически передаваемые подпрограмме при ее вызове. Эти данные должны быть описаны в вызывающей программе. Порядок перечисления и другие характеристики формальных и фактических параметров должны соответствовать друг другу.

ГЛОБАЛЬНЫЕ параметры – это данные, имена которых не присутствуют в списке формальных параметров и в описаниях данных внутри подпрограммы. Такие параметры должны быть описаны вне подпрограммы (например, в основной программе или ближайшей объемлющей подпрограмме). Внутри подпрограммы глобальные параметры используются в соответствии со смыслом этих внешних описаний.

ЛОКАЛЬНЫЕ параметры – это данные, которые описаны внутри самой подпрограммы. Эти параметры "недолговечные", они "живут" только во время работы подпрограммы. При начале работы подпрограммы они как бы "создаются" (в соответствии со смыслом описания), а при окончании работы "уничтожаются".

С понятием локальных параметров связан очень важный принцип современного программирования – ПРИНЦИП ЛОКАЛИЗАЦИИ.

Принцип локализации

Суть этого принципа состоит в том, что программный объект, описанный в подпрограмме, имеет смысл, соответствующий этому описанию. Иными словами, описание локализует смысл программного объекта в подпрограмме. Это необходимо пояснить. Допустим, в основной программе и подпрограмме используются одинаковые имена для обозначения разных по смыслу программных объектов. Например, имя целой переменной в программе и имя строки в подпрограмме, описанной как локальный параметр. При вызове подпрограммы порождается новый объект − строка с таким же именем, а то, что это имя целой переменной в

программе, как бы "забывается". То есть на время работы подпрограммы на это имя будет "отзываться" строка. Как только прекратится работа подпрограммы, имя станет указывать на целую переменную. Как важное следствие, принцип локализации дает возможность использовать в программе и подпрограммах одинаковые имена для обозначения разных программных объектов. Это особенно удобно при разработке больших программных проектов, состоящих из большого количества подпрограмм. Написание подпрограмм, когда определено их назначение, можно поручить разным людям, не ограничивая их в выборе имен.

Виды подпрограмм: ФУНКЦИЯ

Структура описания функции:

```
function
  <имя>(<список_форм_параметров>):<тип_результата>;
    < Раздел описания меток >
    < Раздел описания констант >
    < Раздел описания типов >
    < Раздел описания переменных >
    < Раздел описания функций и процедур >
 begin
    < Операторы >
    <имя> := <выражение> { ОБЯЗАТЕЛЬНЫЙ ОПЕРАТОР!!! }
 end;
```

Здесь <имя> – имя функции, через которое она возвращает свое значение в вызывающую программу. Для возврата значения функция языка Pascal обязательно должна содержать хотя бы один оператор, в котором имени функции присваивается значение; <список_форм_параметров> – это перечень описаний формальных параметров одного типа, каждое описание однотипных параметров вместе с именем типа отделяется от других таких же описаний (если, конечно, они есть) точкой с запятой.

ПРИМЕРЫ:

1.

```
     // Описание функции, определяющей, является ли
     // переданный ей символ латинской буквой
function ItIsLatinChar(Litera: char): boolean;
begin
ItIsLatinChar := ((Litera >= 'A') and (Litera <= Z'))
                 or
                 ((Litera >= 'a') and (Litera <='z'))
end;
.................
```

```
var
  Soobschenie: string;
  i : Integer;
  Soobschenie := Edit1.Text; //исходная строка текста
  for i := 1 to Length(Soobschenie) do
     // если не латинский символ, то заменить его пробелом
     // Вызов функции ItIsLatinChar
     if not ItIsLatinChar(Soobschenie[i]) then
       Soobschenie[i] := ' ');
  //преобразованная строка текста
  Edit2.Text := Soobschenie;
..................
```

2.

```
//Описание функции, подсчитывающей кол-во разрядов числа
.................
function NumberOfDigits(Number: integer): integer;
var
  i: integer;
begin
  i := 0;
  while ((Number div 10) <> 0) do
  begin
    Number := Number div 10;
    i := i + 1
  end;
  NumberOfDigits := i + 1 //Обязательный оператор!!
end;
.................
var
 Chislo, ChisloRazryadov : integer;
//Вызов функции
.........
 Chislo := 76588302;
 ChisloRazryadov := NumberOfDigits(Chislo);
```

Виды подпрограмм: ПРОЦЕДУРА

Структура описания процедуры:

```
procedure <имя_процедуры>(<список_форм_параметров>);
   < Раздел описания меток >
   < Раздел описания констант >
   < Раздел описания типов >
   < Раздел описания переменных >
   < Раздел описания функций и процедур >
begin
   < Операторы >
end;
```

<список_форм_параметров> имеет тот же смысл, что и для функции.

О параметрах-значениях и параметрах-переменных

В Object Pascal используются три вида процедур:

- без параметров;
- с параметрами-значениями;
- с параметрами-переменными.

Процедуры без параметров

Работа процедуры не зависит ни от каких внешних значений. Но и заставить ее делать что-то, незначительно отличающееся от заданного невозможно. Придется писать другую процедуру.

Пример:

```
// Эта процедура выводит на консоль двойную линию
// символами "=" длиной 60 символов
procedure Line;
var
  i: integer;
begin
  // вывод на экран консоли символа "=" 60 раз
  for i := 1 to 60 do write ('=');
  // переход на новую строку
  writeln
end;
.............
// использование процедуры
Line;
```

Сделать процедуру гибкой и настраиваемой на нужный вариант можно при использовании параметров.

Процедуры с параметрами-значениями

В этом случае в процедуру передаются только значения ее параметров. В самой процедуре компилятором создаются переменные (скрытые от программиста), которым присваиваются значения переданных параметров, т. е. создаются их копии, с которыми и производятся все дальнейшие действия. При этом, если даже в процедуре происходит изменение значений параметров, то меняются значения только копий. Сами же переданные процедуре параметры не изменяют своих значений.

Пример:

```
// Эта процедура выводит на консоль с заданной позиции
// линию заданной длины заданными символами
procedure Line(StartPos: integer; Symbol: char;
               Length: integer);
```

```
var
  i: integer;
begin
  // вывод на экран консоли пробелов
 for i := 1 to StartPos-1 do write (' ');
  // вывод на экран консоли заданного символа
 for i := 1 to Length do write (Symbol);
  // переход на новую строку
  writeln
end;
.............
// использование процедуры
Line(20, '-', 50);
Line(20, '*', 50);
Line(1, '-', 70);
```

Однако иногда требуется, чтобы процедура изменила значения переданных ей объектов программы, например, в процедуре обмена значениями двух переменных. В таких случаях используются параметры-переменные.

Процедуры с параметрами-переменными

Для таких параметров компилятор не создает копий, но обеспечивает непосредственный доступ к переданной процедуре переменной. Изменения этой переменной в процедуре сохраняются и в самой программе. В Паскале перед именем параметра-переменной должно стоять служебное слово **var**.

Пример:

```
procedure Obmen(var Znach1, Znach2:integer);
var
  Znach: integer;
begin
  Znach := Znach1;
  Znach1 := Znach2;
  Znach2 := Znach
end;

var
  Number1, Number2: integer;
 ...........
begin
 ........
  Number1 := 5;
  Number2 := 9;
  // Вызов процедуры Obmen
  Obmen(Number1, Number2);
```

Пример разработки программы с использованием процедурного подхода и метода пошаговой детализации

Рассмотрим метод пошаговой детализации на примере следующей задачи: **вычислить сумму разрядов заданного целого четырехзначного числа.**

Выделить нужный разряд числа можно с помощью операций целочисленного деления и нахождения остатка от деления и по следующему выражению:

$$разряд_n = число \ \mathbf{div} \ 10^{n-1} \ \mathbf{mod} \ 10$$

здесь n – номер разряда (Разряды нумеруются справа налево, первый разряд – разряд единиц, второй – разряд десятков и т. д.)

Проектирование начинается с записи последовательности действий, приводящих к желаемому результату. Эта запись делается на русском языке с использованием основных управляющих структур программирования. Полученный текст называется псевдокодом.

Этап 1. Запись псевдокода решения задачи

```
Ввод числа;
если число введено правильно то
начало
  Вычислить сумму разрядов числа;
  Вывод результата
конец
```

Эту задачу сразу можно писать в виде программы и отлаживать ее. Вот как будет выглядеть обработчик щелчка по кнопке, которая запускает рассматриваемую задачу в Delphi:

Этап 2. Запись основного алгоритма программы в терминах вызова процедур и функций

```
        var
        // Число и сумма его разрядов IntNumber,
   SumOfDigit: integer;
   // Признак правильного ввода числа
   WasRightInput: boolean;
procedure TForm1.Button1Click(Sender: TObject);
begin
   // Ввод числа
   GetNumber;
   if WasRightInput then
   begin
   // Вычисление суммы разрядов числа
   SumOfDigit := SumOfNumberDigit(IntNumber);
   // Вывод результата
   ResultOutput(SumOfDigit)
   end;
end;
```

Чтобы задачу сразу можно было отлаживать, необходимо до выполнения второго шага детализации (т. е. детальной записи процедур и функций) использовать так называемый «метод заглушек». Его суть состоит в том, что в каждой процедуре первого шага можно вместо подробного кода поместить, например, вывод сообщений. Например, процедура GetNumber вместо ввода числа может выдать сообщение: «Ввод числа проработал». Количество решений, принятых на первом шаге детализации, невелико, программа легко читается, лишние детали не затрудняют понимание. Логику программы первого шага детализации легко проверить. После завершения отладки программа этого этапа детализации уже готова, и можно сосредоточиться на деталях следующего уровня.

Этап 3. Запись ранее использованных процедур и функций (второй шаг детализации)

```
// Процедура ввода числа, в которой проверяется, было
// ли введено четырехзначное число
procedure GetNumber;
begin
   WasRightInput := False;
 // Был ли сделан ввод числа?
 if Form1.Edit1.Text <> '' then
   // проверка четырехзначности числа
   if Length(Form1.Edit1.Text) < 4 then
   begin
     ShowMessage('Число должно быть четырехзначным');
     Form1.Edit1.SetFocus;
   end
   else // Ввод правильный
   begin
     IntNumber := StrToInt(Form1.Edit1.Text);
     WasRightInput := True
   end
 else
 begin
   ShowMessage('Введите четырехзначное число');
   Form1.Edit1.SetFocus
 end;
end;
// Вычисление суммы разрядов числа
function SumOfNumberDigit(INumber: Integer): Integer;
begin
  SumOfNumberDigit := Digit(INumber, 1) +
    Digit(INumber, 2) + Digit(INumber, 3) +
    Digit(INumber, 4);
end;
```

```
// Вывод числа
procedure ResultOutput(INumber: Integer);
begin
  Form1.Memo1.Lines.Add('Сумма цифр числа ' +
    IntToStr(IntNumber) +' равна ' + IntToStr(INumber))
end;
// Обработчик события ввода символов, допускающий ввод
// только символов-цифр
procedure TForm1.Edit1Change(Sender: TObject);
var
  LastSym: char;
  SymbolIsNotDigit: boolean;
  Str: string;
begin
  if Edit1.Text <> '' then
  begin
    Str := Edit1.Text;
    LastSym := Str[Length(Str)];
    SymbolIsNotDigit := (LastSym < '0') or
                        (LastSym > '9');
    if SymbolIsNotDigit then
    begin
      ShowMessage('Неправильный символ.');
      Delete(Str, Length(Str), 1);
      Edit1.Text := Str;
    end
  end;
end;
```

Этап 4. Запись функции для вычисления заданной по порядку цифры числа (третий шаг детализации)

```
function Digit(Number, NumberDigit: Integer): Integer;
// локальная функция для вычисления заданной степени 10
   function _10_v_Stepeni(Exponent: Integer) : Integer;
   var
     i, Step10: Integer;
   begin
     i := 1;
     Step10 := 1;
     while (i < Exponent) do
     begin
       Step10 := Step10 * 10;
       i := i + 1
     end;
```

```
    // Возвращение результата
    _10_v_Stepeni := Step10
  end; // конец описания функции _10_v_Stepeni
begin
  Digit:=(Number div _10_v_Stepeni(NumberDigit)) mod 10;
end;
```

Классы и объекты

Понятие объекта и его описание

До середины 1980-х годов основным ограничением для создания больших программных систем была разобщенность в программе данных и методов их обработки. Решением проблемы стало появление нового направления в программировании, основанном на понятии объекта. В реальном мире объект – это нечто, обладающее совокупностью свойств, способностью изменять эти свойства разными методами и реагировать на события, происходящие как вне, так и внутри объекта. Это направление получило название объектно-ориентированного программирования (ООП).

По аналогии с этим в программировании **объект – это совокупность свойств (*структур данных*), методов их обработки (*подпрограмм изменения свойств*), и событий, на которые данный объект может реагировать**. Объекты объединяют в единое целое данные и средства действий над ними.

Классы – описание объектов

Как и в случае объектов реального мира, все объекты в программе можно классифицировать, взяв за основу классификации структуру объекта. Описание в программе объектов с одинаковой структурой называется классом. ***Класс*** – это средство описания типа объекта, помещается в разделе описания типов **type**. Действующие в программе объекты называются экземплярами класса. Описав в программе один раз класс, в дальнейшем можно **создавать** необходимое количество экземпляров этого класса – объектов.

Принадлежащие классу данные называются его **полями**. Средствами манипуляции над данными служат функции и процедуры, принадлежащие классу. Их называют **методами** класса.

Объекты как представители класса объявляются в программе в разделе **var**, например:

```
var
  student: TPerson;
  professor: TPerson;
```

Основные свойства классов

Основными свойствами классов являются:

- **Инкапсуляция**
- **Наследование**
- **Полиморфизм**

Эти три понятия являются основными для ООП.

Инкапсуляция − скрытие данных и методов внутри использующего их класса. Это означает, что данные и методы описываемого класса доступны для использования только ему.

Наследование − это возможность порождения новых классов от уже описанных. В этом случае данные и методы родительского класса автоматически включаются в порожденный класс и не нуждаются в повторном описании. Исходный класс называется предком, а порожденный от него класс-наследник − потомком.

Полиморфизм − это возможность использовать одинаковые имена для методов разных классов с общим предком, имеющих одинаковый смысл, но по-разному выполняющихся.

Структура описания класса

```
<ИМЯ КЛАССА> = class(<ИМЯ НАСЛЕДУЕМОГО КЛАССА>)
   // Для классов, описываемых в среде Delphi, здесь
   // помещаются описания компонентов Delphi и
   // заголовки методов-обработчиков событий
protected
   //Здесь помещаются описания элементов класса,
   //доступных напрямую в пределах данного модуля,
   // а также в классах-наследниках в других модулях
private
   // Здесь помещаются описания элементов класса,
   // доступных напрямую только в пределах данного //модуля
public
   // Здесь помещаются описания элементов класса,
   // которые доступны напрямую в пределах любого
   // модуля программы
end;
```

Пример описания класса для выделения разрядов целого числа

Описание класса для выделения разрядов целого числа может иметь вид (помещается в секцию **Interface** модуля):

```
Interface
type
  TRazriadyCelogo = class
```

```
  private
     Celoe : integer;
     Razriady : array [1..10] of integer;
     NomerRazriada : 1..10;
  public
     procedure PoluchitCeloe(Chislo: integer);
     procedure VydelitRazriady;
     function ZnachenieRazriada (N:integer) :integer;
end;
```

Теперь можно описать объект (переменную) этого типа:

```
var
  RazriadCelogo : TRazriadyCelogo;
```

Полное описание объявленных в классе процедур в помещается в секцию **Implementation** модуля:

```
Implementation
procedure TRazriadyCelogo.PoluchitCeloe(Chislo: integer);
begin
   Celoe := Chislo;
end;

procedure TRazriadyCelogo.VydelitRazriady;
var
  i, CelChislo : integer;
begin
  CelChislo := Celoe;
  i := 1;
  while ((CelChislo div 10) <> 0) and (i < 10) do
  begin
     Razriady[i] := CelChislo mod 10;
     CelChislo := CelChislo div 10;
     i := i + 1;
  end;
  Razriady[i] := CelChislo
end;

function ZnachenieRazriada (N:integer) :integer;
begin
  ZnachenieRazriada:= Razriady[N]
end;
```

Для создания объекта используется специальная процедура (конструктор объекта), которая создает объект во время выполнения про-

граммы. Для обращения к полям или методам объекта используется операция ".", отделяющая имя объекта от имени поля или метода.

```
RazriadCelogo.PoluchitCeloe(4321);
RazriadCelogo.VydelitRazriady;
NovoeChislo:=RazriadCelogo.ZnachenieRazriada(2)*10+
             RazriadCelogo.ZnachenieRazriada(4);
```

ГЛАВА 8. ОСНОВНЫЕ АЛГОРИТМЫ ОБРАБОТКИ ДАННЫХ

Наиболее часто в информационных и вычислительных задачах используются алгоритмы, связанные с обработкой массивов различных типов данных. К ним относятся:

- алгоритмы накопления (счетчик, вычисление суммы и произведения элементов массива);
- поиск экстремума;
- сортировка.

Далее эти алгоритмы приводятся в виде фрагментов программ.

Счетчик

```
//В этом фрагменте вычисляется количество отрицательных
//элементов массива
   KolOtr := 0;
      for i := 1 to 100 do
        if MassivChisel[i] < 0 then
          KolOtr := KolOtr + 1;
 ..............................
```

Накопление

```
var
  i: Integer; //номер элемента (параметр цикла)
  Summa, Proizvedenie: Integer;
  MassivChisel: array [1..100] of Integer;
begin
 ..............................
  Summa := 0;
  for i := 1 to 100 do
    Summa := Summa + MassivChisel[i];
 ..............................
  Proizvedenie := 1;
  for i := 1 to 100 do
    Proizvedenie:=Proizvedenie* MassivChisel[i];
 ..............................
end;
```

Поиск экстремума

Алгоритм поиска экстремума приведен здесь на примере поиска максимального элемента массива и его номера (индекса).

```
var
  i: Integer; //номер элемента(параметр цикла)
  Max, MaxNumber: Integer;
  MassivChisel: array [1..100] of Integer;
begin
.............................
  Max := MassivChisel[1];
  MaxNumber := 1;

  for i := 2 to 100 do
    if MassivChisel[i] > Max then
    begin
       Max := MassivChisel[i];
       MaxNumber := i
    end

end;
```

Сортировка

Сортировкой называется процесс упорядочивания данных по какому-либо признаку (например, расположение числовых данных в порядке возрастания). Существует несколько алгоритмов сортировки, различающихся эффективностью, большей или меньшей сложностью для программирования. Здесь рассмотрен простейший метод сортировки, получивший название "метод пузырька". Суть его в том, что наименьший элемент массива, подобно поднимающемуся в жидкости пузырьку воздуха, перемещается в начало массива, меняясь местом с первым элементом. Затем этот процесс многократно повторяется с оставшейся частью массива до тех пор, пока все элементы не будут упорядочены.

```
const
  NumOfElements = 100;
var
  MassivChisel: array [1..NumOfElements] of Integer;
  i, j, MinNumber: integer;
  ...........................
  for i := 1 to NumOfElements - 1 do
  begin
    MinNumber := i;
    for j := i+1 b NumOfElements do
      if MassivChisel[j] < MassivChisel[MinNumber] then
     MinNumber := j;
     if MinNumber <> i then Ob-
     men(MassivChisel[MinNumber], MassivChisel[i])
  end;
```

Алгоритм сортировки сводится к сочетанию двух алгоритмов: алгоритма поиска экстремума (минимума), который последовательно применяется к исходному массиву, затем к подмассиву, начинающемуся со второго элемента, с третьего и т. д. После нахождения минимума выполняется процедура обмена минимального элемента с первым в подмассиве. Процесс повторяется, пока все элементы не будут расположены в нужном порядке.

ГЛАВА 9. ДОПОЛНИТЕЛЬНЫЕ СРЕДСТВА ОПИСАНИЯ ДАННЫХ

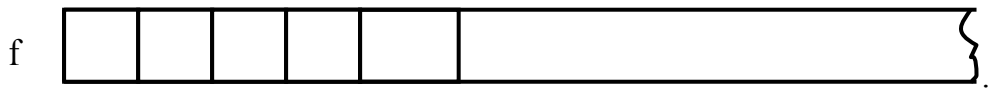
Рассмотренных ранее средств описания данных вполне достаточно для создания программ, реализующих алгоритмы практически любой сложности. Тем не менее, в языках программирования имеются типы данных, расширяющие возможности разработчика программ. До сих пор мы не упоминали о средствах, которые позволяют осуществить ввод данных в программу и вывод данных из нее, т. е. обеспечивают связь программы с "внешним миром" (внешними носителями данных, например, с магнитными дисками). Они необходимы, в частности, когда программа использует данные, подготовленные и записанные другой программой. Эти данные хранятся в файлах. Для представления таких данных в программе в языке Object Pascal реализован файловый тип данных. Переменные этого типа называются файловыми переменными (или, для краткости, тоже файлами). Другой класс средств не настолько необходим, но весьма облегчает жизнь программисту и позволяет очень компактно и ясно описывать алгоритмы. К такому классу относятся множества. Учитывая краткость курса, мы не будем подробно рассматривать множественный тип данных. Файловый тип вследствие его важности для связи программы с "внешним миром" нуждается в более подробном рассмотрении.

Файловый тип и файловая переменная

Часто встречаются задачи, в которых число однотипных элементов заранее неизвестно и может меняться в значительных пределах. Это число определяется в процессе решения задачи.

Для описания произвольной последовательности значений одного типа в Object Pascal существует специальный тип данных – файловый. Длина этой последовательности заранее не определяется. Этот тип данных не во всех языках представлен явно (т. е. на уровне описания переменных этого типа), но средства работы с файлами, находящимися на внешних устройствах, в виде стандартных библиотечных процедур имеются в любом современном языке программирования.

Условно файл можно изобразить в виде ленты, у которой есть начало, но нет определенного конца (хороший образ файла – магнитофонная лента с записью)



Новый элемент можно записать только в конец файла (тем самым увеличивая его длину). Такой файл называется последовательным. В последовательном файле доступ к определенному его элементу можно получить только двигаясь от начала, от его первого элемента. Если в процессе работы необходим один из предыдущих элементов, то приходится возвращаться к началу файла и снова проходить все до нужного элемента.

Над файлом в целом не определены никакие операции. Операции могут производиться лишь над элементами файлов, и определяются они типом элементов. Однако для любых языков программирования существуют библиотечные процедуры, обеспечивающие работу с файлами и их элементами. Эти процедуры можно рассматривать как стандартные операции над файлами:

- процедуры открытия файлов (устанавливают режимы работы):
  - открытие в режиме записи в файл;
  - открытие в режиме чтения из файла;
- запись в файл (значение некоторой переменной присваивается конечному элементу файла);
- чтение из файла (значение элемента файла присваивается некоторой переменной);
- закрытие файла.

Файл – это хранилище информации. Поскольку речь идет о хранении, то необходимо знать, куда можно положить данное и откуда его взять. Положение в файле, куда можно записать очередной его элемент или откуда можно его прочитать, называют текущей позицией в файле (или, проще, текущей позицией файла). Файл, открытый в режиме записи, сначала не имеет элементов (пустой файл). Его первая позиция одновременно является текущей и последней. Эта текущая позиция автоматически помечается специальным маркером – признаком конца файла. Именно поэтому открытие на запись даже ранее созданного файла приводит к исчезновению информации в нем. После записи элемента в текущую позицию маркер конца файла смещается в следующую позицию, и она становится текущей. При открытии же ранее созданного файла в режиме чтения, текущей позицией является первая, а маркер конца файла располагается в позиции, следующей за последней. В процессе чтения сдвигается только текущая позиция, а положение маркера остается неизменным.

Двоичные и текстовые файлы

Существует два вида файлов – двоичные и текстовые. Двоичные файлы подразделяются на типизированные и не типизированные. Информация и в тех и в других хранится в виде последовательности единиц информации – байтов, но при чтении или записи обрабатываются

они по-разному, в зависимости от вида файла. Текстовые файлы используются для хранения последовательности символов, разбитой на порции – строки, т. е. в виде, пригодном для чтения и редактирования обычных текстов. Любую другую информацию удобнее хранить в двоичных файлах. На уровне библиотечных процедур существуют различные средства для работы с двоичными и текстовыми файлами.

Задание файлового типа в программе производится в разделе описания типов:

```
<имя_файлового_типа> = <задание типа>
```

Задание типа определяется видом файла:

- `textfile` – для текстовых файлов;
- **`file of`** `<тип элементов>` – для типизированных файлов;
- **`file`** – для не типизированных файлов.

Примеры:

```
type
  zapis = record
    FIO: string[30];
    YearOfBorn: 1900..1980
  end;
  document = textfile;
  date = file of integer;
  personnel = file of zapis;
```

Переменная файлового типа называется файловой переменной или просто файлом. Такие переменные описываются в разделе описания переменных по общим правилам:

```
<имя_файловой_переменной> : <имя_файлового_типа>
```

Например, на основе описанных ранее типов могут быть описаны переменные:

```
var
  letter   : document;
  measuring: data;
  staff    : personnel;
```

Внешние и внутренние файлы

Файловая переменная, описанная в программе, существует только во время работы программы. Такая переменная называется внутренним файлом. Данные, записанные во внутреннем файле, после работы программы пропадают. Файлы, существующие на внешних устройствах,

называются внешними. Они могут использоваться для долговременного хранения информации и связи между программами по данным. Внутренние и внешние файлы связываются специальными библиотечными процедурами, которые устанавливают соответствие между именем внешнего файла и именем файловой переменной.

Процедуры работы с файлами

Хотя над переменными файлового типа не определены никакие операции, в библиотеке любой версии языка Pascal операции над файлами и их элементами представлены широким набором процедур. Здесь рассмотрены лишь основные из них. Для краткости описания процедур будем использовать следующие обозначения:

f – имя файловой переменной, описанной в программе;

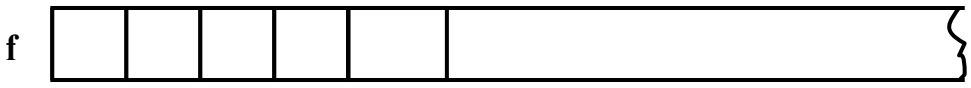
x – имя переменной, предназначенной для хранения элемента, прочитанного из файла или записываемого в него.

Открытие файла в режиме записи

Процедура `rewrite(f)` устанавливает файл с именем f в режим записи. После открытия текущей является первая позиция файла. Если файл был создан ранее и в нем уже были данные, то они пропадают.

Открытие файла в режиме чтения

Процедура `reset(f)` устанавливает файл с именем f в режим чтения. Первая позиция файла – текущая, данные сохраняются.



Связывание внутренних файлов с внешними

Процедура `assign(f, <имя_внешнего_файла>)` устанавливает связь внутреннего файла (файловой переменной с именем f) с внешним, имя которого указано в процедуре (например, процедура `assign(letter, 'letter.txt')` связывает описанный ранее файл письмо и существующий в текущем директории на диске файл с именем letter.txt).

Запись в файл

Процедура write(f, x) записывает в текущую позицию файла f значение переменной x (т. е. выводит значение x из программы в файл f). После записи текущей позицией становится следующая за ней:

до записи:

после записи:

Чтение из файла

Процедура read(f,x) считывает значение элемента из текущей позиции файла f и присваивает его переменной x (т. е. вводит значение x из файла f в программу). После чтения текущей позицией становится следующая.

Закрытие файла

Процедура close(f) закрывает файл f, сохраняя записанные в нем данные от разрушения. Закрытие всех открытых файлов происходит автоматически после завершения программы. Тем не менее, рекомендуется закрывать в программе файлы сразу после того, как они становятся не нужны для дальнейшей работы. Иначе при любом аварийном завершении программы данные в незакрытых файлах могут исчезнуть.

Функция проверки на достижение конца файла (eof)

При работе с файлом необходимо знать, когда достигнут его конец. Существует стандартная логическая функция EOF(f). Значение этой функции равно true, если текущая позиция совпадает с концом файла, и false − в любом другом случае.

Текстовые файлы

Для описания текстовых файлов существует специальный стандартный тип. Имя типа: textfile. Файл типа textfile представляет из себя единую последовательность литер, но это не то же самое, что файл типа file of char. В текстовом файле заложена возможность выделения отдельных порций − строк разной длины (в том числе и пустых) – при чтении или записи. Признаком конца строки служит специальный символ, не имеющий графического представления. Для определения конца строки в Паскале существует стандартная функция EOLN(f), дающая

значение true, если текущая позиция в файле f содержит признак конца строки, и false – в противном случае.

Процедуры ввода и вывода для текстовых файлов

Наряду с процедурами write и read для текстовых файлов в библиотеке языка Pascal имеются дополнительные процедуры ввода и вывода writeln и readln. Процедура writeln(f, x) после вывода значения x заносит в файл f признак конца строки. Если второй параметр процедуры writeln отсутствует, то выводится только признак конца строки. Процедура readln(f, x) читает из файла f значение элемента в текущей позиции и присваивает его переменной x, после этого текущей позицией становится позиция файла, следующая за ближайшим "концом строки". Если между прочитанным значением и "концом строки" в файле были другие данные, то они пропускаются. Отсутствие второго параметра приводит просто к перемещению текущей позиции файла в позицию, следующую за ближайшим "концом строки". Таким образом, в случае одного параметра процедуры readln(f) и writeln(f) могут использоваться для пропуска нужного количества строк при чтении или записи.

Ввод и вывод с использованием списков параметров

В процедурах read, readln, write, writeln разрешено использовать списки вводимых или выводимых параметров. Например,

```
writeln(f, x1, x2,...,xN)
```

полностью эквивалентно последовательности процедур

```
write(f,x1); write(f,x2);...; write(f,xN); writeln(f).
```

При работе со стандартными текстовыми файлами при вводе можно использовать не только значения литерного типа, но и целые и вещественные. При этом требуется соблюдать несколько правил, самое важное из которых гласит:

тип вводимого данного должен соответствовать типу переменной в списке параметров процедуры read; т. е., если параметр целого типа, то с клавиатуры должно быть введено целое число, для литерного – литера и т. д.

Динамические массивы

Динамические массивы отличаются от обычных статических массивов тем, что в них не объявляется заранее длина − число элементов. Поэтому динамические массивы удобно использовать в приложениях, где объем обрабатываемых массивов заранее неизвестен и определяется в процессе выполнения в зависимости от действий пользователя или объема перерабатываемой информации.

Одномерные динамические массивы

Объявление динамического массива содержит только его имя и тип элементов – один из базовых типов. Синтаксис объявления:

```
<имя> array of <базовый тип>
```

Например, код в программе

```
var
  A: array of integer;
```

объявляет переменную A как динамический массив целых чисел.

При объявлении динамического массива место под него не отводится. Прежде, чем использовать массив, надо задать его размер процедурой SetLength. В качестве аргументов в нее передаются имя массива и целое значение, характеризующее число элементов. Например, обращение к процедуре

```
SetLength(A, 10);
```

выделяет для массива A место в памяти под 10 элементов и задает нулевые значения всех элементов. Индексы динамического массива – всегда целые числа, начинающиеся с 0. Таким образом, в приведенном примере массив содержит элементы от A[0] до A[9]. Повторное применение SetLength к уже существующему в памяти массиву изменяет его размер. Если новое значение размера больше предыдущего, то все значения элементов сохраняются и просто в конце добавляются новые нулевые элементы. Если же новый размер меньше предыдущего, то массив усекается, и в нем остаются значения первых элементов. Например, для приведенной ниже программы размерность и значения элементов массива при выполнении различных операторов показаны в комментариях к тексту.

```
var
  A: array of integer;
  N,i: integer;
begin
  N:=5;
  SetLength(A, N); // массив A(0,0,0,0,0)
  for i:=0 to N-1 do A[i]:=i+1; // массив A(1,2,3,4,5)
  N:=7;
  SetLength(A, N) ; // массив A(1,2,3,4,5,0,0)
  N:=3;
  SetLength(A, N); // массив A{1,2,3)
  N:=4;
  SetLength(A, N); // массив A(1,2,3,0)
end;
```

Впрочем, усечение динамического массива можно проводить и функцией `Copy`, присваивая ее результат самому массиву. Например, оператор

```
A := Copy(A, 0, 3);
```

усекает динамический массив A, оставляя неизменными первые три его элемента. Если динамический массив уже размещен в памяти, к переменной этого массива можно применять стандартные для массивов функции `Length` – длина, `High` – наибольшее значение индекса (очевидно, что всегда `High` = `Length -1`) и `Low` – наименьшее значение индекса (всегда 0). Если массив имеет нулевую длину, то `High` возвращает –1, т. е. при этом получается, что `High` < `Low`. Сама переменная динамического массива является указателем (специальный тип данных, значениями которого являются адреса данных в памяти) на начало массива.

Удалить из памяти динамический массив можно установив для него нулевую длину:

```
SetLength(A,0);
```

Если динамические массивы определены как переменные одного типа, например,

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

то возможно присваивание вида

```
B := A;
```

которое приводит к тому, что переменная `B` начинает указывать на тот же самый массив, что и `A`, т. е. получается как бы два псевдонима для одного массива. А содержимое массива `B` при этом теряется. *В этом коренное различие присваивания статических и динамических массивов*.

Если динамические массивы объявлены не как переменные одного типа, т. е.

```
var
  A: array of integer;
  B: array of integer;
```

то присваивание

```
B := A;
```

вообще не допускается.

В операциях сравнения динамических массивов сравниваются только сами указатели (адреса начала массивов), а не значения элементов массивов. Таким образом, выражение A = B вернет true только в случае, если A и B указывают на один и тот же массив. А вот выражение A[0] = B[0] сравнивает значения первых элементов двух массивов.

Динамические массивы могут передаваться в качестве параметров в те функции и процедуры, в описаниях которых параметр объявлен как массив базового типа без указания индекса, т. е. открытый массив. Например, функция

```
function CheckStrings(A: array of string): Boolean;
```

может работать в равной степени и со статическими, и с динамическими массивами.

Многомерные динамические массивы

Многомерный динамический массив определяется как динамический массив динамических массивов динамических массивов и т. д. Например:

```
var
  A2: array of array of integer;
```

определяет двумерный динамический массив. Один из способов отвести память под такой массив − использовать ту же процедуру SetLength, которая использовалась для одномерного массива, но передавать ей в качестве параметров не один, а несколько размеров. Например, оператор

```
SetLength(A2,3,4);
```

задает размер массива 3 на 4.

Доступ к элементам многомерных динамических массивов осуществляется так же, как и для статических массивов. Например, A2[1,2] − элемент, лежащий на пересечении второй строки и третьего столбца (индексы считаются от нуля, так что индекс 1 соответствует второй строке, а индекс 2 − третьему столбцу). Можно создавать и более интересные объекты − непрямоугольные массивы, в которых, например, второй размер не постоянен и варьируется в зависимости от номера строки. В этом случае сначала процедурой SetLength задается первый размер. Например, оператор

```
SetLength(A2,3);
```

задает первый размер − 3. В памяти отводится место под 3 строки − 3 динамических массива A2[0], A2[1] и A2[2]. Размеры каждого из этих массивов еще не определены. Далее можно, например, задать размер первого из них равным 4:

```
SetLength(A2[0],4);
```

а размер следующего, например, равным 5:

```
SetLength(A2[1],5) ;
```

В качестве примера приведем программу построения и заполнения нижней треугольной матрицы произвольного размера N.

```
var
  A2: array of array of integer;
  N,il,i2,m: integer;
begin
  N:=3;
  m: = 1;
  SetLength(A2,N); // Задание числа строк = N
  for il:=0 to N do // Цикл по строкам
  begin
  // Число столбцов равно номеру строки
    SetLength(A2[il] , il + 1) ;
    for i2:=0 to il do
    begin
      A2 [il,i2] :=m; // Заполнение строки
      Inc(m); // Увеличение m на 1
    end;
  end;
end;
```

Программа формирует двумерный массив, в котором число строк равно значению N, а число столбцов в каждой строке равно номеру строки. В результате получается следующая матрица:

1
2 3
4 5 6

ГЛАВА 10. ОБЪЕКТНО-ОРИЕНТИРОВАННЫЙ ПОДХОД К ВВОДУ/ВЫВОДУ ИНФОРМАЦИИ

Общие сведения о потоках

Разнообразие физических устройств, предназначенных для хранения информации, усложняет реализацию программных продуктов, заставляя разрабатывать уникальные для каждого устройства программные модули. Часть подобных задач берет на себя операционная система, представляя одинаковый программный интерфейс к похожим устройствам, например, Flash-памяти и жестким дискам. Однако этого не достаточно.

С другой стороны, если рассматривать загрузку и сохранение информации с точки зрения логики процесса, без учета особенностей конкретного устройства, будь то оперативная память, магнитный носитель, или сетевое соединение, то можно легко описать абстрактное устройство для хранения информации. Вне зависимости от типа устройства ввода/вывода оно должно поддерживать довольно узкий набор возможностей, таких, как:

- запись информации из программы на устройство;
- чтение информации из устройства в программу;
- определение параметров устройства, например, его емкости.

Систематизация устройств ввода/вывода с такой позиции обеспечивает возможности объектно-ориентированного подхода к процессам ввода/ вывода с помощью создания некоторого базового класса, описывающего абстрактное устройство, и классов-наследников, реализующих заявленные методы чтения и записи информации в зависимости от особенностей конкретного устройства. Такой подход не только снижает сложность программы за счет использования объектов, но и придает ей значительную гибкость, так как переход с одного устройства хранения информации на другое не затрагивает фрагментов программы, непосредственно записывающих и считывающих информацию. Абстрактное устройство ввода/вывода называется потоком.

Базовым классом, описывающем потоки (англ. Stream – поток), является TStream, в котором заявлены методы чтения и записи информации с устройства и на устройство. Кроме того, в нем содержатся свойства, с помощью которых можно получить емкость устройства и некоторые параметры процесса чтения или записи. Имеется также целый набор классов-наследников TStream, каждый из которых предназначен для работы с конкретным устройством, или ориентирован на работу с информацией определенного типа. Список таких классов приведен в табл. 3.

На рис. 102 представлена схема работы программы, использующей потоки для ввода и вывода информации. Фрагменты программы (подпрограммы) обращаются к ссылке на экземпляр конкретного потока как к абстрактному устройству TStream. Далее, в зависимости от реального класса, на экземпляр которого указывает ссылка, информация перенаправляется на соответствующее физическое устройство, например, в файл.

Таблица 3

Стандартные классы-потоки

Класс	Назначение
TFieStream	Работа с файлами
TStringStream	Работа со строками, хранимыми в оперативной памяти
TMemoryStream	Работа с динамической памятью
TBlobStream	Работа с двоичными BLOB-полями баз данных
TWinSocketStream	Работа с сетевыми соединениями
TOleStream	Ввод/вывод информации в OLE-объектах

Возвращаясь к вопросу о гибкости программы по отношению к устройствам хранения информации, заметим, что при использовании потоков логика чтения и записи информации локализуется в двух разных частях программы. Указание на информацию, с которой следует произвести какие-либо действия, происходит в подпрограммах, а непосредственная работа с физическими устройствами обеспечивается экземпляром класса-потока.

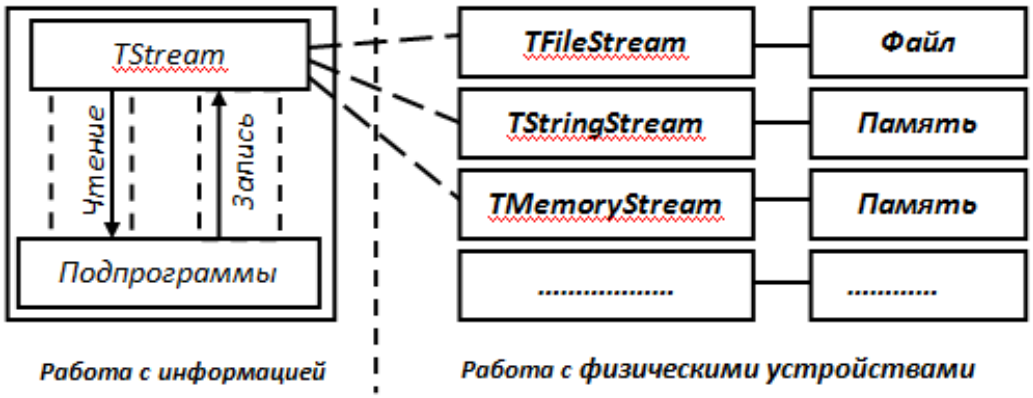


Рис. 102. Схема работы программы с потоками ввода/вывода

Предположим, что в программе используется файловый ввод и вывод информации, для чего в переменную Stream типа TStream заносится ссылка на экземпляр класса TFileStream:

```
.........
 var
   Stream: TStream; {Описание ссылки на абстрактное
                     устройство}
```

```
..........
Stream := TFileStream.Create (...);
                {Создание экземпляра класса
                для работы с файлами}
..........
Stream.Read(...);   {Чтение данных из потока}
Stream.Write (...); {Запись данных в поток}
..........
Stream.Free;        {Разрушение экземпляра класса}
..........
```

При подобной организации программы для перехода на другое устройство ввода/вывода достаточно изменить тип создаваемого объекта на соответствующий новому устройству. Например, для использования динамической памяти вместо файла на диске, следует создать экземпляр класса TMemoryStream, а не класса TFileStream, остальная же часть программы остается неизменной:

```
..........
var
  Stream: TStream; {Описание ссылки на абстрактное
                    устройство}
..........
Stream := TMemoryStream.Create (...);
                 {Создание экземпляра класса
                  для работы с памятью}
..........
Stream.Read(...) ;  (Чтение данных из потока}
Stream.Write (...); (Запись данных в поток}
..........
Stream.Free; {Разрушение экземпляра класса}
..........
```

Базовый класс для работы с потоками

Информация о состоянии устройства ввода/вывода

В базовом потоковом классе TStream описано свойство Size, определяющее объем информации, который может храниться в потоке:

```
Property Size: Int64;
```

Данное свойство обычно используется для определения текущего размера потока, однако некоторые наследники класса TStream реализуют изменение размера потока при изменении свойства size.

Потоки построены на основе последовательного доступа к информации. Это значит, что после открытия потока информация считывается из его начала (записывается в начало). Позицию, откуда будет считана (куда будет записана) очередная порция информации, определяет свойство Position:

```
property Position: Int64;
```

Свойство Position доступно для изменения. Например, для указания потоку, что запись следует производить в его начало, можно воспользоваться следующей конструкцией:

```
<Поток>.Position:= 0;
```

Создание и разрушение потока

Создание экземпляров потоковых классов существенно зависит от типов устройств, с которыми они связаны, так как в конструкторе любого потокового класса передается вся информация, необходимая для установки связи с устройством. Формат конструкторов для разных видов потоков рассматривается ниже.

Разрушение потока производится в обычном порядке с помощью вызова метода Free:

```
Экземпляр потокового класса>.Free;
```

Чтение и запись информации в поток

Чтение информации из потока производится на уровне двоичной информации с помощью методов Read и ReadBuffer, а также на уровне экземпляров классов-наследников TComponent с помощью метода ReadComponent:

```
Function Read(var Buffer; Count: Longint): Longint;
Procedure ReadBuffer (var Buffer; Count: Longint);
Function ReadComponent (Instance:TComponent): TComponent;
```

Для записи информации в поток используются аналогичные методы:

```
Function Write(var Buffer; Count: Longint):Longint;
Procedure WriteBuffer (var Buffer; Count: Longint);
Procedure WriteComponent(Instance: TComponent);
```

Методы Read и Write при описании в классе TStream являются абстрактными, а методы ReadBuffer и WriteBuffer их вызывают. Таким образом, создание и использование экземпляра класса TStream недопустимо. Классы, реализующие потоки, связанные с реальными устройствами, переопределяют данные методы для достижения соответствующей функциональности.

Для записи в поток информации, хранимой в другом потоке, вне зависимости от того, экземпляром какого класса он является, используется метод CopyFrom:

```
Function CopyFrom(Source:TStream; Count:Int64):Int64;
```

Для изменения текущей позиции в потоке предусмотрено свойство Position, однако в некоторых случаях более удобно использовать метод Seek, изменяющий текущую позицию относительно настоящего значения:

```
Function Seek(const Offset:Int64;
              Origin:TSeekOrigin):Int64;
```

Метод Seek перемещает позицию текущего указателя на количество байтов, заданное параметром Offset относительно начала потока, конца потока, или текущего положения указателя. Возможные значения данного параметра приведены в табл. 4.

Таблица 4

Возможные значения параметра Origin метода Seek класса TStream

Значение	Описание
soFromBeginnino	Текущий указатель устанавливается на Offset байт от начала потока. Значение Offset должно быть положительным
soFromCurrent	Текущий указатель смещается на Olfset байт от текущего положения. Значение Offset может быть как положительным, так и отрицательным
soFromEnd	Текущий указатель устанавливается на Offset байт от конца потока в сторону его начала Значение Offset должно быть отрицательным

Особенности реализации разных потоков

Файловые потоки

Файловые потоки предназначены для ввода/вывода информации с помощью файлов, реализованы в виде класса TFileStream, и поддерживают все возможности операционной системы, предусмотренные для работы с файлами. Для создания файлового потока TFileStream предусмотрено два конструктора:

```
Constructor Create (const FileName:string; Mode:Word);
overload;
Constructor Create (const FileName: string; Mode: Word;
Rights: Cardinal) ; overload;
```

Вторая версия конструктора имеет дополнительный параметр Rights (англ. Right − Права), использование которого имеет смысл только в операционной системе Linux. В Windows параметр Rights игнори-

руется и его рассмотрение нами не имеет смысла. Параметры FileName и Mode (англ. Mode − Режим) определяют соответственно название файла, с которым связывается поток, и набор флагов доступа к файлу. Флаги доступа разделяются на две группы: направления движения информации и разделения доступа.

По направлению движения информации файлы могут открываться (см. табл. 5):

- только для чтения;
- только для записи;
- для чтения и записи одновременно.

Таблица 5

Флаги, используемые при создании файлового потока для определения направления движения информации

Флаг	Доступ к файлу
fm Create	Создание файла для записи. Если файл с заданным именем уже существует, то он открывается для записи, что эквивалентно использованию флага fmOpenWrite. Если файл с заданным именем уже существует, то информация, находящаяся в нем будет утеряна
fmOpenRead	Открытие файла для чтения
fmOpenWrite	Открытие файла для записи. Если файл с заданным именем уже существует, то информация, находящаяся в нем, будет утеряна
fmOpenReadWrite	Открытие файла для чтения и записи одновременно

Разделение доступа определяет, какие операции могут выполняться над файлом другими приложениями во время работы с ним создаваемого потока (см. табл. 6):

Таблица 6

Флаги, используемые при создании файлового потока для определения типа разделения доступа к файлу

Флаг	Разделение файла между приложениями
fmShareExclusive	Файл открывается в монопольном режиме, во время работы с файлом ни одно приложение не может записывать в этот файл информацию и считывать ее из него
fmShareDenyWrite	Другие приложения могут считывать информацию из файла, но не могут ее записывать в него
fmShareDenyRead	Другие приложения могут записывать информацию в файл, но не могут считывать ее из него
frnShareDenyNone	Другие приложения могут использовать файл для любых операций

Потоки на основе оперативной памяти

Для временного хранения информации в оперативной памяти предусмотрен потоковый класс TMemoryStream. Этот класс поддерживает все свойства и методы, описанные в TStream, и реализует дополнительные методы для сохранения в файл записанной в него информации, а также и чтения информации из файла.

Поток TMemory Stream построен на основе нетипизированного указателя, который может быть доступен для чтения через свойство Memory типа Pointer, если есть необходимость прямого доступа к данным.

Для создания потока TMemoryStream используется конструктор Create без параметров:

```
Constructor Create;
```

Для чтения информации из файла с одновременным занесением ее в поток используется метод LoadFromFile:

```
Procedure LoadFromFile (const FileName: string);
```

и аналогичный метод для записи содержимого потока в файл:

```
Procedure SaveToFile (const FileName: string);
```

Кроме того, предусмотрены метод для чтения информации из другого потока вне зависимости от его типа, а также метод записи хранимой информации в другой поток:

```
Procedure LoadFromStream(Stream: TStream);
Procedure SaveToStream(Stream: TStream);
```

Также, для потоков, работающих с оперативной памятью, определена операция очистки содержимого с помощью метода Clear:

```
Procedure Clear;
```

Строковые потоки

Строковые потоки используются для доступа к строкам, хранимым в памяти. Реализованы строковые потоки в виде класса TStringStream. В каждом экземпляре такого класса хранится одна строка, доступ к которой возможен с помощью обычных методов, характерных для потоков (то есть Read, Write, и другими).

Если программе, использующей такой поток, необходим доступ к хранимой информации, как к строке в целом, то ссылка на строку может быть получена с помощью свойства DataString:

```
property DataString: string;
```

Помимо методов записи информации, унаследованных от класса TStream, в строковых потоках реализована возможность записи и чтения строк:

```
Function ReadString(Count: Longint): string;
Procedure WriteString (const AString: string);
```

При чтении и записи строк в строчные потоки учитывается положение текущего указателя, то есть значение свойства Position.

ГЛАВА 11. ОБЪЕКТНО-ОРИЕНТИРОВАННЫЙ ПОДХОД К ОБРАБОТКЕ ОШИБОК

Всякое взаимодействие с операционной системой на предмет получения ресурсов – места на диске, в памяти, открытие файла – может завершиться неудачно. Любое вычисление может закончиться делением на ноль или переполнением. Дополнительный фактор возникновения исключительных ситуаций содержится в данных, к которым могут обращаться программы. Особенно это актуально в приложениях баз данных.

Платой за надежную работу программы в таких условиях служит введение многочисленных проверок, способных предотвратить некорректные действия в случае возникновения нештатной ситуации. Хорошо, если в конце очередной конструкции if..then можно просто поставить оператор Exit. Обычно же для корректного выхода из ситуации нужно отменить целую последовательность действий, предшествующих неудачному. Все это сильно запутывает программу, маскируя четкую структуру главного алгоритма.

От этого громоздкого кода можно раз и навсегда избавиться, взяв на вооружение механизм исключительной ситуации, реализованный в Delphi.

Исключительная ситуация (исключение) – это некоторая ситуация в программе, которая требует специальной обработки. Компилятор Delphi генерирует код, который перехватывает любое нештатное событие, сохраняет необходимые данные о состоянии программы, и выдает разработчику объект.

С точки зрения Object Pascal ***исключительная ситуация – это объект специального вида, характеризующий возникшую в программе ситуацию***.

Особенностью исключения является то, что оно – временный объект. Как только исключение обрабатывается каким-либо обработчиком, оно сразу же разрушается.

При возникновении исключительной ситуации Delphi создает объект некоторого класса, в зависимости от характера произошедшей ошибки. **Все такие классы являются наследниками класса Exception** (англ. **E**xception − исключение), что позволяет им быть «узнаваемыми» для Delphi. **Механизм исключений состоит в досрочном прерывании подпрограммы или метода объекта, в которых произошла ошибка, и поиск обработчика исключения в этой подпрограмме** (реализация обработчиков описана ниже).

Если обработчик не найден, то управление передается подпрограмме, вызвавшей ту подпрограмму, в которой произошла исключительная ситуация. В вызвавшей подпрограмме также производится по-

иск обработчика, и в случае его отсутствия управление передается следующей подпрограмме по стеку вызова подпрограмм.

Искомый обработчик должен обрабатывать именно те виды исключений, объект которого создан во время возникновения ошибки. Если ни в одной из подпрограмм стека вызова не будет найден обработчик, то исключение передается стандартному обработчику исключений, автоматически создаваемому в программе. В этом случае на экран будет выдан диалог с сообщением об ошибке, а программа продолжит свое выполнение в ожидании действий пользователя.

Обработка исключений

Когда некоторый фрагмент программного **кода** необходимо защитить от досрочного завершения по ошибке, его следует заключить в **блок обработки исключения** («защищенный **блок»).** Таких блоков в Delphi предусмотрено **два,** а отличаются они характером действий, выполняемых в случае ошибки.

Блок обработки исключений `try...except`

Блок обработки исключений `try...except` (от англ. **try** – пробовать и **except** – кроме) пытается выполнить заданный фрагмент программы, а в случае появления исключения – передает управление специальному обработчику, расположенному в секции `except`:

```
try
  <фрагмент программы>
  except
    On <Класс исключения> do <Обработчик исключения >;
    …
    On <Класс исключения> do <Обработчик исключения >;
  else <Обработчик по умолчанию>
end;
```

Таким образом, если в контролируемом фрагменте программы, расположенном между ключевыми словами **try** и **except**, произошло исключение, то управление передается в секцию **except**. В секции except производится поиск обработчика, который соответствует классу выброшенного исключения. Если такой обработчик не найден, то **вызывается** обработчик по умолчанию, находящийся после ключевого слова **else**. Если же ключевое слово **else** не используется и при этом не найден обработчик исключения, то исключение передается в подпрограмму, вызвавшую данную, и так далее, в соответствии с механизмом, описанным выше.

```
var
    rezult, shislo : shortint;
…
  try
    B := strtoint(edit1.Text); // ввод числа
    rezult := 100 div shislo;
    edit2.Text:= inttostr(rezult)
  except
    on EConvertError do
    begin
    MessageDlg('Ошибка! Введите число!',mtWarning[mbOK],0;
     edit1.Clear;
     edit2.Clear;
    end;
   on EDivByZero do
      MessageDlg(Введён ноль! Ошибка!', mtWarning,[mbOK],0)
end;
```

Конечно, оператор else должен идти последним, так как в противном случае все обработчики, записанные после него, работать не будут, поскольку все исключения уже будут перехвачены. Внутри обработчика по умолчанию можно получить доступ к объекту исключения, воспользовавшись функцией ExceptObject. Например:

```
MessageBox.Show('Ошибка ' + ExceptObject.ToString,
                'Ошибка1, MessageBoxButtons.OK,
                MessageBoxIcon.Error);
```

Обратим внимание на некоторые особенности работы блока **try...except**.

1. **Если в секции except не используется ни одной структуры on… do**, а, соответственно, и ключевого слова else, то есть секция except представляет собой простую последовательность каких-либо команд, то **содержание этой секции считается обработчиком любого исключения.**

2. При определении обработчика исключения используются правила совместимости типов, поэтому для обработки исключений разных типов, классы которых являются наследниками от одного и того же класса, может быть использован родительский класс. Таким образом, **обработчик on exception do. . . будет являться обработчиком любого исключения,** так как все классы исключений являются наследниками класса Exception.

3. Если обработчик исключения в результате своей работы вызовет еще одно исключение, оно будет обрабатываться по общим правилам. Такое исключение может быть обработано и программно, то есть текст обработчика исключения может быть, в свою очередь, заключен в блок обработки.

4. Если для возникшего исключения подобран обработчик, то сообщение пользователю не выдается.

5. В раздел except могут включаться или только операторы **on** или только другие операторы. Смешение операторов **on** с другими не допустимо.

Последовательность обработки исключений

Блоки **try...except** могут быть вложенными явным или неявным образом.

Примером неявной вложенности является блок **try...except**, в котором среди операторов раздела **try** имеются вызовы функций или процедур, которые имеют свои собственные блоки **try...except**. Рассмотрим последовательность обработки исключений в этих случаях.

При генерации исключения сначала ищется соответствующий ему обработчик **on** в том блоке **try...except**, в котором создалась исключительная ситуация.

Если соответствующий обработчик не найден, поиск ведется в обрамляющем блоке **try...except** (при наличии явным образом вложенных блоков) и т. д. Если в данной функции или процедуре обработчик не найден или вообще в ней отсутствуют блоки **try...except**, то поиск переходит на следующий уровень в блок, из которого была вызвана данная функция или процедура. Этот поиск продолжается по всем уровням. И только если он закончился безрезультатно, выполняется стандартная обработка исключения, заключающаяся, как уже было сказано, в выдаче пользователю сообщения о типе исключения.

Как только оператор **on**, соответствующий данному исключению, найден и выполнен, объект исключения разрушается и управление передается оператору, следующему за тем блоком **try...except**, в котором был осуществлен перехват.

Возможен также вариант, когда в самом обработчике исключения в процессе обработки возникла исключительная ситуация. В этом случае обработка прерывается, прежнее исключение разрушается и генерируется новое исключение. Его обработчик ищется в блоке **try...except**, внешнем по отношению к тому, в котором возникло новое исключение.

Блок обработки исключений try..finally

Блок обработки исключений try. .except может быть использован в ситуациях, когда известно, как именно нужно реагировать на произошедшую ошибку, даже если программа и не пытается установить ее тип (не используются конструкции **on. .do. .else**).

Однако встречаются ситуации, когда реакция программы на ошибку не так важна, как корректное завершение фрагмента программы, вызвавшего эту ошибку. Такие ситуации связаны обычно с необходимостью вернуть операционной системе ресурсы, выделенные ошибочному фрагменту программы. Причем, возвращение ресурсов должно произойти как в случае успешного, так и в случае ошибочного выполнения фрагмента. Для реализации такого поведения программы предусмотрен блок обработки исключений try. . finally:

```
try
  <операторы, способные привести к появлению исключений и появлению «мусора»>
finally
  <операторы, выполнимые в любом случае и приводящие к зачистке «мусора»>
end;
```

При таком способе организации обработки исключений сначала производится попытка выполнить фрагмент программы после слова try. В случае успешного его выполнения, управление передается фрагменту программы после слова finally, и блок обработки заканчивается.

Если в фрагменте программы после слова **try** возникает исключительная ситуация, то его выполнение обрывается, но управление все равно передается фрагменту программы после слова **finally**. После выполнения этого фрагмента программы исключение создается заново и происходит его обработка по обычным правилам, например, с помощью блока обработки исключений **try...except**.

```
var
  F: File;
begin
  Assign(F, 'SOMEFILE.EXT');
  Reset(F);
  try
    {действия с файлом F}
  finally
    Close(F);
  end;
end;
```

Различия в работе блоков обработки исключений

На рис. 103 представлены три варианта фрагмента программы, в которой происходит ошибка времени выполнения. Каждый из фрагментов состоит из трех процедур, последовательно вызывающих друг друга. Ошибка происходит в третьей. На рисунке показаны три варианта работы в зависимости от вида используемого блока обработки или без него.

В первом варианте обработка ошибки не производится (чего в принципе не должно быть в корректно созданной программе). Второй вариант позволяет локализовать исключение, то есть не дать ему выйти за пределы какого-либо фрагмента программы. Благодаря этому остается возможным продолжение нормального выполнения программы. Далее мы остановимся более подробно на каждом из вариантов.

Программа на схеме «а» не обрабатывает исключительную ситуацию и работает следующим образом. Сначала выполняется Процедура I до момента вызова Процедуры 2. Путь корректного (безошибочного) выполнения программы показан сплошной линией. Затем Процедура 2 выполняется до момента вызова Процедуры 3. Процедура 3 выполняется, и в ней происходит ошибка, что влечет автоматическое создание объекта исключения и поиск обработчика, соответствующего данному исключению.

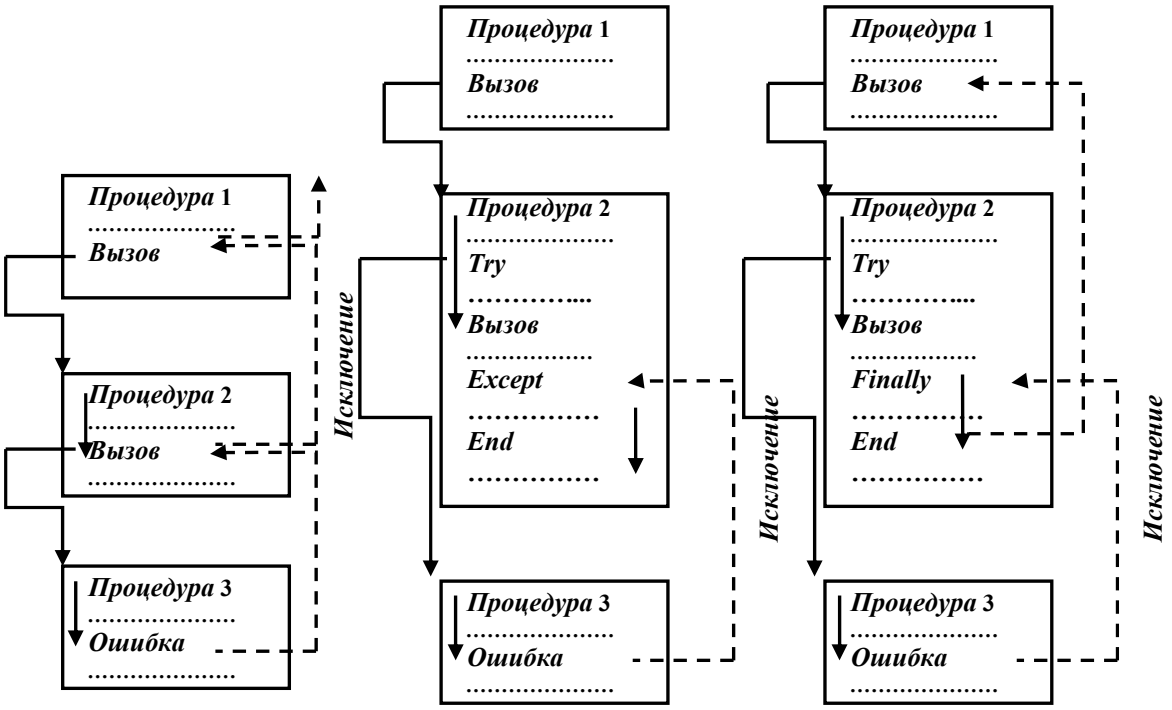


Рис. 103. Механизмы обработки исключительных ситуаций

Так как Процедура 3 не обрабатывает исключительную ситуацию, то она завершается, управление передается в вызвавшую ее Процедуру 2, и уже там начинается поиск обработчика исключения (направление потока выполнения при возникновении исключительной ситуации показано пунктирной линией). Однако Процедура 2 также не содержит обработчика исключений, следовательно, она будет прервана, а управление перейдет к Процедуре 1. Процедура I не содержит обработчика ошибок, следовательно, также будет прервана. В результате фрагмент программы, вызвавший, но не обработавший ошибку, вообще теряет управление.

Теперь рассмотрим схему «б». Программа, представленная на ней, работает аналогично схеме «а» до момента возврата управления в Процедуру 2 при возникновении исключения. Вызов Процедуры 3 в Процедуре 2 был сделан из блока обработки **try...except**, поэтому управление передается в секцию **except** этого блока, после окончания которой фрагмент программы продолжает свое нормальное выполнение.

На схеме «в» представлен фрагмент программы, имеющий блок обработки исключений **try...finally**. Его работа аналогична предыдущему случаю, за исключением того, что после завершения блока **finally...end**, Процедура 2 не будет продолжать свою работу, а завершится, и управление перейдет в Процедуру 1. Далее фрагмент будет работать аналогично представленному на рисунке «а».

С помощью исключений стандартные подпрограммы и методы классов Delphi сообщают об ошибках, происходящих внутри них. Преимущество такого подхода состоит в том, что программист не должен анализировать значение, возвращаемое подпрограммами, а может сосредоточиться на логике собственной программы, считая, что каждый фрагмент программы работает корректно, а в случае ошибки управление будет передано в соответствующее место автоматически.

Управление программой с помощью исключений является наиболее прогрессивным в современном программировании и рекомендуется для использования при написании собственных программ.

ГЛАВА 12. ГРАФИКА В DELPHI

Delphi позволяет программисту разрабатывать программы, которые могут выводить графику: схемы, чертежи, иллюстрации.

Программа выводит графику на поверхность объекта (формы или компонента Image). Поверхности объекта соответствует свойство Canvas. Для того чтобы вывести на поверхность объекта графический элемент (прямую линию, окружность, прямоугольник и т. д.), необходимо применить к свойству canvas этого объекта соответствующий метод. Например, инструкция Form1.Canvas.Rectangle (10,10,100,100) вычерчивает в окне программы прямоугольник.

Холст

Как было сказано ранее, поверхности, на которую программа может выводить графику, соответствует свойство Canvas. В свою очередь, свойство Canvas – это объект типа TCanvas. Методы этого типа обеспечивают вывод графических примитивов (точек, линий, окружностей, прямоугольников и т. д.), а свойства позволяют задать характеристики выводимых графических примитивов: цвет, толщину и стиль линий; цвет и вид заполнения областей; характеристики шрифта при выводе текстовой информации.

Методы вывода графических примитивов рассматривают свойство Canvas как некоторый абстрактный холст, на котором они могут рисовать (canvas переводится как "поверхность", "холст для рисования"). Холст состоит из отдельных точек – пикселов. Положение пиксела характеризуется его горизонтальной (X) и вертикальной (Y) координатами. Левый верхний пиксел имеет координаты (0, 0). Координаты возрастают сверху вниз и слева направо (рис. 104). Значения координат правой нижней точки холста зависят от размера холста.

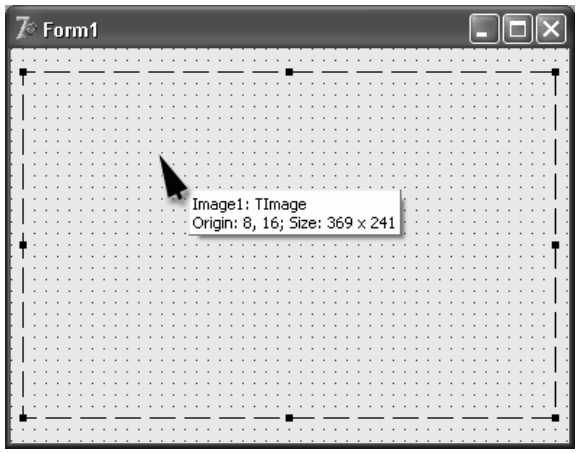


Рис. 104. Координаты точек холста

Размер холста можно получить, обратившись к свойствам Height и Width **области иллюстрации** (Image) или к **свойствам формы:** ClientHeight и ClientWidth.

Карандаш и кисть

Художник в своей работе использует карандаши и кисти. Методы, обеспечивающие вычерчивание на поверхности холста графических примитивов, тоже используют карандаш и кисть. Карандаш применяется для вычерчивания линий и контуров, а кисть – для закрашивания областей, ограниченных контурами.

Карандашу и кисти, используемым для вывода графики на холсте, соответствуют два свойства объекта Canvas: Pen (карандаш) и Brush (кисть), которые представляют собой объекты типа TPen и TBrush, соответственно. Значения свойств этих объектов определяют вид выводимых графических элементов.

Карандаш используется для вычерчивания точек, линий, контуров геометрических фигур: прямоугольников, окружностей, эллипсов, дуг и др. Вид линии, которую оставляет карандаш на поверхности холста, определяют свойства объекта треп, которые перечислены в табл. 7.

Свойство Color задает цвет линии, вычерчиваемой карандашом. В табл. 8 перечислены именованные константы (тип TColor), которые можно использовать в качестве значения свойства Color.

Таблица 7

Свойства объекта TPen (карандаш)

Свойство	**Определяет**
Color	Цвет линии
Width	Толщину линии
Style	Вид линии
Mode	Режим отображения

Свойство Width задает толщину линии (в пикселах). Например, инструкция Canvas.Pen.Width: =2 устанавливает толщину линии в 2 пиксела.

Свойство Style определяет вид (стиль) линии, которая может быть непрерывной или прерывистой, состоящей из штрихов различной длины. В табл. 9 перечислены именованные константы, позволяющие задать стиль линии. Толщина пунктирной линии не может быть больше 1. Если значение свойства Pen.Width больше единицы, то пунктирная линия будет выведена как сплошная.

Таблица 8

Значение свойства Color определяет цвет линии

Константа	Цвет	Константа	Цвет
clBlack	Черный	clSilver	Серебристый
clMaroon	Каштановый	clRed	Красный
clGreen	Зеленый	clLime	Салатный
clOlive	Оливковый	clBlue	Синий
clNavy	Темно-синий	clFuchsia	Ярко-розовый
clPurple	Розовый	clAqua	Бирюзовый
clTeal	Зелено-голубой	clWhite	Белый
clGray	Серый		

Таблица 9

Значение свойства Pen.Type определяет вид линии

Константа	Вид линии
psSolid	Сплошная линия
psDash	Пунктирная линия, длинные штрихи
psDot	Пунктирная линия, короткие штрихи
psDashDot	Пунктирная линия, чередование длинного и короткого штрихов
psDashDotDot	Пунктирная линия, чередование одного длинного и двух коротких штрихов
psClear	Линия не отображается (используется, если не надо изображать границу области, например, прямоугольника)

Свойство Mode определяет, как будет формироваться цвет точек линии в зависимости от цвета точек холста, через которые эта линия прочерчивается. По умолчанию вся линия вычерчивается цветом, определяемым значением свойства Pen.Color.

Однако программист может задать инверсный цвет линии по отношению к цвету фона. Это гарантирует, что независимо от цвета фона все участки линии будут видны, даже в том случае, если цвет линии и цвет фона совпадают.

В табл. 10 перечислены некоторые константы, которые можно использовать в качестве значения свойства Pen.Mode.

Кисть (Canvas.Brush) используется методами, обеспечивающими вычерчивание замкнутых областей, например геометрических фигур, для заливки (закрашивания) этих областей. Кисть, как объект, обладает двумя свойствами, перечисленными в табл. 11.

Таблица 10

Значение свойства Pen.Mode влияет на цвет линии

Константа	Цвет линии
pmBlack	Черный, не зависит от значения свойства Pen. Color
pmWhite	Белый, не зависит от значения свойства Pen. Color
pmCopy	Цвет линии определяется значением свойства Pen . Color
pmNotCopy	Цвет линии является инверсным по отношению к значению свойства Pen. Color
pmNot	Цвет точки линии определяется как инверсный по отношению к цвету точки холста, в которую выводится точка линии

Таблица 11

Свойства объекта TBrush (кисть)

Свойство	Определяет
Color	Цвет закрашивания замкнутой области
Style	Стиль (тип) заполнения области

Область внутри контура может быть закрашена или заштрихована. В первом случае область полностью перекрывает фон, а во втором − сквозь незаштрихованные участки области будет виден фон.

В качестве значения свойства Color можно использовать любую из констант типа TColor (см. список констант для свойства Pen.Color в табл. 8).

Константы, позволяющие задать стиль заполнения области, приведены в табл. 12.

Таблица 12

Значения свойства Brush.Style определяют тип закрашивания

Константа	Тип заполнения (заливки) области
bsSolid	Сплошная заливка
bsClear	Область не закрашивается
bsHorizontal	Горизонтальная штриховка
bsVertical	Вертикальная штриховка
bsFDiagonal	Диагональная штриховка с наклоном линий вперед
bsBDiagonal	Диагональная штриховка с наклоном линий назад
bsCross	Горизонтально-вертикальная штриховка, в клетку
bsDiagCross	Диагональная штриховка, в клетку

В качестве примера приведен модуль brustyle программы Стили заполнения областей, которая в окно (рис. 105) выводит восемь прямо-

угольников, закрашенных черным цветом с использованием разных стилей.

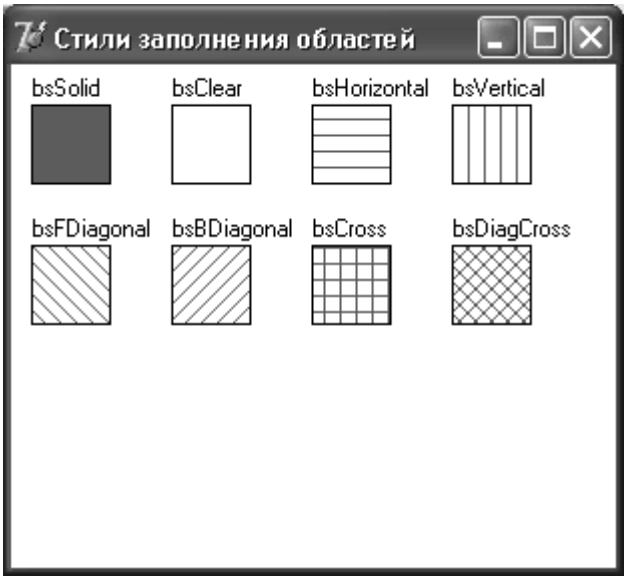


Рис. 105. Окно программы Стили заполнения областей

Пример программы:

```
unit brustyle;
interface
  uses
    Windows, Messages, SysUtils, Classes,
    Graphics, Controls, Forms, Dialogs, ExtCtrls;
  type
    TForm1 = class(TForm)
      procedure FormPaint(Sender: TObject);
    private
    { Private declarations}
    public
    { Public declarations )
    end;
  var
    Form1: TForm1;
implementation
{$R *.DFM}
  // перерисовка формы
  procedure TForm1.FormPaint(Sender: TObject);
  const
  bsName: array[1..8] of string =
  ('bsSolid','bsClear','bsHorizontal',
  'bsVertical','bsFDiagonal','bsBDiagonal',
  'bsCross','bsDiagCross');
var
```

```
  x,y: integer; // координаты левого верхнего угла
   // прямоугольника
  w,h: integer; // ширина и высота прямоугольника
  bs: TBrushStyle;// стиль заполнения области
  k: integer; // номер стиля заполнения
  i,j: integer;
begin
  w:=40; h:=40; // размер области(прямоугольника)
  y:=20;
  for i:=l to 2 do
  begin
    x:=10;
    for j:=1 to 4 do
    begin
      k:=j+(i-1)*4; // номер стиля заполнения
      case k of
        1: bs = bsSolid;
        2: bs = bsClear;
        3: bs = bsHorizontal;
        4: bs = bsVertical;
        5: bs = bsFDiagonal;
        6: bs = bsBDiagonal;
        7: bs = bsCross;
        8: bs = bsDiagCross;
      end;
      // вывод прямоугольника
      // цвет закрашивания - зеленый
      Canvas.Brush.Color := clGreen;
      // стиль закрашивания
      Canvas.Brush.Style := bs;
      Canvas.Rectangle (x, y, x+w, y-t-h) ;
      Canvas.Brush.Style := bsClear;
      // вывод названия стиля
      Canvas.TextOut(x, y-15, bsName[k]);
      x := x+w+30;
    end;
    y := y+h+30;
  end;
end;
end.
```

Вывод текста

Для вывода текста на поверхность графического объекта используется метод TextOut. Инструкция вызова метода TextOut в общем виде выглядит следующим образом:

```
Объект.Canvas.TextOut(x, y, Текст)
```

где

- объект − имя объекта, на поверхность которого выводится текст;
- x, y − координаты точки графической поверхности, от которой выполняется вывод текста (рис. 106);
- Текст − переменная или константа символьного типа, значение которой определяет выводимый методом текст.

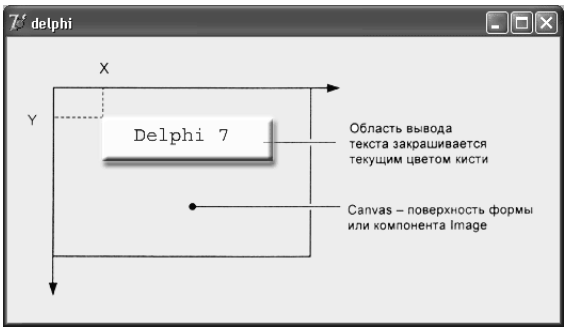


Рис. 106. Координаты области вывода текста

Шрифт, который используется для вывода текста, определяется значением свойства Font соответствующего объекта Canvas. Свойство Font представляет собой объект типа TFont. В табл. 13 перечислены свойства объекта TFont, позволяющие задать характеристики шрифта, используемого методами TextOut и TextRect для вывода текста.

Таблица 13

Свойства объекта TFont

Свойство	**Определяет**
Name	Используемый шрифт. В качестве значения следует использовать название шрифта, например Arial
Size	Размер шрифта в пунктах (points). Пункт− это единица измерения размера шрифта, используемая в полиграфии. Один пункт равен 1/72 дюйма
Style	Стиль начертания символов. Может быть: нормальным, полужирным, курсивным, подчеркнутым, перечеркнутым. Стиль задается при помощи следующих констант: fsBold (полужирный), fsItalic (курсив), fsUnderline (подчеркнутый), fsStrikeOut (перечеркнутый). Свойство Style является множеством, что позволяет комбинировать необходимые стили. Например, инструкция программы, устанавливающая стиль "полужирный курсив", выглядит так: `Объект.Canvas.Font := [fsBold, fsItalic]`
Color	Цвет символов. В качестве значения можно использовать константу типа TColor

Внимание! *Область вывода текста закрашивается текущим цветом кисти. Поэтому перед выводом текста свойству Brush.Color нужно присвоить значение bsClear или задать цвет кисти, совпадающий с цветом поверхности, на которую выводится текст.*

Следующий фрагмент программы демонстрирует использование функции TextOut для вывода текста на поверхность формы:

```
with Form1.Canvas do
begin
// установить характеристики шрифта
  Font.Name := 'Tahoma';
  Font.Size := 20;
  Font.Style := [fsltalic, fsBold] ;
  // область вывода текста не закрашивается
  Brush.Style := bsClear;
  TextOut(0, 10, 'Borland Delphi 7');
end;
```

После вывода текста методом Textout указатель вывода (карандаш) перемещается в правый верхний угол области вывода текста.

Иногда требуется вывести какой-либо текст после сообщения, длина которого во время разработки программы неизвестна. Например, это может быть слово "руб." после значения числа, записанного прописью. В этом случае необходимо знать координаты правой границы уже выведенного текста. Координаты правой границы текста, выведенного методом Textout, можно получить, обратившись к свойству PenPos.

Следующий фрагмент программы демонстрирует возможность вывода строки текста при помощи двух инструкций TextOut.

```
with Form1.Canvas do
begin
  TextOut(0, 10, 'Borland ') ;
  TextOut(PenPos.X, PenPos.Y, 'Delphi 7');
end;
```

Методы вычерчивания графических примитивов

Любая картинка, чертеж, схема могут рассматриваться как совокупность графических примитивов: точек, линий, окружностей, дуг и др. Таким образом, для того чтобы на экране появилась нужная картинка, программа должна обеспечить вычерчивание (вывод) графических примитивов, составляющих эту картинку.

Вычерчивание графических примитивов на поверхности компонента (формы или области вывода иллюстрации) осуществляется применением соответствующих методов к свойству Canvas этого компонента.

Линия

Вычерчивание прямой линии осуществляет метод LineTo, инструкция вызова которого в общем виде выглядит следующим образом:

```
Компонент.Canvas.LineTo(x,y)
```

Метод LineTo вычерчивает прямую линию от текущей позиции карандаша в точку с координатами, указанными при вызове метода.

Начальную точку линии можно задать, переместив карандаш в нужную точку графической поверхности. Сделать это можно при помощи метода MoveTo, указав в качестве параметров координаты нового положения карандаша.

Вид линии (цвет, толщина и стиль) определяется значениями свойств объекта Реп графической поверхности, на которой вычерчивается линия.

Довольно часто результаты расчетов удобно представить в виде графика. Для большей информативности и наглядности графики изображают на фоне координатных осей и оцифрованной сетки. В следующем примере приведен текст модуля программы, который обеспечивает вывод координатных осей и оцифрованной сетки на поверхность формы (рис. 107).

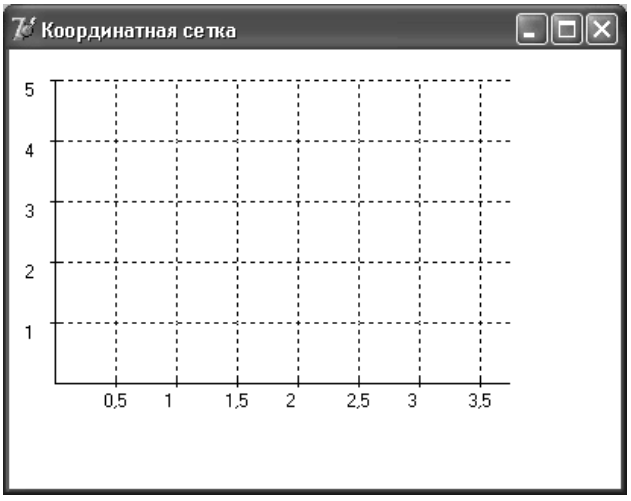


Рис. 107. Форма приложения Координатная сетка

Текст модуля:

```
unit grid_;

interface

uses
  Windows, Messages, SysUtils, Classes,
  Graphics, Controls, Forms, Dialogs, StdCtrls;
```

```
type
  TForm1 = class(TForm)
    procedure FormPaint(Sender: TObject);
  private
  { Private declarations }
  public
  { Public declarations }
  end;
var
  Form1: TForm1;

implementation
{$R *.DFM}

procedure TForm1.FormPaint(Sender: TObject);
var
  x0,y0:integer; // координаты начала координатных осей
  dx,dy:integer; // шаг координатной сетки (в пикселах)
  h,w:integer; // высота и ширина области вывода
               // координатной сетки
  x,y:integer;
  lx,ly:real; // метки (оцифровка) линий сетки по X и Y
  dlx,dly:real; // шаг меток (оцифровки) линий сетки по X и Y
  cross:integer; // счетчик неоцифрованных линий сетки
  dcross:integer;// количество неоцифрованных линий между
  // оцифрованными
begin
  x0:=30;
  y0:=220; // оси начинаются в точке (40,250)
  dx:=40;
  dy:=40; // шар координатной сетки 40 пикселов
  dcross:=1; // помечать линии сетки X: 1 - каждую;
             // 2 - через одну;
             // 3 - через две;
  dlx:=0.5; // шаг меток оси X
  dly:=1.0; // шаг меток оси Y, метками будут: 1, 2, 3 и т. д.
  h:=200;
  w:=300;
  with forml.Canvas do
  begin
    cross:=dcross;
    MoveTo(x0,y0);
    LineTo(x0,y0-h); // ось X
    MoveTo(x0,y0);
    LineTo(x0+w, y0); // ось Y
```

```
      // засечки, сетка и оцифровка по оси X
      x:=x0+dx;
      lx:=dlx;
      repeat
        MoveTo(x,y0-3);LineTo(x,y0+3); // засечка
        cross:=cross-1;
        if cross = 0 then // оцифровка
        begin
          TextOut(x-8,y0+5,FloatToStr(lx));
          cross:=dcross ;
        end;
        Pen.Style:=psDot;
        MoveTo(x,y0-3);
        LineTo(x,y0-h); // линия сетки
        Pen.Style:=psSolid;
        lx:=lx+dlx;
        x:=x+dx;
      until (x>x0+w);
      // засечки, сетка и оцифровка по оси Y
      y:=y0-dy;
      ly:=dly;
      repeat
        MoveTo(x0-3,y);LineTo(x0+3,y); // засечка
        TextOut(x0-20,y,FloatToStr(ly)); // оцифровка
        Pen.Style:=psDot;
        MoveTo(x0+3,y);
        LineTo(x0+w,y); // линия сетки
        Pen.Style:=psSolid;
        y:=y-dy;
        ly:=ly+dly;
      until (y<y0-h);
      end;
    end;
  end.
```

Особенность приведенной программы заключается в том, что она позволяет задавать шаг сетки и оцифровку. Кроме того, программа дает возможность оцифровывать не каждую линию сетки оси x, а через одну, две, три и т. д. Сделано это для того, чтобы предотвратить возможные наложения изображений чисел оцифровки друг на друга в случае, если эти числа состоят из нескольких цифр.

Ломаная линия

Метод PolyLine вычерчивает ломаную линию. В качестве параметра метод получает массив типа TPoint. Каждый элемент массива

представляет собой запись, поля x и y которой содержат координаты точки перегиба ломаной. Метод Polyline вычерчивает ломаную линию, последовательно соединяя прямыми точки, координаты которых находятся в массиве: первую со второй, вторую с третьей, третью с четвертой и т. д.

В качестве примера использования метода Polyline приведен текст процедуры, которая выводит график изменения некоторой величины. Предполагается, что исходные данные находятся в доступном процедуре массиве Data (тип Integer).

```
procedure TForml.Button1Click(Sender: TObject);
var
  gr: array[1..50] of TPoint; //график-ломаная линия
  x0,y0: integer; //координаты точки начала координат
  dx,dy: integer;//шаг координатной сетки по осям X и Y
  i: integer;
begin
  x0 := 10;
  y0 := 200;
  dx :=5;
  dy := 5;
  // заполним массив gr
  for i:=l to 50 do
  begin
    gr[i].x := x0 + (i-l)*dx;
    gr[i].y := y0 - Data[i]*dy;
  end;
  // строим график
  with forml.Canvas do
  begin
    MoveTo(x0,y0); LineTo(x0,10); // ось Y
    MoveTo(x0,y0); LineTo(200,y0); // ось X
    Polyline(gr); // график
  end;
end;
```

Метод Polyline можно использовать для вычерчивания замкнутых контуров. Для этого надо, чтобы первый и последний элементы массива содержали координаты одной и той же точки. В качестве примера использования метода Polybine для вычерчивания замкнутого контура приведен текст программы, которая на поверхности диалогового окна, в точке нажатия кнопки мыши, вычерчивает контур пятиконечной звезды (рис. 15). Цвет, которым вычерчивается звезда, зависит от того, какая из кнопок мыши была нажата. Процедура обработки нажатия кнопки мыши (событие MouseDown) вызывает процедуру рисования звезды

StarLine и передает ей в качестве параметра координаты точки, в которой была нажата кнопка. Звезду вычерчивает процедура StarLine, которая в качестве параметров получает координаты центра звезды и холст, на котором звезда должна быть выведена. Сначала вычисляются координаты концов и впадин звезды, которые записываются в массив p. Затем этот массив передается в качестве параметра методу Polyline. При вычислении координат лучей и впадин звезды используются функции sin и cos. Так как аргумент этих функций должен быть выражен в радианах, то значение угла в градусах домножается на величину pi/180, где pi – это стандартная именованная константа равная числу π.

Пример: Вычерчивание замкнутого контура (звезды) в точке нажатия кнопки мыши (рис. 108).

```
unit Stars_;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes,
  Graphics, Controls, Forms, Dialogs, StdCtrls;

type
  TForm1 = class(TForm)
  procedure FormMouseDown(Sender: TObject;
                          Button: TMouseButton;
                          Shift: TShiftState;
                          X, Y: Integer);
 private
    { Private declarations }
public
   { Public declarations }
end;
var
   Forml: TForml;

implementation
{$R *.dfm}

// вычерчивает звезду
procedure StarLine(x0,y0,r: integer; Canvas: TCanvas);
// x0,y0 - координаты центра звезды
//r - радиус звезды
var
  // массив координат лучей и впадин
```

```
  p : array [1.. 11] of TPoint;
  a: integer; // угол между осью OX и прямой, соединяющей
              // центр звезды и конец луча или впадину
  i: integer;
begin
  a := 18; // строим от правого гор. луча
  for i:=1 to 10 do
  begin
    if (i mod 2=0) then
    begin // впадина
      p[i].x := x0+Round(r/2*cos(a*pi/180));
      p[i].y := y0-Round(r/2*sin(a*pi/180));
    end
    else
    begin // луч
      p[i].x := x0+Round(r*cos(a*pi/180)) ;
      p[i].y := y0-Round(r*sin(a*pi/180)) ;
    end;
    a := a+36;
  end;
  p[ll].x := p[l].x; // чтобы замкнуть контур звезды
  Canvas.Polyline(p) ; // начертить звезду
end;

  // нажатие кнопки мыши
procedure TForm1.FormMouseDown(Sender : TObject;
                               Button: TMouseButton;
                               Shift: TShiftState;
                               X, Y: Integer);
begin
  if Button = mbLeft then // нажата левая кнопка?
    Form1.Canvas.Pen.Color : = clRed
  else
    Form1. Canvas. Pen. Color := clGreen;
    StarLine(x, y, 30, Forml.Canvas);
end;
end.
```

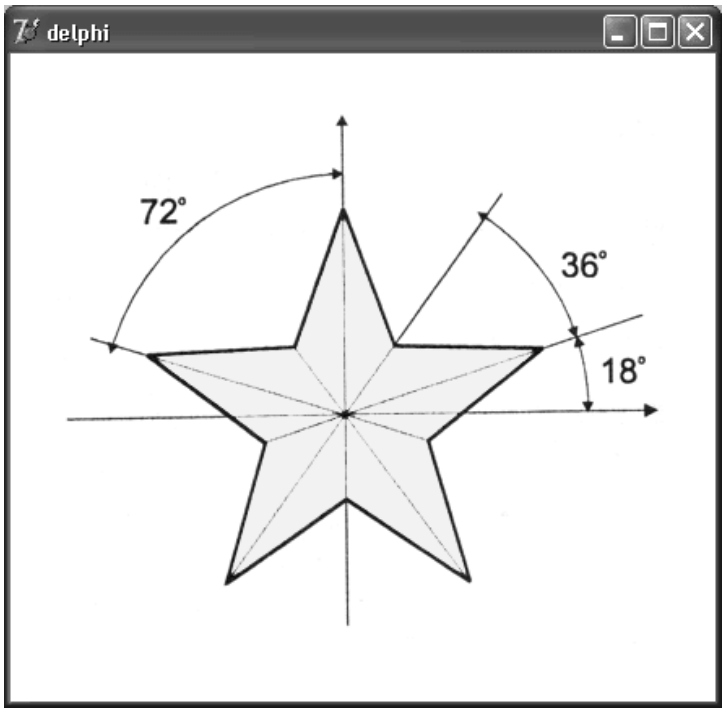


Рис. 108. Звезда

Примечание

Обратите внимание, что размер массива p на единицу больше, чем количество концов и впадин звезды, и что значения первого и последнего элементов массива совпадают.

Окружность и эллипс

Метод Ellipse вычерчивает эллипс или окружность, в зависимости от значений параметров. Инструкция вызова метода в общем виде выглядит следующим образом:

```
Объект.Canvas.Ellipse(x1,y1, x2,y2]
```

где:

- объект – имя объекта (компонента), на поверхности которого выполняется вычерчивание;
- x1, y1, x2, y2 – координаты прямоугольника, внутри которого вычерчивается эллипс или, если прямоугольник является квадратом, окружность (рис. 109).

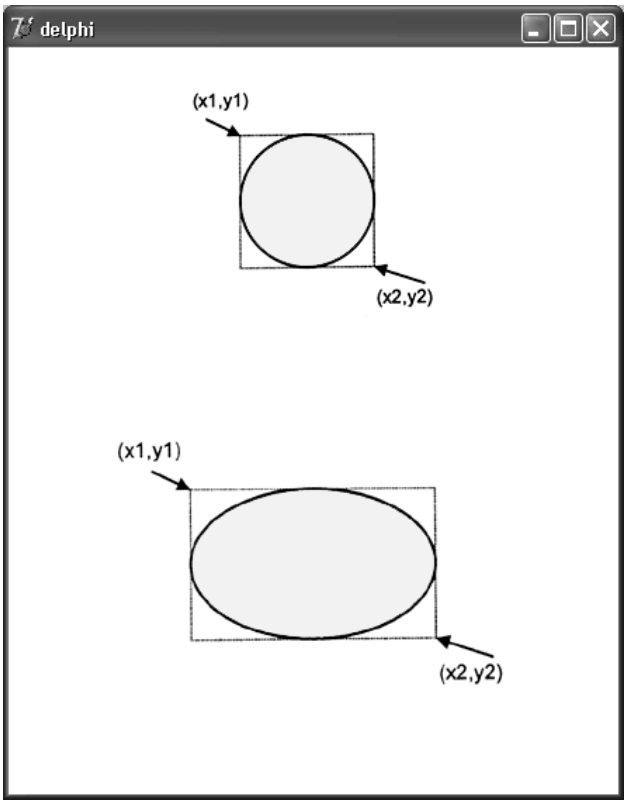


Рис. 109. Значения параметров метода Ellipse определяют вид геометрической фигуры

Цвет, толщина и стиль линии эллипса определяются значениями свойства Pen, а цвет и стиль заливки области внутри эллипса − значениями свойства Brush поверхности (Canvas), на которую выполняется вывод.

Дуга

Вычерчивание дуги выполняет метод Arc, инструкция вызова которого в общем виде выглядит следующим образом:

```
Объект.Canvas.Arc(x1,y1,x2,y2,x3,y3,x4,y4)
```

где:

- x1, y1, x2, y2 – параметры, определяющие эллипс (окружность), частью которого является вычерчиваемая дуга;
- x3, y3 – параметры, определяющие начальную точку дуги;
- x4, y4 – параметры, определяющие конечную точку дуги.

Начальная (конечная) точка – это точка пересечения границы эллипса и прямой, проведенной из центра эллипса в точку с координатами x3 и y3 (x4, y4). Дуга вычерчивается против часовой стрелки от начальной точки к конечной (рис. 110).

Цвет, толщина и стиль линии, которой вычерчивается дуга, определяются значениями свойства Pen поверхности (Canvas), на которую выполняется вывод.

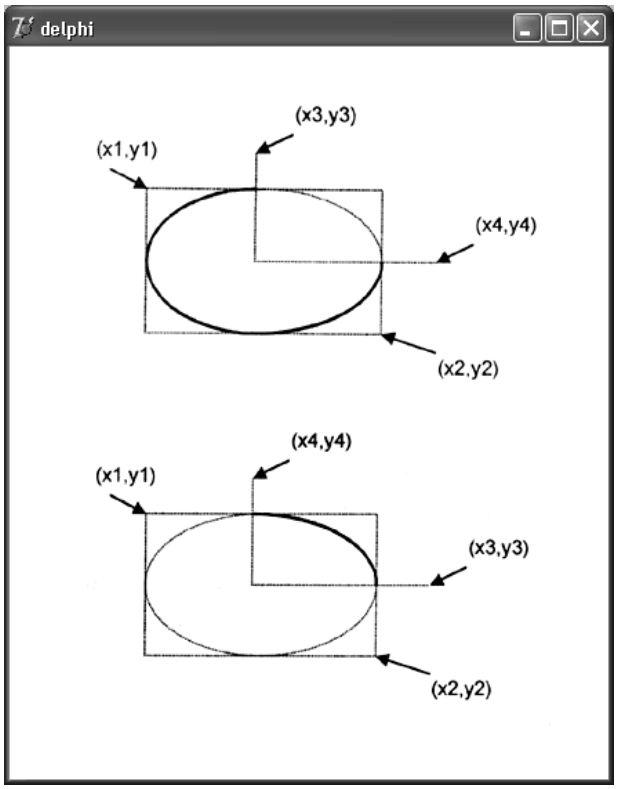


Рис. 110. Значения параметров метода Arc определяют дугу как часть эллипса (окружности)

Прямоугольник

Прямоугольник вычерчивается методом Rectangle, инструкция вызова которого в общем виде выглядит следующим образом:

```
Объект.Canvas.Rectangle(x1, y1,x2, y2)
```

где:

- объект – имя объекта (компонента), на поверхности которого выполняется вычерчивание;
- x1, y1 и x2, y2 – координаты левого верхнего и правого нижнего углов прямоугольника.

Метод RoundRec тоже вычерчивает прямоугольник, но со скругленными углами. Инструкция вызова метода RoundRec выглядит так:

```
Объект.Canvas.RoundRec(x1,y1,x2, y2, x3, y3)
```

где:

- x1, y1, x2, y2 – параметры, определяющие положение углов прямоугольника, в который вписывается прямоугольник со скругленными углами;
- x3 и y3 – размер эллипса, одна четверть которого используется для вычерчивания скругленного угла (рис. 111).

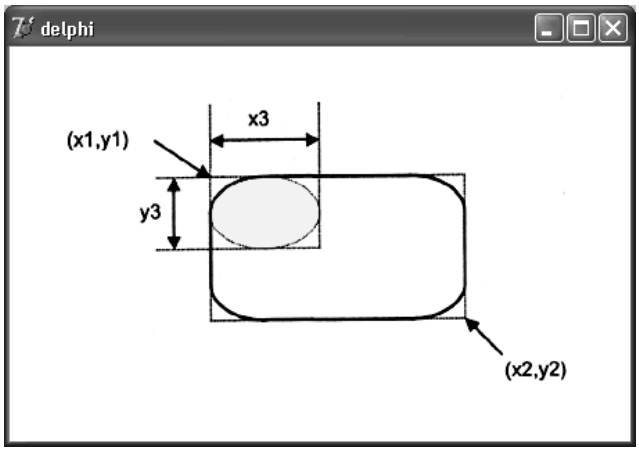


Рис. 111. Метод RoundRec вычерчивает прямоугольник со скругленными углами

Вид линии контура (цвет, ширина и стиль) определяется значениями свойства Реп, а цвет и стиль заливки области внутри прямоугольника − значениями свойства Brush поверхности (canvas), на которой прямоугольник вычерчивается.

Есть еще два метода, которые вычерчивают прямоугольник, используя в качестве инструмента только кисть (Brush). Метод FillRect вычерчивает закрашенный прямоугольник, а метод FrameRect − только контур. У каждого из этих методов лишь один параметр − структура типа TRect. Поля структуры TRect содержат координаты прямоугольной области, они могут быть заполнены при помощи функции Rect.

Ниже в качестве примера использования методов FillRect и FrameRect приведена процедура, которая на поверхности формы вычерчивает прямоугольник с красной заливкой и прямоугольник с зеленым контуром.

```
procedure TForm1.Button1Click(Sender: TObject);
var
  r1, r2: TRect; // координаты углов прямоугольников
begin
  // заполнение полей структуры
  // зададим координаты углов прямоугольников
  r1 := Rect(20,20,60,40);
  r2 := Rect(10,10,40,50);
  with form1.Canvas do
  begin
    Brush.Color := clRed;
    FillRect(r1); // закрашенный прямоугольник
    Brush.Color := clGreen;
    FrameRect(r2}; // только граница прямоугольника
  end;
end;
```

Многоугольник

Метод Polygon вычерчивает многоугольник. В качестве параметра метод получает массив типа TPoint. Каждый элемент массива представляет собой запись, поля (x,y) которой содержат координаты одной вершины многоугольника. Метод Polygon вычерчивает многоугольник, последовательно соединяя прямыми линиями точки, координаты которых находятся в массиве: первую со второй, вторую с третьей, третью с четвертой и т. д. Затем соединяются последняя и первая точки.

Цвет и стиль границы многоугольника определяются значениями свойства Pen, а цвет и стиль заливки области, ограниченной линией границы, − значениями свойства Brush, причем область закрашивается с использованием текущего цвета и стиля кисти.

Ниже приведена процедура, которая, используя метод Polygon, вычерчивает треугольник:

```
procedure TForm1.Button2Click(Sender: TObject);
var
  pol: array[1..3] of TPoint; // координаты точек треугольника
begin
  pol[1].x := 10;
  polf1].y := 50;
  pol[2].x := 40;
  pol[2].y := 10;
  pol[3].x := 70;
  pol[3].y := 50;
  Form1.Canvas.Polygon(pol);
end;
```

Сектор

Метод pie вычерчивает сектор эллипса или круга. Инструкция вызова метода в общем виде выглядит следующим образом:

```
Объект. Canvas.Pie(x1,y1,x2,y2,x3,y3,x4,y4)
```

где:

x1, y1, x2, y2 − параметры, определяющие эллипс (окружность), частью которого является сектор;

x3, y3, x4, y4 − параметры, определяющие координаты конечных точек прямых, являющихся границами сектора.

Начальные точки прямых совпадают с центром эллипса (окружности). Сектор вырезается против часовой стрелки от прямой, заданной точкой с координатами (x3, уз), к прямой, заданной точкой с координатами (x4, y4) (рис. 112).

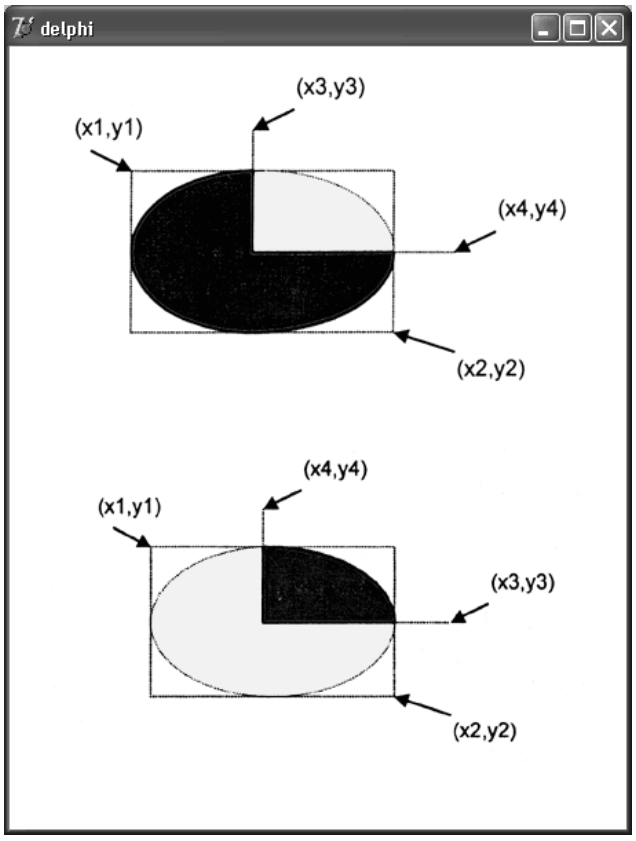


Рис. 112. Значения параметров метода Pie определяют сектор как часть эллипса (окружности)

Точка

Поверхности, на которую программа может осуществлять вывод графики, соответствует объект Canvas. Свойство pixels, представляющее собой двумерный массив типа TColor, содержит информацию о цвете каждой точки графической поверхности. Используя свойство Pixels, можно задать требуемый цвет для любой точки графической поверхности, т. е. "нарисовать" точку. Например, инструкция

```
Form1.Canvas.Pixels[10,10]:=clRed
```

окрашивает точку поверхности формы в красный цвет.

Размерность массива Pixels определяется размером графической поверхности. Размер графической поверхности формы (рабочей области, которую также называют клиентской) задается значениями свойств ClientWidth и ClientHeight, а размер графической поверхности компонента Image − значениями свойств Width и Height. Левой верхней точке рабочей области формы соответствует элемент Pixels [0, 0], а правой нижней – Pixels[ClientWidth − 1, ClientHeight − 1].

Свойство Pixels можно использовать для построения графиков. График строится, как правило, на основе вычислений по формуле. Гра-

ницы диапазона изменения аргумента функции являются исходными данными. Диапазон изменения значения функции может быть вычислен. На основании этих данных можно вычислить масштаб, позволяющий построить график таким образом, чтобы он занимал всю область формы, предназначенную для вывода графика.

Например, если некоторая функция f(x) может принимать значения от нуля до 1000, и для вывода ее графика используется область формы высотой в 250 пикселов, то масштаб оси Y вычисляется по формуле: m = 250/1000. Таким образом, значению f(x) = 70 будет соответствовать точка с координатой Y =233. Значение координаты Y вычислено по формуле

$$Y= h - f(x) \times m = 250 - 70\times (250/1000),$$

где h − высота области построения графика.

Обратите внимание на то, что точное значение выражения 250 − 70× (250/1000) равно 232,5. Но т. к. индексом свойства Pixels, которое используется для вывода точки на поверхность Canvas, может быть только целое значение, то число 232,5 округляется к ближайшему целому, которым является число 233.

Следующая программа, текст которой приведен ниже, используя свойство Pixels, выводит график функции $y = 2 \sin(x) e^{x/5}$. Для построения графика используется вся доступная область формы, причем если во время работы программы пользователь изменит размер окна, то график будет выведен заново с учетом реальных размеров окна.

Пример программы вывода графика функции:

```
unit grfunc_;

interface

uses
  Windows, Messages, SysUtils, Classes,
  Graphics, Controls, Forms, Dialogs;
type
  TForm1 = class(TForm)
     procedure FormPaint(Sender: TObject);
     procedure FormResize(Sender: TObject);
  private
  { Private declarations }
  public
  {Public declarations }
  end;
var
  Forml: TForml;
```

```
implementation
{$R *.DFM}

// Функция, график которой надо построить
function f(x:real):real;
begin
   f:=2*sin(x)*exp(x/5) ;
end;
// строит график функции
procedure GrOfFunc;
var
  x1,x2:real; // границы изменения аргумента функции
  y1,y2:real; // границы изменения значения функции
  x:real; // аргумент функции
  y:real; // значение функции в точке x
  dx:real; // приращение аргумента
  l,b:integer; // левый нижний угол области вывода
  // графика
  w,h:integer; // ширина и высота области вывода графика
  mx,my:real; // масштаб по осям X и Y
  x0,y0:integer; // точка – начало координат
begin
  // область вывода графика
  l:=10; // X – координата левого верхнего угла
  b:=Forml.ClientHeight-20;
  //У – координата левого верхнего угла
  h:=Forml.ClientHeight-40; // высота
  w:=Forml.Width-40; // ширина
  x1:=0; // нижняя граница диапазона аргумента
  x2:=25; // верхняя граница диапазона аргумента
  dx:=0.01; // шаг аргумента
  // найдем максимальное и минимальное значения
  // функции на отрезке [x1,x2]
  y1:=f(xl); // минимум
  y2:=f(xl); //максимум
  x:=x1;
repeat
  y := f (x);
  if y < yl then yl:=y;
  if y > y2 then y2:=y;
  x:=x+dx;
  until (x >= x2);
```

```
// вычислим масштаб
my:=h/abs(y2-yl); // масштаб по оси Y
mx:=w/abs(x2-xl); // масштаб по оси X
x0:=1;
y0:=b-Abs(Round(y1*my));
with Form1.Canvas do
begin
  // оси
  MoveTo(l,b);LineTo(l,b-h);
  MoveTo(x0,y0);LineTo(x0+w,y0);
  TextOut(l+5,b-h,FloatToStrF(y2,ffGeneral,6,3));
  TextOut(l+5,b,FloatToStrF(yl,ffGeneral,6,3));
  // построение графика
  x:=xl;
  repeat
    y:=f(x);
    Pixels[x0+Round(x*mx), y0-Round(y*my)]:=clRed;
    x:=x+dx;
  until (x >= x2);
end;
end;

procedure TForm1.FormPaint(Sender: TObject);
begin
  GrOfFunc;
end;

// изменился размер окна программы
procedure TForm1.FormResize(Sender: TObject);
begin
  // очистить форму
  Forml.Canvas.FillRect(Rect(0,0,ClientWidth,
                             ClientHeight));
  // построить график
  GrOfFunc;
end;

end.
```

Основную работу выполняет процедура GrOfFunc, которая сначала вычисляет максимальное (y2) и минимальное (yl) значения функции на отрезке [x1l,x2]. Затем, используя информацию о ширине (Forml.Clientwidth − 40) и высоте (Form1.ClientHeight − 40) области вывода графика, вычисляет масштаб по осям X (mx) иY(my).

Высота и ширина области вывода графика определяется размерами рабочей (клиентской) области формы, т. е. без учета области заголовка и границ. После вычисления масштаба процедура вычисляет координату y горизонтальной оси (yo) и вычерчивает координатные оси графика. Затем выполняется непосредственное построение графика (рис. 113).

Вызов процедуры GrOfFunc выполняют процедуры обработки событий onPaint и onFormResize. Процедура TForm1.FormPaint обеспечивает вычерчивание графика после появления формы на экране в результате запуска программы, а также после появления формы во время работы программы, например, в результате удаления или перемещения других окон, полностью или частично перекрывающих окно программы. Процедура TForm1.FormResize обеспечивает вычерчивание графика после изменения размера формы.

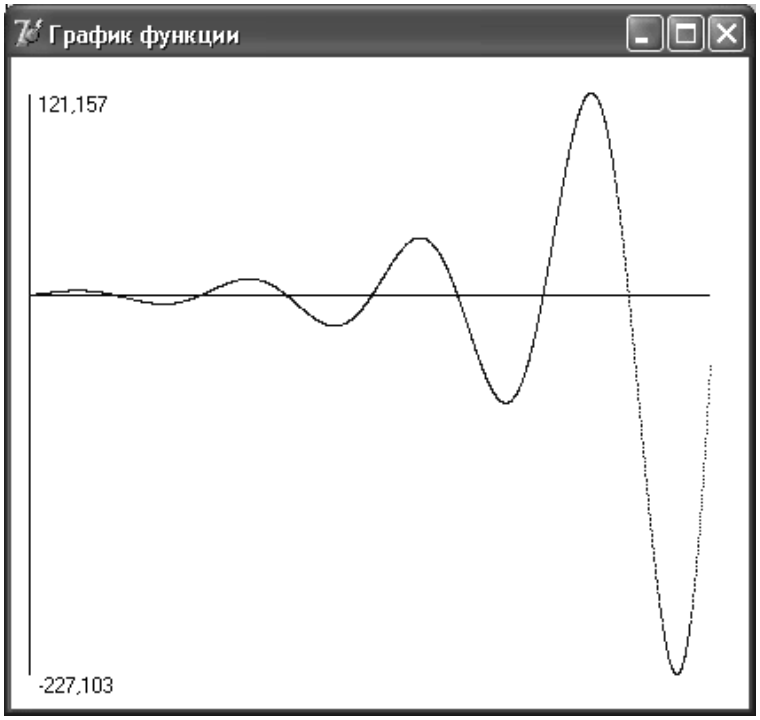


Рис. 113. График, построенный процедурой GrOfFunc

Приведенная программа довольно универсальна. Заменив инструкции в теле функции f (x), можно получить график другой функции. Причем независимо от вида функции ее график будет занимать всю область, предназначенную для вывода.

Примечание

Рассмотренная программа работает корректно, если функция, график которой надо построить, принимает как положительные, так и отрицательные значения. Если функция во всем диапазоне только положительная или только отрицательная, то в программу следует внести изме-

нения. Какие необходимы изменения, можно рассмотреть в качестве самостоятельного упражнения.

Вывод иллюстраций

Наиболее просто вывести иллюстрацию, которая находится в файле с расширением bmp, jpg или ico, можно при помощи компонента Image, значок которого находится на вкладке **Additional** палитры компонентов.

В табл. 14 перечислены основные свойства компонента Image.

Таблица 14

Свойства компонента image

Свойство	Определяет
Picture	Иллюстрацию, которая отображается в поле компонента
Width, Height	Размер компонента. Если размер компонента меньше размера иллюстрации, и значение свойств AutoSize и strech равно False, то отображается часть иллюстрации
AutoSize	Признак автоматического изменения размера компонента в соответствии с реальным размером иллюстрации
Strech	Признак автоматического масштабирования иллюстрации в соответствии с реальным размером компонента. Чтобы было выполнено масштабирование, значение свойства AutoSize должно быть False
Visible	Отображается ли компонент, и, соответственно, иллюстрация, на поверхности формы

Иллюстрацию, которая будет выведена в поле компонента Image, можно задать как во время разработки формы приложения, так и во время работы программы.

Во время разработки формы иллюстрация задается установкой значения свойства Picture путем выбора файла иллюстрации в стандартном диалоговом окне, которое появляется в результате щелчка на командной кнопке **Load** окна **Picture Editor** (рис. 114). Чтобы запустить Image Editor, нужно в окне **Object Inspector** выбрать свойство Picture и щелкнуть на кнопке с тремя точками.

Если размер иллюстрации больше размера компонента, то свойству strech нужно присвоить значение True и установить значения свойств Width и Height пропорционально реальным размерам иллюстрации.

Чтобы вывести иллюстрацию в поле компонента Image во время работы программы, нужно применить метод LoadFromFile к свойству Picture, указав в качестве параметра имя файла иллюстрации. Например, инструкция

```
Form1.Image1.Picture.LoadFromFile('e:\temp\bart.bmp')
```

загружает иллюстрацию из файла bart.bmp и выводит ее в поле вывода иллюстрации (Imagel).

Метод LoadFromFile позволяет отображать иллюстрации различных графических форматов: BMP, WMF, JPEG (файлы с расширением jpg).

Следующая программа, текст которой приведен ниже, использует компонент Image для просмотра иллюстраций, которые находятся в указанном пользователем каталоге. Диалоговое окно программы приведено на рис. 115.

Рис. 114. Окно Picture Editor

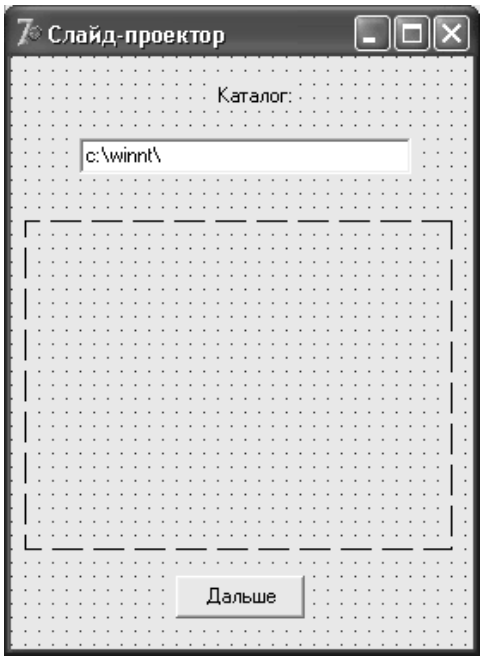


Рис. 115. Слайд-проектор

Пример программы Слайд-проектор:

```
unit shpic_;

interface

uses
  Windows, Messages, SysUtils, Classes,
  Graphics, Controls, Forms,
  Dialogs, ExtCtrls, StdCtrls, Menu

type
  TForm1 = class(TForm) Image1: TImage;
    Button1: TButton;
    procedure FormActivate(Sender: TObject);
    procedure ButtonlClick(Sender: TObject);
  private
  { Private declarations }
  public
  { Public declarations }
  end;

var
  Form1: TForm1;
  aSearchRec : TSearchRec;
  aPath : String; // каталог, в котором находятся иллю-
страции
  aFile : String; // файл иллюстрации
  iw,ih: integer; // первоначальный размер компонента
Image

implementation
{$R *.DFM}

// изменение размера области вывода иллюстрации
// пропорционально размеру иллюстрации
procedure ScaleImage;
var
  pw, ph : integer; // размер иллюстрации
  scaleX, scaleY : real; // масштаб по X и Y
  scale : real; // общий масштаб
begin
  // иллюстрация уже загружена
  // получим ее размеры
  pw := Form1.Image1.Picture.Width;
```

```
    ph := Form1.Image1.Picture.Height;
    // ширина иллюстрации больше ширины компонента Image
    if pw > iw then
      scaleX := iw/pw // нужно масштабировать
    else
      scaleX := 1;
    // высота иллюстрации больше высоты компонента
    if ph > ih then
      scaleY := ih/ph // нужно масштабировать
    else
      scaleY := 1;
    // выберем наименьший коэффициент
    if scaleX < scaleY then
      scale := scaleX
    else
      scale := scaleY;
    // изменим размер области вывода иллюстрации
   Form1.Image1.Height :=
      Round(Form1.Image1.Picture.Height*scale)
   Form1.Image1.Width :=
      Round(Form1.Image1.Picture.Width*scale);
   // т. к. Strech = True и размер области пропорционален
   // размеру картинки, то картинка масштабируется без
   // искажений
end;
// вывести первую иллюстрацию
procedure FirstPicture;
var
  r : integer; // результат поиска файла
begin
  aPath := 'f:\temp\';
  r := FindFirst(aPath+'*.bmp',faAnyFile,aSearchRec);
  if r = 0 then
  begin // в указанном каталоге есть bmp-файл
    aFile := aPath + aSearchRec.Name;
    Form1.Image1.Picture.LoadFromFile(aFile);// загрузить
    // иллюстрацию
    ScaleImage; // установить размер компонента
    r := FindNext(aSearchRec); // найти следующий файл
    if r = 0 then // еще есть файлы иллюстраций
      Forml.Button1.Enabled := True;
  end;
end;
```

```
// вывести следующую иллюстрацию
procedure NextPicture();
var
  r : integer;
begin
  aFile := aPath + aSearchRec.Name;
  Forml.Image1.Picture.LoadFromFile(aFile);
  Scalelmage;
  // подготовим вывод следующей иллюстрации
  r := FindNext(aSearchRec); // найти следующий файл
  if r<>0 then // больше нет иллюстраций
  Forml.Buttonl.Enabled := False;
end;

procedure TForml.FormActivate(Sender: TObject);
begin
  Image1.AutoSize := False;//запрет автоизменения размера
                           // компонента
  Image1.Stretch := True; // разрешим масштабирование
  //запомним первоначальный размер области вывода
  //иллюстрации
  iw := Imagel.Width;
  in := imagel.Height;
  Button1.Enabled := False; // сделаем недоступной кнопку
  // Дальше
  FirstPicture; // вывести первую иллюстрацию
end;
// щелчок на кнопке Дальше
procedure TForm1.Button1Click(Sender: TObject);
begin
  NextPicture;
end;
end.
```

Программа выполняет масштабирование выводимых иллюстраций без искажения, чего нельзя добиться простым присвоением значения True свойству strech. Загрузку и вывод первой и остальных иллюстраций выполняют соответственно процедуры FirstPicture и NextPicture. Процедура FrirstPicture использует функцию FindFirst для того, чтобы получить имя первого BMP-файла. В качестве параметров функции FindFirst передаются:

- имя каталога, в котором должны находиться иллюстрации;
- структура aSearchRec, поле Name которой, в случае успеха, будет содержать имя файла, удовлетворяющего критерию поиска;
- маска файла иллюстрации.

Если в указанном при вызове функции FindFirst каталоге есть хотя бы один BMP-файл, значение функции будет равно нулю. В этом случае метод LoadFromFiie загружает файл иллюстрации, после чего вызывается функция scaieimage, которая устанавливает размер компонента пропорционально размеру иллюстрации. Размер загруженной иллюстрации можно получить, обратившись к свойствам Form1.Image1.Picture.Width и Form1.Image1.Picture.Height, значения которых не зависят от размера компонента Image.

Битовые образы

При работе с графикой удобно использовать объекты типа TBitMap (битовый образ). Битовый образ представляет собой находящуюся в памяти компьютера, и, следовательно, невидимую графическую поверхность, на которой программа может сформировать изображение. Содержимое битового образа (картинка) легко и, что особенно важно, быстро может быть выведено на поверхность формы или области вывода иллюстрации (image). Поэтому в программах битовые образы обычно используются для хранения небольших изображений, например, картинок командных кнопок.

Загрузить в битовый образ нужную картинку можно при помощи метода LoadFromFlie, указав в качестве параметра имя BMP-файла, в котором находится нужная иллюстрация.

Например, если в программе объявлена переменная pic типа TBitMap, то после выполнения инструкции

```
pic.LoadFromFiie('e:\images\aplane.bmp')
```

битовый образ pic будет содержать изображение самолета.

Вывести содержимое битового образа (картинку) на поверхность формы или области вывода иллюстрации можно путем применения метода Draw к соответствующему свойству поверхности (Canvas). Например, инструкция

```
Image1.Canvas.Draw(x,y, bm)
```

выводит картинку битового образа bm на поверхность компонента Image1 (параметры x и y определяют положение левого верхнего угла картинки на поверхности компонента).

Если перед применением метода Draw свойству Transparent объекта TBitMap присвоить значение True, то фрагменты рисунка, окрашенные цветом, совпадающим с цветом левого нижнего угла картинки, не будут выведены – через них будет как бы проглядывать фон. Если в качестве "прозрачного" нужно использовать цвет, отличный от цвета левой ниж-

ней точки рисунка, то свойству Transparentcoior следует присвоить значение символьной константы, обозначающей необходимый цвет.

Следующая программа, текст которой приведен ниже, демонстрирует использование битовых образов для формирования изображения из нескольких элементов.

Пример программы, демонстрирующей использование битовых образов:

```
unit aplanes_;
interface
uses
  Windows, Messages, SysUtils, Classes,
  Graphics, Controls, Forms, Dialogs;
type
  TForml = class(TForm)
    procedure FormPaint(Sender: TObject);
  private
  { Private declarations }
  public
  { Public declarations }
  end;

var
  Forml: TForm1;
  sky, aplane: TBitMap; // битовые образы: небо и самолет

implementation

($R *.DFM}
procedure TForm1.FormPaint(Sender: TObject);
begin
 // создать битовые образы
 sky := TBitMap.Create;
 aplane := TBitMap.Create;

 // загрузить картинки
 sky.LoadFromFile('sky.bmp');
 aplane.LoadFromFile('aplane.bmp') ;
 Form1.Canvas.Draw(0,0,sky); // отрисовка фона

 // отрисовка левого самолета
 Form1.Canvas.Draw(20,20,aplane);
 aplane.Transparent:=True;

 // теперь элементы рисунка, цвет которых совпадает
```

```
  // с цветом левой нижней точки битового образа,
  // не отрисовываются

  // отрисовка правого самолета
  Form1.Canvas.Draw(120,20,aplane);

  // освободить память
  sky.free;
  aplane.free;
end;
end.
```

После запуска программы в окне приложения (рис. 116) появляется изображение летящих на фоне неба самолетов. Фон и изображение самолета – битовые образы, загружаемые из файлов. Белое поле вокруг левого самолета показывает истинный размер картинки битового образа aplane. Белое поле вокруг правого самолета отсутствует, т. к. перед его выводом свойству Transparent битового образа было присвоено значение True.

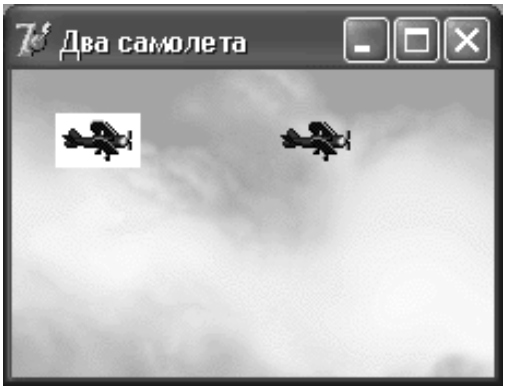


Рис. 116. Влияние значение свойства Transparent на вывод изображения

Мультипликация

Под мультипликацией обычно понимается движущийся и меняющийся рисунок. В простейшем случае рисунок может только двигаться или только меняться.

Как было показано выше, рисунок может быть сформирован из графических примитивов (линий, окружностей, дуг, многоугольников и т. д.). Обеспечить перемещение рисунка довольно просто: надо сначала вывести рисунок на экран, затем через некоторое время стереть его и снова вывести этот же рисунок, но уже на некотором расстоянии от его первоначального положения. Подбором времени между выводом и удалением рисунка, а также расстояния между старым и новым положением рисунка (шага перемещения), можно добиться того, что у наблюда-

теля будет складываться впечатление, что рисунок равномерно движется по экрану.

Следующая простая программа, текст которой приведен далее, а вид формы – на рис. 117, демонстрирует движение окружности от левой к правой границе окна программы.

Рис. 117. Форма программы Движущаяся окружность

Пример программы, демонстрирующей движущуюся окружность:

```
unit mcircle_;

interface

uses
  Windows, Messages, SysUtils, Classes,
  Graphics, Controls, Forms,
  Dialogs, ExtCtrls, StdCtrls;
type
  TForm1 = class(TForm)
    Timer1: TTimer;
    procedure Timer1Timer(Sender: TObject};
    procedure FormActivate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

implementation

{$R *.DFM}
var
  Form1: TForml;
  x,y: byte; // координаты центра окружности
  dx: byte; // приращение координаты x при движении
            // окружности стирает и рисует окружность
            // на новом месте
```

```
procedure Ris;
begin
  // стереть окружность
  Form1.Canvas.Pen.Color:=Form1.Color;
  Form1.Canvas.Ellipse(x,y,x+10,y+10);
  x:=x+dx;

  // нарисовать окружность на новом месте
  Form1.Canvas.Pen.Color:=clBlack;
  Form1.Canvas.Ellipse(x, y, x+10, y+10) ;
end;

  // сигнал от таймера
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  Ris;
end;

procedure TForm1.FormActivate(Sender: TObject);
begin
  x:=0;
  y:=10;
  dx:=5;
  Timer1.Interval:=50;
  // период возникновения события OnTimer - 0.5 сек
  Form1.Canvas.Brush.Color:=Forml.Color;
end;
end.
```

Основную работу выполняет процедура Ris, которая стирает окружность и выводит ее на новом месте. Стирание окружности выполняется путем перерисовки окружности поверх нарисованной, но цветом фона.

Для обеспечения периодического вызова процедуры Ris в форму программы добавлен невизуальный компонент Timer (таймер), значок которого находится на вкладке **System** палитры компонентов. Свойства компонента Timer, перечислены в табл. 15.

Таблица 15

Свойства компонента Timer

Свойство	Определяет
Name	Имя компонента. Используется для доступа к компоненту
Interval	Период генерации события OnTimer. Задается в миллисекундах
Enabled	Разрешение работы. Разрешает (значение True) или запрещает (значение False) генерацию события OnTime

Добавляется компонент Timer к форме обычным образом, однако, поскольку компонент Timer является невизуальным, т. е. во время работы программы не отображается на форме, его значок можно поместить в любое место формы.

Компонент Timer генерирует событие OnTimer. Период возникновения события OnTimer измеряется в миллисекундах и определяется значением свойства Interval. Следует обратить внимание на свойство Enabled. Оно дает возможность программе "запустить" или "остановить" таймер. Если значение свойства Enabled равно False, то событие OnTimer не возникает.

Событие onTimer в рассматриваемой программе обрабатывается процедурой TimeriTimer, которая, в свою очередь, вызывает процедуру Ris. Таким образом, в программе реализован механизм периодического вызова процедуры Ris.

Примечание

Переменные x, y (координаты центра окружности) и dx (приращение координаты x при движении окружности) объявлены вне процедуры Ris, т. е. они являются глобальными. Поэтому надо не забыть выполнить их инициализацию (в программе инициализацию глобальных переменных реализует процедура FormActivate).

Использование битовых образов

В предыдущем примере изображение формировалось из графических примитивов. Теперь рассмотрим, как можно реализовать перемещение одного сложного изображения на фоне другого, например перемещение самолета на фоне городского пейзажа.

Эффект перемещения картинки может быть создан путем периодической перерисовки картинки с некоторым смещением относительно ее прежнего положения. При этом предполагается, что перед выводом картинки в новой точке сначала удаляется предыдущее изображение. Удаление картинки может быть выполнено путем перерисовки всей фоновой картинки или только той ее части, которая перекрыта битовым образом движущегося объекта.

В рассматриваемой программе используется второй подход. Картинка выводится применением метода Draw к свойству canvas компонента Image, а стирается путем копирования (метод copyRect) нужной части фона из буфера на поверхность компонента Image.

Форма программы приведена на рис. 118, а текст – далее.

Компонент image используется для вывода фона, а компонент Timer – для организации задержки между циклами удаления и вывода на новом месте изображения самолета.

Пример программы Летящий самолет:

```
unit anim_;

interface

uses
  Windows, Messages, SysUtils,
  Classes, Graphics, Controls,
  Forms, Dialogs, ExtCtrls, StdCtrls, Buttons;

type
TForm1 = class(TForm)
  Timer1: TTimer;
  Image1: Tlmage;
  procedure FormActivate(Sender: TObject);
  procedure Timer1Timer(Sender: TObject);
  procedure FormClose(Sender: TObject;
                      var Action: TCloseAction);
private
{ Private declarations }
public
{ Public declarations }
end;

var
  Form1: TForm1;

implementation

{$R *.DFM}
var
  Back, bitmap, Buf : TBitMap; // фон, картинка, буфер
  BackRct : TRect; // область фона, которая должна быть
  // восстановлена из буфера
  BufRet: Trect; // область буфера, которая используется
   // для восстановления фона
  x, y :integer; // текущее положение картинки
  W, H : integer; // размеры картинки

procedure TForm1.FormActivate(Sender: TObject);
begin
```

```
    // создать три объекта – битовых образа
    Back := TBitmap.Create; // фон
    bitmap := TBitmap.Create; // картинка
    Buf := TBitmap.Create; // буфер

    // загрузить и вывести фон
    Back.LoadFromFile('factory.bmp');
    Form1.Image1.Canvas.Draw(0,0,Back);

    // загрузить картинку, которая будет двигаться
    bitmap.LoadFromFile('aplane.bmp');

    // определим "прозрачный" цвет
    bitmap.Transparent := True;
    bitmap.TransParentColor := bitmap.Canvas.Pixels[1,1];

    // создать буфер для сохранения копии области фона,
    // на которую накладывается картинка
    W:= bitmap.Width;
    H:= bitmap.Height;
    Buf.Width:= W;
    Buf.Height:=H;
    Buf.Palette:=Back.Palette;

    // Чтобы обеспечить соответствие палитр
    Buf.Canvas.CopyMode:=cmSrcCopy;

    //определим область буфера, которая будет
    // использоваться для восстановления фона
    BufRct:=Bounds(0, 0, W, H);

    // начальное положение картинки
    x := -W;
    y := 20;

    // определим сохраняемую область фона
    BackRct:=Bounds(x, y, W, H); // и сохраним ее
    Buf.Canvas.CopyRect(BufRet,Back.Canvas,BackRct);
end;

// обработка сигнала таймера
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  // восстановлением фона (из буфера) удалим рисунок
  Forml.Image1.Canvas.Draw(x, y, Buf);
```

```
  x:=x+2;
  if x>Form1.Image1.Width then
   x:=-W;

  // определим сохраняемую область фона
  BackRct:=Bounds(x, y, W, H);

  // сохраним ее копию
  Buf.Canvas.CopyRect(BufRct, Back.Canvas, BackRct);
  // выведем рисунок
  Forml.Image1.Canvas.Draw(x, y, bitmap);
end;

// завершение работы программы
procedure TForm1.FormClose(Sender: TObject;
                           var Action: TCloseAction);
begin
  // освободим память, выделенную для хранения битовых
  //образов
  Back.Free;
  bitmap.Free;
  Buf.Free;
end;
end.
```

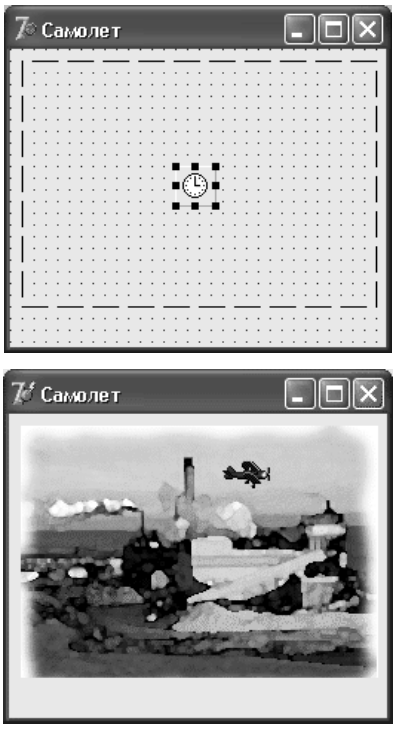


Рис. 118. Форма программы Самолет

Для хранения битовых образов (картинок) фона и самолета, а также копии области фона, перекрываемой изображением самолета, используются объекты типа TBitMap, которые создаются динамически процедурой FormActivate. Эта же процедура загружает из файлов картинки фона (factory.bmp) и самолета (aplane.bmp), а также сохраняет область фона, на которую первый раз будет накладываться картинка.

Сохранение копии фона выполняется при помощи метода CopyRect, который позволяет выполнить копирование прямоугольного фрагмента одного битового образа в другой. Объект, к которому применяется метод CopyRect, является приемником копии битового образа. В качестве параметров методу передаются координаты и размер области, куда должно быть выполнено копирование, поверхность, откуда должно быть выполнено копирование, а также положение и размер копируемой области. Информация о положении и размере копируемой в буфер области фона, на которую будет наложено изображение самолета и которая впоследствии должна быть восстановлена из буфера, находится в структуре BackRct типа TRect. Для заполнения этой структуры используется функция Bounds.

Следует обратить внимание на то, что начальное значение переменной x, которая определяет положение левой верхней точки битового образа движущейся картинки – отрицательное число, равное ширине битового образа картинки. Поэтому в начале работы программы изображение самолета не появляется, картинка отрисовывается за границей видимой области. С каждым событием OnTimer значение координаты x увеличивается, и на экране появляется та часть битового образа, координаты которой больше нуля. Таким образом, у наблюдателя создается впечатление, что самолет вылетает из-за левой границы окна.

Приложение

Приведенный здесь материал включен в пособие, чтобы ознакомить начинающих программистов с правильным структурным оформлением текста программы, а более «продвинутым» дать представление о стандарте оформления, разработанном в самой фирме, создавшей Delphi. Правильное, следующее стандарту, оформление текста программы, является обязательным элементом программистской культуры.

Стандарт стилевого оформления исходного кода DELPHI

Этот стандарт документирует стилевое оформление для форматирования исходного кода Delphi. В стандарте использованы материалы команды разработчиков Delphi [12]. Здесь используется сокращенный вариант стандарта.

Object Pascal является замечательно спроектированным языком. Одним из его многочисленных достоинств является легкая читабельность. Предлагаемый стандарт позволит еще более повысить легкость чтения кода Object Pascal. Следование довольно простым соглашениям, приведенным в этом стандарте, позволит разработчикам понять, что унифицированное оформление намного повышает читабельность и понятность кода. Использование стандарта намного облегчит жизнь во время этапов разработки и тестирования, когда довольно часто приходится подменять друг друга и разбираться в чужом коде.

Процесс перехода с собственного стиля оформления на предлагаемый может оказаться непростым, но человеческий мозг довольно легко адаптируется к стандартам и находит пути для быстрого запоминания предлагаемых правил. В дальнейшем, следование стандарту не вызывает затруднений. Для более комфортабельного перехода на этот стандарт предагается воспользоваться свободно распространяемой утилитой для форматирования исходных текстов "delforexp".

Этот документ не является попыткой определить грамматику языка Object Pascal. Например, если Вы попытаетесь поставить точку с запятой перед выражением else, то компилятор не позволит Вам этого сделать. Этот документ говорит Вам, как нужно поступать, когда есть возможность выбора из многих вариантов при оформлении Вашего исходного кода.

Файлы исходного кода

Исходный код Object Pascal подразделяется на модули и файлы проекта, которые подчиняются одинаковым соглашениям. Файл проекта Delphi имеет расширение DPR. Этот файл является главным исходным файлом для всего проекта. Любые модули, используемые в проекте,

всегда будут иметь расширение PAS. Дополнительные файлы, используемые в проекте (командные, html, DLL и т. д.) могут играть важную роль, но здесь описано форматирование только PAS и DPR файлов.

Именование исходных файлов

Язык Object Pascal поддерживает длинные имена файлов. Если при создании имени Вы используете несколько слов, то необходимо использовать заглавную букву для каждого слова в имени: MyLongName.pas. Этот стиль оформления известен как InfixCaps или CamelCaps. Расширения файлов должны быть в нижнем регистре.

Организация исходных файлов

Все модули Object Pascal могут содержать следующие элементы в определенном порядке:

- Информация о правах (**Copyright/ID**);
- Имя модуля (**Unit Name**);
- Объявление включаемых файлов (**Include files**);
- Секцию интерфейса (**Interface section**);
- Дополнительные определения (**Additional defines**);
- Объявление используемых модулей (**Uses clause**);
- Реализацию (**Implementation**)
- Объявление используемых модулей (**Uses clause**);
- Закрывающий оператор и точку (**A closing end and a period**).

Для визуального разделения элементов между ними должна быть хотя бы одна пустая строка.

Дополнительные элементы могут быть структурированы в порядке, который Вы сочтете нужным, но нужно соблюдать обязательные условия: в начале файла всегда копирайт, затем имя модуля, затем условные директивы и определения, директивы компилятора и файлы включения, затем определение подключений.

```
{*****************************************}
{                                         }
{ Модуль XXX                              }
{ Copyright (c) 2001 OOO XXXX             }
{ отдел/сектор                            }
{                                         }
{ Разработчик: XX XX                      }
{ Модифицирован: 25 июня 2001             }
{                                         }
{*****************************************}

unit Buttons;
```

Директивы компилятора не следует напрямую включать в исходный код. Для этого следует воспользоваться определением включений и подключить глобальный для проекта файл с директивами компилятора:

```
{$I NX.INC}
interface
```

В случае необходимости, можно напрямую переопределить глобальные директивы компилятора. Следует помнить, что переопределяющие директивы должны быть документированы, и Вы должны постараться ограничиться только локальным переопределением. Например, для одной процедуры:

```
{$S-,W-,R-}
{$C PRELOAD}

interface

uses
  Windows, Messages, Classes, Controls, Forms,
  Graphics, StdCtrls, ExtCtrls, CommCtrl;
```

Секции определения типов и констант Вы можете располагать относительно друг друга как Вам угодно. Секция реализации должна начинаться с ключевого слова implementation, затем объявление используемых модулей (Uses clause), затем любые включение файлов или другие директивы.

```
implementation

uses
 Consts, SysUtils, ActnList, ImgList;

{$R BUTTONS.RES}
```

Копирайт и комментарий

Пример заголовка для модуля:

```
{*******************************************}
{ Модуль XXX                                }
{ Copyright (c) 2001 ООО XXXX               }
{ отдел/сектор                              }
{                                           }
{ Разработчик: XX XX                        }
{ Модифицирован: 25 июня 2001               }
{                                           }
{*******************************************}
```

Следует обратить внимание на элементы заголовка:

- Назначение модуля;
- Копирайт;
- Разработчик;
- Дата последней модификации для исполняемой версии.

Объявление модуля

Каждый исходный файл должен содержать объявление модуля. Слово unit является ключевым, поэтому оно должно быть написано в нижнем регистре. Имя модуля может содержать символы как в верхнем, так и в нижнем регистре и должно быть таким же, как и имя используемое для этого файла операционной системой. Например:

```
unit MyUnit;
```

Этот модуль будет назван **MyUnit.pas**, когда он будет сохранен операционной системой.

Объявление используемых модулей

Внутри модуля объявление используемых модулей должно начинаться со слова uses в нижнем регистре. Затем следуют наименования модулей с сохранением регистра символов:

```
uses MyUnit;
```

Каждый используемый модуль должен отделяться от следующего с помощью запятой. Объявление используемых модулей должно заканчиваться точкой с запятой. Список используемых модулей необходимо располагать на следующей строке после слова uses. Если используются модули из разных проектов или производителей, то необходимо сгруппировать модули по проектам или производителям и каждую новую группу начинать с новой строки и снабжать комментариями:

```
uses
  Windows, SysUtils, Classes,
  Graphics, Controls, Forms, TypInfo, // модули Delphi
  XXXMyUnit1, XXXMyUnit2;    // модули XXX
```

Если список используемых модулей не умещается по ширине в 80 символов, то его необходимо перенести на следующую строку.

Объявление классов и интерфейсов

Объявление класса начинается с двух пробелов, затем идет идентификатор класса с префиксом T в нотации InfixCaps. Ни в коем случае в исходных файлах Object Pascal нельзя использовать табуляцию:

```
TMyClass
```

Следом за идентификатором класса идет пробел, знак равенства, пробел и слово class в нижнем регистре:

```
TMyClass = class
```

Если необходимо определить родителя класса, то следует добавить открывающую скобку, имя класса-родителя и закрывающую скобку:

```
TMyClass = class(TObject)
```

Объявления областей видимости начинаются с двух пробелов и, следовательно, области видимости располагаются на одном уровне с идентификатором класса:

```
TMyClass = class(TObject)
  private
  protected
  public
  published
end;
```

Данные всегда должны располагаться только в приватной секции и названия переменных должны всегда начинаться с префикса F. Все объявления внутри класса должны начинаться с четырех пробелов:

```
TMyClass = class(TObject)
  private
    FMyData: Integer;
    function GetData: Integer;
    procedure SetData(Value: Integer);
  public
  published
    property MyData:Integer read GetData write Set-
Data;
  end;
```

Оформление объявлений интерфейсов подчиняется тем же правилам, что и оформление классов. Отличие будет в использовании ключевых слов специфичных для интерфейсов.

Соглашение об именованиях

Исключая зарезервированные слова и директивы, которые всегда пишутся в нижнем регистре, все идентификаторы Object Pascal должны использовать InfixCaps:

```
MyIdentifier MyFTPClass
```

Самое главное исключение для всех правил состоит в использовании оттранслированных заголовочных файлов C/C++. В этом случае всегда используются соглашения, принятые в файле источнике. Например, будет использоваться `WM_LBUTTONDOWN`, а не `wm_LButtonDown`.

Для разделения слов нельзя использовать символ подчеркивания. Имя класса должно быть именем существительным или фразой с именем существительным. Имена интерфейсов или классов должны отражать главную цель их создания:

Правильно:

```
AddressForm
ArrayIndexOutOfBoundsException
```

Неправильно:

```
ManageLayout (глагол)
delphi_is_new_to_me (подчерк)
```

Именование полей

При именовании полей всегда необходимо использовать InfixCaps. Всегда объявлять переменные только в приватных частях и использователь свойства для доступа к переменным. Для переменных использовать префикс F.

Имена процедур для установки/получения значений свойств должны составляться по правилу: для получения − Get+имя свойства; для установки − Set+имя свойства.

- Не используйте все заглавные буквы для констант, за исключением оттранслированных заголовочных файлов.
- Не используйте венгерскую нотацию, за исключением оттранслированных заголовочных файлов.

Правильно

```
FMyString: string;
```

Неправильно

```
lpstrMyString: string;
```

Исключение для Венгерской нотации делается в случае объявления перечислимого типа:

```
TBitBtnKind = (bkCustom, bkOK, bkCancel, bkHelp,
  bkYes, bkNo, bkClose, bkAbort, bkRetry,
  bkIgnore, bkAll);
```

bk обозначает ButtonKind

Когда Вы раздумываете над именами переменных, то имейте в виду, что **нужно избегать однобуквенных имен, кроме как для временных переменных и переменных цикла**.

Переменные цикла именуются **`I`** и **`J`**. Другие случаи использования однобуквенных переменных это **`S`** (строка) и **`R`** (результат). Однобук-

венные имена должны всегда использовать символ в верхнем регистре, но лучше использовать боле значимые имена. Не рекомендуется использовать переменную **`l`** (эль), потому что она похожа на **`1`** (единица).

Именование методов

При именовании полей всегда необходимо использовать стиль InfixCaps. Не допускается использование символов подчеркивания для разделения слов. В имени метода всегда должна содержаться команда к действию или глагольная фраза

Правильно:

```
    ShowStatus
    DrawCircle
AddLayoutComponent
```

Неправильно:

`MouseButton` (Существительное, не описывает функцию)

`drawCircle` (Начинается с маленькой буквы)

`add_layout_component` (Используются символы подчерка)

`ServerRunning` (Глагольная фраза, но без команды)

Обратите внимание на последний пример (`ServerRunning`) – непонятно, что делает этот метод. Этот метод может использоваться для запуска сервера (лучше StartServer) или для проверки работы сервера (лучше `IsServerRunning`).

Методы для установки или получения значений свойств должны именоваться `Get`+имя свойства и – `Set`+имя свойства.

Например:

```
GetHeight, SetHeigh
```

Методы для теста/проверки булевских свойств класса должны именоваться с префиксом Is+имя свойства.

Например:

```
IsResizable, IsVisible
```

Именование локальных переменных

Имена всех локальных переменных должны подчиняться тем же правилам, которые установлены для именования полей, исключая префикс F.

Зарезервированные слова

Зарезервированные слова и директивы должны быть все в нижнем регистре. Производные типы должны начинаться с большой буквы (**`Integer`**), однако **`string`** – это зарезервированное слово и оно должно быть в нижнем регистре.

Объявление типов

Все объявления типов должны начинаться с префикса T и должны придерживаться правил, приведенных при описании оформления модуля или описании оформления класса.

Использование пробелов

Использование пустых строк

Пустые строки могут повысить читабельность путем группирования секций кода, которые логически связаны между собой. Пустые строки должны использоваться в следующих местах:

- После блока копирайта;
- После декларации пакета;
- После секции импорта;
- Между объявлениями классов;
- Между реализациями методов;

Использование пробелов

Язык Object Pascal является очень легким для понимания языком, поэтому нет особой необходимости в использовании большого количества пробелов. Следующие пункты дадут Вам понимание того, в каком случае необходимо использовать пробелы.

Пробелы, запрещенные к использованию

- До или после оператора . (точка);
- Между именем метода и открывающей скобкой;
- Между унарным оператором и его операндом;
- Между выражением приведения (cast) и приводимым выражением;
- После открывающей скобки или перед закрывающей;
- После открывающей квадратной скобки [или перед закрывающей];
- Перед точкой с запятой;

Примеры правильного использования:

```
function TMyClass.MyFunc(var Value: Integer);
MyPointer := @MyRecord;
MyClass := TMyClass(MyPointer);
MyInteger := MyIntegerArray[5];
```

Примеры неправильного использования:

```
function TMyClass.MyFunc( var Value: Integer ) ;
MyPointer := @ MyRecord;
MyClass := TMyClass ( MyPointer ) ;
MyInteger := MyIntegerArray [ 5 ] ;
```

Использование отступов

Всегда необходимо использовать два пробела для всех уровней отступа. Другими словами, первый уровень отступает на два пробела, второй на четыре и так далее. Никогда не используйте символы табуляции.

Существует несколько исключений из этого правила. Зарезервированные слова **unit, uses, type, interface, implementation, initialization** и **finalization** всегда должны примыкать к левой границе. Также должны быть отформатированы финальный end и end, завершающий исходный модуль. В файле проекта выравнивание по левой границе применяется к словам program, главным begin и end. Код внутри блока begin..end должен иметь отступ два символа.

Перенос строк

Все строки должны быть ограничены 80 столбцами. Строки, длиннее, чем 80 столбцов, должны быть разделены и перенесены. Все перенесенные строки должны быть выровнены по первой строке и иметь отступ в два символа. Выражение begin всегда должно находиться на своей отдельной строке.

Пример:

Правильно

```
function CreateWindowEx(dwExStyle: DWORD;
  lpClassName: PChar; lpWindowName: PChar;
  dwStyle: DWORD; X, Y, nWidth, nHeight: Integer;
  hWndParent: HWND; hMenu: HMENU; hInstance: HINST;
  lpParam: Pointer): HWND; stdcall;
```

Никогда не разрывайте строку между параметром и его типом, кроме параметров, перечисляемых через запятую. Двоеточие для всех объявлений переменных не должно содержать перед собой пробелов и иметь один пробел перед именем типа.

Правильно

```
procedure Foo(Param1: Integer; Param2: Integer);
```

Неправильно

```
procedure Foo( Param :Integer; Param2:Integer );
```

Нельзя переносить строки в тех местах, где не допускаются пробелы, например между именем метода и открывающей скобкой или между именем массива и открывающей квадратной скобкой. Никогда нельзя помещать выражение `begin` на строку, содержащую другой код.

Неправильно

```
while (LongExpression1 or LongExpression2) do begin
    // DoSomething
    // DoSomethingElse;
end;
```

Правильно

```
while (LongExpression1 or LongExpression2) do
begin
   // DoSomething
   // DoSomethingElse;
end;
if (LongExpression1) or
 (LongExpression2) or
```

В случае с логическими операторами предпочтительнее будет следующий вариант:

```
if (LongExpression1)
  or (LongExpression2)
  or (LongExpression3) then
```

Комментарии

Язык Object Pascal поддерживает два типа комментариев: блочные и однострочные. Общие соображение по использованию комментариев могут быть следующими:

- Помещайте комментарий недалеко от начала модуля для пояснения его назначения;
- Помещайте комментарий перед объявлением класса;
- Помещайте комментарий перед объявлением метода;
- Избегайте очевидных комментариев: (i := i + 1 // добавить к i единицу);
- Помните, что вводящий в заблуждение комментарий хуже чем его отсутствие;
- Избегайте помещать в комментарий информацию, которая со временем может быть не верна;
- Избегайте разукрашивать комментарии звездочками или другими символами;
- Для временных (отсутствующие в релизе) комментариев используйте "TODO".

Блочные комментарии

Object Pascal поддерживает два типа блочных комментариев. Наиболее часто используемый блочный комментарий − это пара фигурных скобок: { }. Команда разработчиков Delphi предпочитает использовать этот комментарий как можно проще и как запасной. Используйте в

таких комментариях пробелы для форматирования текста и не используйте символы зведочка "*". При переносе строк необходимо сохранять отступы и выравнивание

Пример из DsgnIntf.pas:

```
{ TPropertyEditor

  Edits a property of a component, or list of compo-
nents,
  selected into the Object Inspector. The property
  editor is created based on the type of the
  property being edited as determined by the types
  registered by...

  etc...

    GetXxxValue
     Gets the value of the first property in the
     Properties property. Calls the appropriate
     TProperty GetXxxValue method to retrieve the
     value.

    SetXxxValue Sets the value of all the properties
     in the Properties property. Calls the appropriate
     TProperty SetXxxxValue methods to set the value. }
```

В блочный комментарий всегда заключается информация о модуле: копирайт, дата модификации и так далее. Блочный комментарий, описывающий метод должен идти перед объявлением метода.

Правильно

```
{ TMyObject.MyMethod

  This routine allows you to execute code. }

procedure TMyObject.MyMethod;
begin
end;
```

Неправильно

```
procedure TMyObject.MyMethod;
{************************************************
  TMyObject.MyMethod

  This routine allows you to execute code.
    ************************************************}
begin
end;
```

Второй тип блочного комментария содержит два символа: скобку и звездочку: **(* *)**. Этот тип комментария используется при разработке исходного кода. Его преимуществом является то, что он поддерживает вложенные комментарии, правда комментарии должны быть разного типа. Вы можете использовать это свойство для комментирования больших кусков кода, в котором встречаются другие комментарии:

```
  (* procedure TForm1.Button1Click(Sender: TObject);
begin
  DoThis; // Start the process
  DoThat; // Continue iteration
  { We need a way to report errors here, perhaps using
    a try finally block ??? }
  CallMoreCode; // Finalize the process
end; *)
```

Однострочные комментарии

Однострочный комментарий состоит из символов // со следующим за ними текстом комментария. После символов // должен идти пробел и затем текст. Однострочные комментарии должны иметь отступы такие же, как и код, в котором они встречаются. Однострочные комментарии можно сгруппировать, чтобы сформировать большой комментарий.

Однострочный комментарий может начинаться с новой строки и может продолжать код, который он комментирует. В этом случае между кодом и комментарием должен быть хотя бы один пробел. Если больше одного комментария следует за кодом, то они должны быть выровнены по одному столбцу.

Пример однострочного строкового комментария:

```
// Open the table
Table1.Open;
```

Пример комментария в коде:

```
if (not IsVisible) then
  Exit;          // nothing to do
Inc(StrLength); // reserve space for null terminator
```

Необходимо избегать использовать комментарии в коде для каждой строки модуля.

Классы

Структура тела класса

Тело класса при его декларации подчинено следующей структуре:

- Объявление полей;
- Объявление методов;
- Объявление свойств.

Поля, свойства и методы в вашем классе должэны быть упорядочены в алфавитном порядке.

Уровни доступа

Исключая код, вставленный IDE, директивы видимости должны быть объявлены в следующем порядке:

- Приватные (скрытые) члены класса (**private**);
- Защищенные члены класса (**protected**);
- Общедоступные члены класса (**public**);
- Публикуемые члены класса (**published**)

Таким образом, в Object Pascal существует четыре уровня доступа для членов класса: **published**, **public**, **protected** и **private** − в порядке уменьшения видимости. По умолчанию, уровень доступа − **published**. В общем, члены класса должны давать наименьший уровень доступа, который подходит для этого члена. Например, член, к которому имеют доступ классы из одного модуля должен иметь уровень доступа private. Кроме того, объявляя члены класса с наименьшим уровнем доступа, Вы позволяете компилятору воспользоваться дополнительными возможностями для оптимизации. С другой стороны, если Вы планируете в дальнейшем порождать дочерние классы от Вашего класса, то нужно использовать уровень доступа protected.

Никогда не указывайте уровень доступа **public** для данных. Данные всегда должны быть объявлены в приватной секции и доступ к ним должен осуществляться с помощью методов или свойств.

Объявление конструктора

Все методы класса должны быть упорядочены по алфавиту. Однако Вы можете поместить объявления конструктора и деструктора перед всеми остальными методами. Если у класса существует более чем один конструктор и если они имеют одинаковые имена, то они должны располагаться в порядке увеличения числа параметров

Объявление методов

По возможности, объявление метода должно располагаться на одной строке:

Например:

```
procedure ImageUpdate(Image img, infoflags: Integer,
  x: Integer, y: Integer, w: Integer, h: Integer)
```

Операторы

Операторы − это одна или более строк кода, разделенных точкой с запятой. Простые операторы имеют одну точку с запятой, а составные могут иметь более чем одну точку с запятой и, таким образом, состоят из множества простых операторов.

Это ***простой оператор:***

```
A := B;
```

Это ***составной или структурированный оператор:***

```
begin
   B := C;
   A := B;
end;
```

Простые операторы

Простые операторы содержат одну точку с запятой. Если Вам необходимо разделить операторы, то перенесите продолжение оператора на следующую строку с отступом в два пробела:

```
MyValue :=
 MyValue + (SomeVeryLongStatement / OtherLongStatement);
```

Составные операторы

Составные операторы всегда заканчиваются точкой с запятой.

```
begin
  MyStatement;
  MyNext Statement;
  MyLastStatement;
end;
```

Присвоения и выражения

Каждое присвоение и каждое выражение должно располагаться на разных строках.

Правильно

```
a := b + c;
Inc(Count);
```

Неправильно

```
a := b + c; Inc(Count);
```

Объявление локальных переменных

Локальные переменные должны иметь стиль Camel Caps. Для локальных переменных префикс F не требуется.

```
var
  MyData: Integer;
  MyString: string;
```

Все переменные с их типами, особенно поля класса, должны быть объявлены на различных строках.

Объявление массивов

В объявлении массива перед и после квадратных скобок должны стоять пробелы.

```
type
  TMyArray = array [0..100] of Char;
```

Оператор if

Оператор if всегда должен располагаться, по крайней мере, на двух строках.

Неправильно

```
if A < B then DoSomething;
```

Правильно

```
if A < B then
  DoSomething;
```

В случае составного оператора необходимо поместить каждый оператор на новую строку.

Неправильно

```
if A < B then begin
    DoSomething;
    DoSomethingElse;
 end else begin
    DoThis;
    DoThat;
 end;
```

Правильно

```
if A < B then
begin
    DoSomething;
    DoSomethingElse;
end
else
begin
    DoThis;
    DoThat;
end;
```

Все остальные варианты расположения операторов не рекомендуются и не одобряются, хотя и являются синтаксически правильными. Избегайте использования круглых скобок в простых проверках. Например:

Правильно

```
if I > 0 then
 DoSomething;
```

Неправильно

```
if (I > 0) then
     DoSomething;
```

Оператор for

Неправильно

```
for i := 0 to 10 do begin
     DoSomething;
     DoSomethingElse;
 end;
```

Правильно

```
for i := 0 to 10 do
begin
     DoSomething;
     DoSomethingElse;
end;
for I := 0 to 10 do
     DoSomething;
```

Оператор while

Неправильно

```
while x < j do begin
     DoSomething;
     DoSomethingElse;
end;
```

Правильно

```
 while x < j do
 begin
     DoSomething;
     DoSomethingElse;
 end;
 while x < j do
     Something;
```

Оператор repeat until

Правильно

```
repeat
     x := j;
     j := UpdateValue;
until j > 25;
```

Оператор case

Несмотря на то, что существует множество синтаксически правильных конструкций, одобренной и рекомендованной считается следующая:

Правильно

```
case ScrollCode of
  SB_LINEUP, SB_LINEDOWN:
     begin
       Incr := FIncrement div FLineDiv;
       FinalIncr := FIncrement mod FLineDiv;
       Count := FLineDiv;
     end;
  SB_PAGEUP, SB_PAGEDOWN:
     begin
       Incr := FPageIncrement;
       FinalIncr := Incr mod FPageDiv;
       Incr := Incr div FPageDiv;
       Count := FPageDiv;
     end;
else
  Count := 0;
  Incr := 0;
  FinalIncr := 0;
end;
```

Оператор try

Несмотря на то, что существует множество синтаксически правильных конструкций, одобренной и рекомендованной считается следующая:

Правильно

```
try
  try
    EnumThreadWindows(CurrentThreadID, @Disable, 0);
    Result := TaskWindowList;
  except
    EnableTaskWindows(TaskWindowList);
    raise;
  end;
finally
    TaskWindowList := SaveWindowList;
    TaskActiveWindow := SaveActiveWindow;
end;
```

Дополнения

В этой части собраны дополнения, которые не вошли в стандарт Borland. Эти дополнения взяты из правил JCL и опыта российских разработчиков.

Const, Var и Type

Зарезервированные слова **var**, **const** и **type** всегда пишутся на новой строке и не допускают появления на этой же строке какого-либо текста.

Правильно

```
type
  TMyType = Integer;

const
  MyConstant = 100;

var
  MyVar: Integer;
```

Неправильно

```
type TMyType = Integer;

const MyConstant = 100;

var MyVar: Integer;
```

Процедуры должны иметь только по одной секции **type**, **const** и **var** в следующем порядке:

```
procedure SomeProcedure;
type
  TMyType = Integer;
const
  ArraySize = 100;
var
  MyArray: array [1..ArraySize] of TMyType;
begin
 ...
end;
```

Категории и разделение алгоритмов

Обычно, содержимое каждого созданного модуля есть набор классов, функций и процедур, принадлежащих к одной категории. Например, XXXLogin содержит все, что относится к идентификации и персонификации пользователя. Для ясного восприятия исходного кода следует придерживаться следующего правила: в интерфейсной части модуля каждая группа функций, относящихся к одной субкатегории должны отделяться от другой группы функций тремя строками шириной 80 столбцов с описанием субкатегории на второй строке:

```
1
2    Информация о последней попытке идентификации
3
procedure GetLastUserName(var ZUser: string);
procedure GetLastDatabase(var ZDatabase: string);
```

В секции реализации каждая подкатегория или класс должен разделяться строкой, состоящей из символов равенства (=), закомментированных однострочным комментарием и пустой строкой перед и после группы функций:

```
//=====================================================

procedure GetLastUserName(var ZUser: string);
begin
...
end;
procedure GetLastDatabase(var ZDatabase: string);
begin
...
end;

//=====================================================
```

Каждая функция из одной группы или методы класса должны разделяться между собой строкой, состоящей из символов минуса (−), закомментированных однострочным комментарием и пустой строкой перед и после функции или метода:

```
//=====================================================

procedure GetLastUserName(var ZUser: string);
begin
...
end;

//-----------------------------------------------------

procedure GetLastDatabase(var ZDatabase: string);
begin
...
end;

//=====================================================
```

Локальные процедуры

Локальные функции и процедуры должны иметь стандартный отступ в два пробела вправо от их владельца и сама процедура должна выделяться пустыми строками по одной перед и после локальной процедуры. Если "внешняя" процедура имеет локальные переменные, то они должны декларироваться перед локальной процедурой, независимо от того, будет ли в локальной процедуре осуществляться доступ к ним или нет. Однако общие соображения таковы, что локальных процедур следует избегать.

```
procedure SomeProcedure;
var
 I: Integer;

 procedure LocalProcedure;
 begin
 ...
 end;

begin
 ...
 LocalProcedure;
 ...
end;
```

Объявление параметров

Когда объявляется список параметров для процедуры, функции или метода пользуйтесь следующими рекомендациями:

- Комбинируйте формальные параметры одного типа в одно выражение;
- Не используйте префикс А, кроме случаев, когда вызывается метод класса, который работает со свойством, которе имеет идентичное имя;
- Придерживайтесь следующего порядка в параметрах: сначала входные параметры, затем входные/выходные, затем выходные. Параметры, имеющие значение по умолчанию по правилам Object Pascal помещаются в конец списка;
- Используйте const для параметров, которые не изменяются при работе вызываемых методов, процедур или функций.

Список литературы

1. Информатика. Базовый курс 2-е изд / Симонович С.В. и др. – СПб: Издательство «Питер», 2008. – 640 с.: ил.
2. В. Дьяконов. Mathcad 8/2000. Специальный справочник. Питер, 2000 г.
3. Гурин С.В., Немировский В.Б., Стоянов А.К.. Основные понятия и принципы программирования с иллюстрациями на языке Паскаль. Учебное пособие. – Томск, ТПУ, 1996. – 96 с.
4. Немировский В.Б, Стоянов А. К.. Информатика. Учебное пособие. – Томск, ТПУ, 2000. 120 с.
5. Иванова Г.С. Технология программирования: Учебник для вузов. − М.: Изд-во МГТУ им. Н.Э. Баумана, 2002. − 320 с.: ил. (Сер. Информатика в техническом университете.)
6. М.В. Сухарев. Основы Delphi. Профессиональный подход − СПб.: Наука и Техника, 2004. − 600 с.: ил.
7. Фленов М. Е. Библия Delphi. − СПб.: БХВ-Петербург, 2004. − 880 с.: ил.
8. Фаронов В. В.. Delphi. Программирование на языке высокого уровня: учебник для вузов – СПб. : Питер, 2005. − 640 с. : ил.
9. Архангельский А.Я. Delphi 2006. Справочное пособие: Язык Delphi, классы, функции Win32 и .NET. − М.: ООО «Бином-Пресс», 2006 г. − 1152 с: ил.
10. Ткаченко А.В.. Стандарт стилевого оформления исходного кода DELPHI // http://www.citforum.ru/programming/delphi/style_delphi/
11. Королевство Дельфи (Виртуальный клуб программистов) // http://www.delphikingdom.com/

Предметный указатель

Учебное издание

НЕМИРОВСКИЙ Виктор Борисович
СТОЯНОВ Александр Кириллович

Информатика

Учебное пособие

Научный редактор *канд. физ.-мат. наук, доцент Рейзлин В.И.*
Редактор
Компьютерная верстка
Дизайн обложки

Подписано к печати 00.00.2012. Формат 60x84/16. Бумага «Снегурочка».
Печать XEROX. Усл.печ.л 18,26. Уч.-изд.л. 16,51.
Заказ 000-12. Тираж 100 экз.

Национальный исследовательский Томский политехнический университет
Система менеджмента качества
Томского политехнического университета сертифицирована
NATIONAL QUALITY ASSURANCE по стандарту ISO 9001:2008

ИЗДАТЕЛЬСТВО ТПУ. 634050, г. Томск, пр. Ленина, 30
Тел./факс: 8(3822)56-35-35, www.tpu.ru