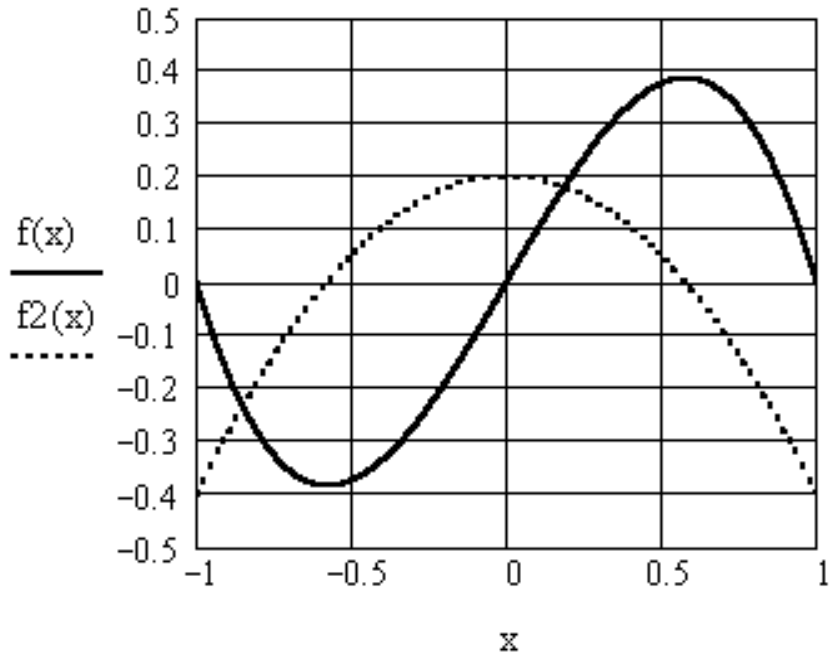
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

И.М. Егоров

ИНФОРМАТИКА

Учебное пособие



ТОМСК – 2007

Федеральное агентство по образованию

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра промышленной электроники (ПРЭ)

И.М. Егоров

ИНФОРМАТИКА

Учебное пособие

2007

Егоров И.М.
Информатика: Учебное пособие. — Томск: Томский государственный университет систем управления и радиоэлектроники, 2007. — 245 с.

Учебное пособие предназначено для студентов первого курса, обучающихся по направлению 210100 Электроника и микроэлектроника и специальности 210106 Промышленная электроника.

В пособии содержатся традиционные для курса «Информатика» разделы, касающиеся понятий информации, информационных процессов и технологий, приведена общая структура процессорных устройств обработки информации, рассмотрена структура и состав аппаратной части персональных компьютеров. В разделе «Компоненты прикладного программного обеспечения» рассмотрены текстовой процессор MS Word, электронная таблица MS Excel и СУБД MS Access. Значительное место в пособии отведено описанию работы с математическим процессором MathCad. Наибольшее внимание в пособии уделено изложению основ программирования на языке C++.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Информатика как отдельная научная и инженерная дисциплина сформировалась во второй половине XX в., прежде всего в связи с появлением и широким распространением вычислительных машин (компьютеров).

Академик В.М. Глушков определил информатику как новую область науки, связанную с разработкой, созданием, оценкой, использованием и материально-техническим обслуживанием систем обработки информации, включая машины и оборудование, материальное обеспечение, организационные и людские аспекты, а также комплекс их промышленного, коммерческого, административного, социального и политического воздействия.

Информатика — наука, которая изучает общие законы, методы и средства организации, переработки и использования информации на основе современных средств вычислительной техники и средств телекоммуникации. Информатика занимается научными и прикладными проблемами создания и функционирования информационного обеспечения процессов управления любыми объектами на базе компьютерных систем. Поэтому она тесно связана с теорией управления.

Информатика как наука о законах получения, передачи и использования информации в общественной практике подводит теоретический фундамент под использование ЭВМ и автоматизированных информационных систем, которые предназначены для реализации информационных процессов.

Объектами изучения информатики являются информационные процессы и информационные технологии.

Информационные процессы — это процессы сбора, накопления, хранения, обработки (переработки), распространения (передачи) и использования информации. Отметим, что процесс — это определенная совокупность действий, направленных на достижение поставленной цели.

Понятие «технологии» включает в себя материальные, технические, энергетические и трудовые факторы производства и способы их соединения для создания продукта или услуги.

Информационные технологии — это методы и средства реализации информационных процессов на основе современных средств вычислительной техники и средств телекоммуникации, совокупность методов, производственных процессов и программно-технических средств, обеспечивающих сбор, хранение, обработку и распространение информации.

1 ПОНЯТИЕ ИНФОРМАЦИИ

Вся жизнь человека, так или иначе, связана с накоплением и обработкой информации, которую он получает из окружающего мира, используя пять органов чувств — зрение, слух, вкус, обоняние и осязание. Как научная категория «информация» составляет предмет изучения для самых различных дисциплин: информатики, кибернетики, философии, физики, биологии, теории связи и т.д. Несмотря на это, строгого научного определения, что же такое информация, до настоящего времени не существует, а вместо него обычно используют понятие об информации. Понятия отличаются от определений тем, что разные дисциплины в разных областях науки и техники вкладывают в него разный смысл, с тем, чтобы оно в наибольшей степени соответствовало предмету и задачам конкретной дисциплины. Имеется множество определений понятия информации — от наиболее общего философского (информация есть отражение реального мира) до наиболее частного прикладного (информация есть сведения, являющиеся объектом переработки). Вот некоторые из них:

- сообщение, осведомление о положении дел, сведения о чем-либо, передаваемые модели;
- уменьшаемая, снимаемая неопределенность в результате получения сообщений;
- передача, отражение разнообразия в любых процессах и объектах, отраженное разнообразие;
- товар, являющийся объектом купли-продажи знаний для достижения определенных целей;
- данные как результат организации символов в соответствии с установленными правилами;
- продукт взаимодействия данных и адекватных им методов;
- сведения о лицах, предметах, фактах, событиях, явлениях и процессах независимо от формы их представления.

Наряду с названными существуют сотни других, зачастую противоречащих друг другу или взаимоисключающих определений информации. Многообразие этих определений свидетельствует о широте подхода к понятию информации и отражает становление концепции информации в современной науке.

Первоначально смысл слова «информация» (от лат. *informatio* — разъяснение, изложение) трактовался как нечто присущее только человеческому сознанию и общению: «знания, сведения, сообщения, известия, передаваемые людьми устным, письменным или другим способом». Затем смысл этого слова начал расширяться и обобщаться. Так, с позиций материалистической теории познания одним из всеобщих свойств материи (наряду с движением, развитием, пространством, временем и др.) было признано отражение, заключающееся в способности адекватно отображать одним реальным объектом другие реальные объекты, а сам факт отражения состояний одного объекта в состояниях другого (или просто одного объекта в другом) и означает присутствие в нем информации об отражаемом объекте. Таким образом, как только состояния одного объекта находятся в соответствии с состояниями другого объекта (например, соответствие между положением стрелки вольтметра и напряжением на его клеммах или соответствие между нашим ощущением и реальностью), это значит, что один объект отражает другой, т.е. содержит информацию о другом.

Высшая, специфическая форма отражения — сознание человека. Кроме этого, существуют и другие формы: психика (присущая не только человеку, но и животным, несущая в себе информацию, способную влиять на их эмоциональные состояния и поведение), раздражимость (охватывающая, помимо прочего, растения и простейшие организмы, реагирующие на слабые механические, химические, тепловые контакты с окружающей средой) и самая элементарная форма — запечатление взаимодействия (присущая и неорганической природе, и элементарным частицам, т.е. всей материи вообще).

Так, кусок каменного угля несет в себе «отражение» событий, произошедших в далекие времена, т.е. обладает свойством информативности. Деловое письмо с предложениями сотрудничества информативно, поскольку отражает серьезные намерения определенного учреждения или ведомства. Команда приступить к конкретным действиям (запуску ракеты, отправке груза, производству вычислений и т.п.) содержит информацию о подготовленности соответствующих служб и своевременности предпринимаемых шагов.

Информация не является ни материей, ни энергией. В отличие от них она может возникать и исчезать. В указанных примерах информация в куске каменного угля или делового письма может исчезнуть, если исчезнет ее носитель, например, сгорит.

Особенность информации заключается в том, что проявляется она только при взаимодействии объектов, причем обмен информацией может совершаться не вообще между любыми объектами, а только между теми из них, которые представляют собой организованную структуру (систему). Элементами этой системы могут быть не только люди: обмен информацией может происходить в животном и растительном мире, между живой и неживой природой, людьми и устройствами. Так, информация, заключенная в куске каменного угля, проявится лишь при взаимодействии с человеком, а растение, получая информацию о свете, днем раскрывает свои лепестки, а ночью закрывает их.

Понятие «информация» обычно предполагает наличие двух объектов — «источника» информации и «приемника» (потребителя, адресата) информации.

Информация передается от источника к приемнику в материально-энергетической форме в виде *сигналов* (например, электрических, световых, звуковых и т.д.), распространяющихся в определенной среде.

Сигнал (от лат. *signum* — знак) — физический процесс (явление), несущий сообщение (информацию) о событии или состоянии объекта наблюдения.

Информация может поступать непрерывно или дискретно, т.е. в виде последовательности отдельных сигналов. Соответственно различают непрерывную и дискретную информацию.

Информация — специфический атрибут реального мира, представляющий собой его объективное отражение в виде совокупности сигналов и проявляющийся при взаимодействии с «приемником» информации, позволяющим выделять, регистрировать эти сигналы из окружающего мира и по тому или иному критерию их идентифицировать.

Из этого определения следует, что:

- информация объективна, так как это свойство материи — отражение;
- информация проявляется в виде сигналов и лишь при

взаимодействии объектов;

- одна и та же информация различными получателями может быть интерпретирована по-разному, в зависимости от «настройки» «приемника».

Человек воспринимает сигналы посредством органов чувств, которые «идентифицируются» мозгом. Поэтому, например, при наблюдении одного и того же объекта человек с лучшим зрением может получить больше информации об объекте, чем тот, у кого зрение хуже. В то же время при одинаковой остроте зрения в случае, например, прочтения текста на иностранном языке человек, не владеющий этим языком, вообще не получит никакой информации, так как его мозг не сможет ее идентифицировать. Приемники информации в технике воспринимают сигналы с помощью различной измерительной и регистрирующей аппаратуры. При этом приемник, обладающий большей чувствительностью при регистрации сигналов и более совершенными алгоритмами их обработки, позволяет получить большие объемы информации.

Информация имеет определенные функции в обществе, основные из которых:

- **познавательная**, цель которой — получение новой информации. Функция реализуется в основном через такие этапы обращения информации, как:
  - ее синтез (производство),
  - представление,
  - хранение (передача во времени),
  - восприятие (потребление);
- **коммуникативная** — функция общения людей, реализуемая через такие этапы обращения информации, как:
  - передача (в пространстве),
  - распределение;
- **управленческая**, цель которой — формирование целесообразного поведения управляемой системы, получающей информацию. Эта функция информации неразрывно связана с познавательной и коммуникативной и реализуется через все основные этапы обращения, включая обработку.

Без информации не может существовать жизнь в любой форме и не могут функционировать созданные человеком любые

информационные системы. Без нее биологические и технические системы представляют груду химических элементов. Общение, коммуникации, обмен информацией присущи всем живым существам, но в особой степени — человеку. Будучи аккумулированной и обработанной с определенных позиций, информация дает новые сведения, приводит к новому знанию. Получение информации из окружающего мира, ее анализ и генерирование составляют одну из основных функций человека, отличающую его от остального живого мира.

1.1 Информационные процессы и системы

Роль информации может ограничиваться эмоциональным воздействием на человека, однако наиболее часто она используется для выработки управляющих воздействий в автоматических (чисто технических) и автоматизированных (человеко-машинных) системах. В подобных системах можно выделить отдельные этапы (фазы) обращения информации, каждый из которых характеризуется определенными действиями.

Последовательность действий, выполняемых с информацией, называют ***информационным процессом.***

Системы, реализующие информационные процессы, называют ***информационными системами.***

Основные этапы (фазы) обращения информации в системах:

- сбор (восприятие) информации;
- подготовка (преобразование) информации;
- передача информации;
- обработка (преобразование) информации;
- хранение информации;
- отображение (воспроизведение) информации.

Так как материальным носителем информации является сигнал, то реально это будут этапы обращения и преобразования сигналов).

На *этапе восприятия* информации осуществляется целенаправленное извлечение и анализ информации о каком-либо объекте (процессе), в результате чего формируется образ объекта, проводятся его опознание и оценка. Главная задача на этом этапе — отделить полезную информацию от мешающей (шумов), что в

ряде случаев связано со значительными трудностями. Простейшим видом восприятия является различение двух противоположных состояний: наличия («да») и отсутствия («нет»), более сложным — измерение.

На *этапе подготовки* информации осуществляется ее первичное преобразование. На этом этапе проводятся такие операции, как нормализация, аналого-цифровое преобразование, шифрование. Иногда этап подготовки рассматривается как вспомогательный на этапе восприятия. В результате восприятия и подготовки получается сигнал в форме, удобной для передачи, хранения или обработки.

На *этапе передачи* информация пересылается из одного места в другое (от отправителя получателю-адресату). Передача осуществляется по каналам различной физической природы, самыми распространенными из которых являются электрические, электромагнитные и оптические. Извлечение сигнала на выходе канала, подверженного действию шумов, носит характер вторичного восприятия.

На *этапах обработки информации* выявляются ее общие и существенные взаимозависимости, представляющие интерес для системы. Преобразование информации на этапе обработки (как и на других этапах) осуществляется либо средствами информационной техники, либо человеком.

В общем случае под *обработкой информации* понимается любое ее преобразование, проводимое по законам логики, математики, а также неформальным правилам, основанным на «здравом смысле», интуиции, — обобщенном опыте, сложившихся взглядах и нормах поведения. Результатом обработки является тоже информация, но либо представленная в иных формах (например, упорядоченная по каким-то признакам), либо содержащая ответы на поставленные вопросы (например, решение некоторой задачи). Если процесс обработки формализуем, он может выполняться техническими средствами. Кардинальные сдвиги в этой области произошли благодаря созданию ЭВМ — универсального преобразователя информации, в связи с чем появились понятия *данных* и *обработки данных.*

Данные — факты, сведения, представленные в формализованном виде (закодированные), занесенные на те или иные носи-

тели и допускающие обработку с помощью специальных технических средств (в первую очередь, ЭВМ).

Обработка данных предполагает проведение различных операций над ними, в первую очередь арифметических и логических, для получения новых данных, которые объективно необходимы (например, при подготовке ответственных решений).

На *этапе хранения* информацию записывают в запоминающее устройство для последующего использования. Для хранения информации используются в основном полупроводниковые, магнитные и оптические носители. Решение задач извлечения хранимой информации (поиска информации) связано с разработкой классификационных признаков и схем размещения хранимой информации, систематизацией, правилами доступа к ней, порядком ее пополнения и обновления, т.е. всем тем, что определяет возможность целенаправленного поиска и оперативного извлечения хранимой информации.

Этап отображения информации должен предшествовать этапам, связанным с участием человека. Цель этого этапа — предоставить человеку нужную ему информацию с помощью устройств, способных воздействовать на его органы чувств.

Информационные системы можно классифицировать по различным признакам. Так, по сфере применения информационные системы подразделяются на административные, производственные, учебные, медицинские, военные и др., по территориальному признаку — информационные системы района, города, области и т.п. С точки зрения возможности организации конкретных информационных процессов различают информационно-справочные, информационно-поисковые системы, системы обработки и передачи данных, системы связи.

Большинство автоматизированных информационных систем являются локальными системами и функционируют на уровне предприятий и учреждений. В настоящее время происходит интенсивный процесс интеграции таких систем в корпоративные системы и далее — в региональные и глобальные системы.

Системы более высокого уровня становятся территориально рассредоточенными, иерархичными как по функциональному принципу, мак и по их технической реализации. Обеспечение взаимодействия территориально рассредоточенных систем требу-

ет протяженных высокоскоростных и надежных каналов связи, а увеличение объема обрабатываемой информации — ЭВМ высокой производительности. Это приводит к необходимости коллективного использования дорогостоящих средств автоматизации (ЭВМ и линий связи) и обрабатываемой информации (баз данных). Техническое развитие, как самих электронных вычислительных машин, так и средств связи позволило решить эту проблему путем перехода к созданию *распределенных информационно-вычислительных сетей коллективного пользования.*

Централизация различных видов информации в одной сети дает возможность использовать ее для решения широкого спектра задач, связанных с административным управлением, планированием, научными исследованиями, конструкторскими разработками, технологией производства, снабжением, учетом и отчетностью.

Если поставляемая информация извлекается из какого-либо объекта (процесса), а выходная применяется для целенаправленного изменения состояния того же объекта (процесса), причем абонентом, использующим информацию для выбора основных управляющих воздействий (принятия решения), является человек, то такую автоматизированную информационную систему называют *автоматизированной системой управления* (АСУ).

Управление и информация служат основными понятиями кибернетики — науки об общих принципах управления в различных системах: технических, биологических, социальных и др.

Управление — функция организованных систем различной природы (технических, биологических или социальных), направленная на реализацию их целевых установок и поддержание внутренне присущей им структуры.

Понятие «кибернетика» как научный термин введено в первой половине XIX в. французским физиком Андре Мари Ампером, который назвал кибернетикой (от греч. *kybernetike* — искусство управления) науку, занимающуюся изучением искусства управления людьми, обществом. В Древней Греции этого титула удостаивались лучшие мастера управления боевыми колесницами. Впоследствии слово «кибернетикос» было заимствовано римлянами — так в латинском языке появилось слово «губернатор» (управляющий провинцией).

Основоположником кибернетики считается выдающийся американский математик *Норберт Винер* (1894—1964), а датой ее рождения —1948 г., когда Н. Винер опубликовал книгу «Кибернетика, или Управление и связь в животном и машине».

Кибернетика — наука, изучающая с единых позиций связь и управление (самоуправление) в организованных системах любой физической природы.

Сущность кибернетики в самом общем виде может быть выражена основными ее законами. Кибернетическая система (система управления) может рассматриваться как совокупность двух систем — управляющего объекта и объекта управления. При этом управляющая система воздействует на объект управления, подавая на него управляющие сигналы (управляющие воздействия).

Для выработки управляющих решений, обеспечивающих достижение цели управления, управляющая система осуществляет сбор информации о текущем состоянии объекта управления (по каналу обратной связи), ее хранение (накопление), а также оценку текущего состояния управляемого объекта с желаемым (соответствующим цели).

Автоматизированные системы управления нашли широкое применение во всех сферах современного общества, в первую очередь как системы управления технологическими процессами и коллективами людей. АСУ технологическими процессами служат для автоматизации различных функций на производстве. Они широко используются при организации поточных линий, изготовлении микросхем, для поддержания технологического цикла в машиностроении и т.п. Информационные системы организационного управления предназначены для автоматизации функций управленческого персонала, например информационные системы управления банками, гостиницами, торговыми фирмами и т.п.

1.2 Информационные ресурсы и технологии

Особенностью современного этапа развития общества является переход от индустриального общества к информационному. Процесс, обеспечивающий этот переход, называют информатизацией.

Информатизация общества — организованный социаль-

но-экономический и научно-технический процесс создания оптимальных условий для удовлетворения информационных потребностей и реализации прав граждан, органов государственной власти, органов местного самоуправления, организаций, общественных объединений на основе формирования и использования информационных ресурсов.

Неизбежность информатизации обусловлена резким возрастанием роли и значения информации. Для нормального функционирования организации любого масштаба уже не достаточно только традиционных для индустриального общества ресурсов (материальных, природных, трудовых, финансовых, энергетических), необходимо знать, как наиболее эффективно эти ресурсы использовать, иметь информацию о технологиях. Поэтому существенным ресурсом стала информация. Информационные ресурсы в настоящее время рассматриваются как отдельная экономическая категория, важнейший стратегический ресурс общества.

Информационные ресурсы — отдельные документы и отдельные массивы документов, документы и массивы документов в информационных системах (библиотеках, архивах, фондах, банках данных, других информационных системах).

Информационная система — организационно упорядоченная совокупность документов, информационных технологий, в том числе с использованием средств вычислительной техники и связи, реализующих информационные процессы.

В общем случае под *информационными ресурсами* понимают весь имеющийся в информационной системе объем информации, отчужденной от ее создателей и предназначенной для общественного использования.

В отличие от других видов ресурсов (материальных, природных и др.), информационные ресурсы практически неисчерпаемы: по мере развития общества и роста потребления информации их запасы не убывают, а растут. Эта специфика информационных ресурсов хорошо иллюстрируется следующим высказыванием: «Если у вас есть по яблоку и вы обменяетесь ими, у вас опять будет по яблоку, но если у вас есть по идее и вы обменяетесь ими, то у каждого их будет по две». Более того, в процессе применения информационные ресурсы постоянно развиваются и совершенствуются, избавляясь от ошибок и уточняя свои параметры.

Выделяют пассивную и активную формы информационных ресурсов. К *пассивной форме* относятся книги, журнальные статьи, патенты, банки данных и т.п. Примерами *активных форм* служат: модель, алгоритм, проект, программа и т.п.

Государственная политика в сфере формирования информационных ресурсов и информатизации направлена на создание условий для эффективного и качественного информационного обеспечения решения задач социально-экономического развития. Основные направления государственной политики в этой области:

- обеспечение условий для развития и защиты всех форм собственности на информационные ресурсы;
- формирование и защита информационных ресурсов;
- создание и развитие федеральных и региональных информационных систем и сетей, обеспечение их совместимости и взаимодействия в едином информационном пространстве РФ;
- создание условий для качественного и эффективного информационного обеспечения граждан, органов государственной власти, органов местного самоуправления, организаций и общественных объединений на основе государственных информационных ресурсов;
- содействие формированию рынка информационных ресурсов, услуг, информационных систем, технологий и средств их обеспечения;
- формирование и осуществление единой научно-технической и промышленной политики в сфере информатизации с учетом современного мирового уровня развития информационных технологий;
- создание и совершенствование системы привлечения инвестиций и механизма стимулирования разработки и реализации проектов информатизации;
- развитие законодательства в сфере информационных процессов, информатизации и защиты информации.

Базовой технической составляющей процесса информатизации общества является компьютеризация. Под компьютеризацией понимается развитие и внедрение технической базы — компьютеров, обеспечивающих оперативное получение результатов переработки информации и ее накопление.

Научным фундаментом процесса информатизации общества является информатика, призванная создавать новые *информационные технологии и системы* для решения задач информатизации.

Технология (от греч. lechrie — искусство, мастерство, умение и греч. logos — слово, учение) — совокупность методов обработки, изготовления, изменения состояния, свойств, формы сырья, материала или полуфабриката, осуществляемых в процессе производства продукции.

Основными компонентами материальных технологий являются: подготовка сырья и материалов, производство материального продукта, сбыт произведенных продуктов потребителям.

В информационной технологии в качестве исходного материала выступает информация. В качестве конечного продукта — также информация, но это качественно новая информация о состоянии объекта, процесса или явления. При этом основными компонентами информационных технологий служат: сбор данных (первичной информации), обработка данных, получение результатной информации и передача ее потребителю.

Выделяют несколько поколений информационных технологий:

- самую древнюю — «наскально-берестяную»;
- «бумажную», связанную с изобретением печатного станка (середина XV в.);
- «безбумажную», или «электронную», относящуюся к появлению ЭВМ (середина XX в.);
- «новую информационную технологию», связанную с внедрением персональных ЭВМ и телекоммуникационных средств (с середины 1980-х гг.).

Новая информационная технология (компьютерная информационная технология) — технология, основанная на использовании персональных компьютеров и телекоммуникационных средств.

В процессе информатизации общества происходит проникновение информационных технологий во все сферы жизнедеятельности общества, в том числе и связанные с принятием ответственных решений.

1.3 Негативные последствия информатизации

Сейчас информационные технологии оказывают существенное воздействие на многие сферы деятельности людей, человеческих коллективов и общество в целом. Это воздействие в некоторых случаях может иметь и негативный характер.

Во-первых, вычислительная техника все более широко внедряется в системы управления такими технологическими процессами, выход которых за регламентированные пределы грозит не только крупными авариями, но и крупномасштабными катастрофами (системы управления вооружением, атомными реакторами и т.п.). Отличительной особенностью таких систем управления является необходимость осуществления сложных видов обработки больших объемов информации в крайне ограниченные сроки.

В силу этого сложность систем неуклонно растет, а гарантировать отсутствие ошибок в программном обеспечении, исчисляемом многими десятками миллионов машинных команд, практически невозможно. Помимо этого, возможны сбои или отказы аппаратуры, провокационные и диверсионные действия персонала, заражение компьютеров электронными вирусами и т.п. Нетрудно представить себе возможные последствия таких событий в системе военного назначения. Иными словами, в современных условиях надо защищать как системы обработки информации от воздействия внешней среды, так и среду от воздействия информации, находящейся в системах обработки. Должна быть обеспечена не только безопасность информации, накапливаемой, хранимой и обрабатываемой в системах, но также и информационная безопасность окружающей среды, т.е. предупреждение негативного воздействия на окружающую среду, которое может иметь место в результате непредусмотренных (ошибочных или злоумышленных) видов обработки.

Во-вторых, массовое использование вычислительной техники в различных сферах деятельности резко увеличивает потенциальные возможности нарушения гражданских прав и свобод человека, поскольку в условиях повсеместного внедрения новых информационных технологий расширяются возможности ведения досье на людей, подслушивания телефонных разговоров, несанкционированного чтения электронной почты, контролирования вкладов, осуществления компьютерной слежки и т.п.

В-третьих, увеличивается опасность нарушения авторского права и права собственности, в первую очередь на программные продукты. Во многих случаях программное обеспечение как отдельными пользователями, так и целыми организациями приобретается в результате незаконного копирования, т.е. хищения.

В-четвертых, информатизация может являться и источником социальной напряженности. Так, автоматизация производства ведет к полному изменению технологии, что влечет за собой смену номенклатуры профессий и сокращение численности персонала. При этом не все люди могут легко освоить новую специальность или найти новое место работы.

Заслуживает быть отмеченным и такой несколько необычный аспект, как опасность профанации искусства, поскольку широкие возможности современных ЭВМ позволяют существенно интенсифицировать процесс творчества, а интенсификация его сверх некоторой меры неизбежно скажется на качестве произведения. Кроме того, по мнению некоторых ученых, руководители и специалисты, длительное время и регулярно использующие вычислительную технику в процессе своей деятельности, теряют навык выполнения своих функций.

Негативные аспекты информационных технологий необходимо учитывать при решении задач информатизации. Эти проблемы должны также стать предметом изучения современной информатики.

1.4 Персональный компьютер и здоровье[1]

Впервые комплексные исследования воздействия компьютера на организм пользователей были проведены в 1984 г. Поводом для их проведения стали многочисленные жалобы сотрудниц одного из канадских госпиталей. По итогам анкетирования и результатам измерений параметров излучений на рабочих местах было установлено, что причиной ухудшения здоровья (головные боли, бессонница, другие жалобы) были электромагнитные излучения монитора компьютера.

[1] Материал этого раздела написан профессором Сибирского государственного медицинского университета Волкотруб Людмилой Петровной.

Анализ обобщенных данных канадских исследователей показал, что у работающих за монитором от 2 до 6 часов в сутки функциональные нарушения центральной нервной системы происходят в 4,6 раза, болезни опорно-двигательного аппарата — в 3,1 раза, болезни сердечно-сосудистой системы — в 2 раза, болезни верхних дыхательных путей — в 1,9 раза чаще, чем в контрольных группах.

Изучение здоровья операторов персональных компьютеров (ПК) является объектом исследовательских программ Национальной академии наук и Национального института охраны труда и профилактики профессиональных заболеваний США, научно-медицинских учреждений Швеции, Франции, Германии, Австрии, Японии и России. Специалисты различных отраслей знаний, обобщив результаты комплексных междисциплинарных исследований, пришли к выводу, что причиной отклонений здоровья пользователей ПК являются не столько сами компьютеры, как источники неблагоприятно действующих на организм факторов, сколько недостаточно строгое соблюдение принципов эргономики. Эргономика (ergos — труд, nomos — закон) — отрасль знаний, изучающая трудовые процессы с целью создания наилучших условий труда.

По данным Бюро трудовой статистики США, в период с 1982 по 1990 гг. наблюдалось 8-кратное увеличение случаев расстройств здоровья (нетрудоспособности) у пользователей персональных компьютеров. Обследование 1583 женщин, проведенное в Окленде (штат Калифорния, США) Кайзеровским медицинским центром, показало, что у женщин, более 20 часов в неделю пользующихся компьютерными терминалами, риск выкидыша на ранних и поздних стадиях беременности на 80 % выше в сравнении с женщинами, выполняющими ту же работу без дисплейных терминалов.

У женщин, работающих за компьютером, в 2 раза увеличивается частота самопроизвольных абортов в первые три месяца беременности, в 2,5 раза выше вероятность рождения детей с врожденными пороками развития в сравнении с женщинами, не занятыми компьютерным трудом. Поэтому женщины со времени установления беременности должны переводиться на работы, не связанные с использованием ПК.

Специалисты нью-йоркского комитета по охране труда и профилактике профессиональных заболеваний считают, что беременных или имеющих намерение забеременеть женщин необходимо переводить на работы, не связанные с использованием видеодисплейных терминалов.

Большинство исследователей подчеркивают, что отрицательное влияние персональных компьютеров наиболее сильно проявляется у детей и молодых людей. Было показано, что за 20 минут работы с ПК у школьников в возрасте 10 лет активность правого полушария головного мозга снижалась в 1,5—2 раза.

Многие люди, постоянно работающие с компьютером, через короткое время работы за монитором, отмечают головную боль, болезненные ощущения в области мышц лица и шеи, ноющие боли в позвоночнике, боли при движении рук, резь в глазах, нарушение ясного видения. Степень выраженности этих симптомов пропорциональна времени работы за ПК.

Из-за длительного сидения в малоподвижной позе у некоторых операторов развивается мышечная слабость, происходит изменение формы позвоночника, что определяется термином — *синдром длительной статической нагрузки***.**

У работающих с отображенной на экране монитора информацией по семь и более часов в день вероятность появления *астенопических жалоб* (пелена перед глазами, неясные очертания предметов, ощущение инородного тела в глазах) и воспаления глаз значительно выше в сравнении с людьми, работа которых не связана с компьютером. Кроме того, установлено, что среди профессиональных операторов ПК отмечается повышенная частота заболеваний глаукомой и катарактой. По данным Всемирной Организации Здравоохранения, люди, работающие за ПК, вынуждены каждые 6—9 месяцев менять очки на более сильные.

Постоянные пользователи ПК часто подвергаются психологическим стрессам, у них наблюдаются функциональные нарушения центральной нервной системы, болезни сердечно-сосудистой системы и верхних дыхательных путей.

Среди пользователей ПК выявлен комплекс симптомов, названный *синдромом компьютерного стресса* (СКС), который проявляется головной болью, воспалением слизистой оболочки глаз,

повышенной раздражительностью, вялостью, депрессией. Симптомы этого патологического состояния организма разнообразны:

1. *Общее недомогание***.** Оно проявляется в сонливости; повышенной утомляемости; непроходящей усталости (даже после отдыха); головных болях после работы; болях в нижней части спины, области бедер, ногах; чувстве покалывания, онемения и болях в руках, запястьях, кистях; напряженности мышц шеи, спины, плеч и рук.

2. *Глазные симптомы* — быстрая зрительная утомляемость, покраснений глаз, слезотечение, чувство острой боли, жжение, зуд; частое моргание, ощущение инородного тела в глазах.

3. *Нарушения визуального восприятия* — снижение остроты зрения как на дальнее расстояние сразу после работы за ПК (пелена перед глазами), так и на близком расстоянии (изображение на экране плохо фокусируется); снижение остроты зрения усиливается в течение дня; двоение в глазах; очки становятся «слабыми», требуется смена очков; медленная рефокусировка; косоглазие.

4. *Снижение сосредоточенности и работоспособности***.** Оно является следствием визуальных нарушений и проявляется в невозможности в течение длительного времени сконцентрировать внимание; повышенной раздражительностью во время и после работы; потерей рабочей точки на экране; пропуском строк, слов, вводом повторных строк; ошибками при заполнении колонок; переставлении букв, слов и цифр местами.

Причинами СКС являются:

- сверхнапряженная работа глаз;
- использование несоответствующих очков и контактных линз;
- неправильное положение тела при работе за ПК;
- неправильная организация рабочего места;
- суммирование умственных, физических и визуальных нагрузок.

Путем исключения или минимизации воздействия отрицательных факторов компьютерного труда на организм оператора ПК можно существенно снизить вероятность возникновения СКС

Электромагнитное поле (ЭМП) около компьютера, особенно низкочастотное, оказывает определенное влияние на нервно-психическое состояние пользователей ПК, вызывая головные бо-

ли, бессонницу, головокружение, депрессию, отсутствие аппетита, тошноту.

Низкочастотное ЭМП может провоцировать кожные заболевания — угревую сыпь, экзему, розовый лишай и др. Исследования ученых Института общей генетики им. Н.И. Вавилова показали, что длительное воздействие компьютерного излучения приводит к изменениям в системе крови, аналогичным таковым у онкологических больных и больных с аутоиммунными заболеваниями.

Исследование реакций организма при работе с видеодисплейными терминалами, проведенное на Украине, показало, что среди различных изменений функционального состояния организма наиболее сильно выражены отклонения со стороны гормональной и иммунной систем. Нарушения в иммунном статусе являются основополагающими в дискоординации процессов, которые поддерживают гомеостаз организма.

Многочисленные эксперименты на животных подтверждают возможность воздействия слабых электромагнитных полей сверхнизких и низких частот на биологические объекты, особенно на мозг. Низкочастотные электромагнитные поля в комплексе с другими отрицательными факторами компьютерного труда могут индуцировать злокачественные новообразования.

Исследованиями показано, что при обследовании 295 человек, из которых 55 постоянно работали за компьютерами, в организме операторов происходят изменения, которые можно охарактеризовать как постепенную утрату контроля за размножением и ростом клеток, то есть как предстадию развития новообразований. При этом отмечена прямая зависимость между длительностью работы с ВДТ и степенью изменения иммунной системы.

Длительная работа за компьютером приводит к снижению внимания, ухудшению восприятия и переработки информации, возникновению негативно-эмоциональных состояний, депрессий.

Работа компьютера сопровождается положительной ионизацией воздуха и генерацией высоковольтного статического потенциала. Воздействие этих факторов, как правило, совпадает по времени и в совокупности может приводить к снижению напряженности иммунитета, нарушению сна, повышенной утомляемости и склонности к кожно-аллергическим болезням.

1.5 История развития информатики

Задачи накопления (хранения), обработки и передачи информации стояли перед человечеством на всех этапах его развития. Каждому этапу соответствовал определенный уровень развития средств информационного труда. Долгое время средства хранения, обработки и передачи информации развивались независимо друг от друга.

В течение этого времени основными инструментами для решения задач обработки и передачи информации были мозг, язык и слух человека. Первое кардинальное изменение произошло с приходом письменности. Это привело к гигантскому качественному и количественному скачку в развитии общества, появилась возможность передачи знаний от поколения к поколению. Изобретение книгопечатания (середина XV в.) радикально изменило индустриальное общество, культуру, организацию деятельности.

Эти два этапа (письменность и книгопечатание) создали принципиально новую технологию накопления и распространения (передачи) информации, избавившую человечество от необходимости всецело полагаться на такой зыбкий и ненадежный инструмент, каким является человеческая память.

Конец XIX в. ознаменован изобретением электричества, благодаря которому появились телеграф, телефон, радио, позволяющие оперативно передавать и накапливать информацию в любом объеме.

Бурное развитие науки и промышленности в XX в., неудержимый рост объемов поступающей информации привели к тому, что человек оказался не в состоянии воспринимать и перерабатывать все ему предназначенное. Возникла необходимость классифицировать поступления по темам, организовать их хранение, доступ к ним, понять закономерности движения информации в различных изданиях и т.д. Исследования, позволяющие разрешить возникшие проблемы, стали называть информатикой. В этом смысле информатика — научная дисциплина, изучающая структуру и общие свойства научной информации, а также закономерности всех процессов научной коммуникации.

Информатика, являясь базой библиотечного дела, многие

годы гак и занималась изучением структуры и общих свойств научной информации, передаваемой посредством научной литературы. Тогда преждевременно было говорить о том, что информатика связана с разработкой эффективных методов сбора, хранения, обработки и преобразования информации. Не существовало почти ничего общего в методах сбора и обработки информации у медиков, географов, психологов, физиков, филологов и т.д. С этой точки зрения много общего между собой имели математика и физика, химия и медицина. Примеров отдельных связей было много, но общего стержня, вокруг которого объединились бы все науки, не было.

Положение существенно изменилось с появлением электронно-вычислительных машин (ЭВМ).

Первые ЭВМ создавались для проведения расчетов в атомной физике, в авиационной и ракетной технике. Последовавшее далее внедрение ЭВМ в области административного управления и экономики дало не только большой экономический эффект, но и привело к созданию и бурному росту новой промышленной отрасли — средств и методов электронной обработки информации. Электронно-вычислительные машины стали обрабатывать числовую, текстовую, графическую и другую информацию. Появились новые ЭВМ, новые методы и средства общения с ними. Информация стала товаром.

Вычислительная техника сразу же показала свою эффективность в тех областях человеческой деятельности, где широко использовались методы математического моделирования — точные количественные методы. Сюда относятся физика, механика, химия, геофизика и т.д.

Развитие электронно-вычислительной техники, средств и методов общения с ней, создание автоматизированных информационно-поисковых систем, методов распознавания образов привели к тому, что ЭВМ стали эффективным инструментом и для «описательных» наук, которые раньше считались недоступными для методов математического моделирования (биология, юридические науки, история и т.п.). В них шло накопление отдельных фактов, давалось качественное описание объектов и событий. Использование нового рабочего инструмента значительно повысило эффективность проведения описательного анализа изучае-

мых объектов в таких науках.

Появилось новое направление исследований, связанное с машинным моделированием человеческих интеллектуальных функций, — разработка «искусственного интеллекта».

Миниатюризация средств вычислительной техники, снижение ее стоимости позволили создавать станки с программным управлением, гибкие автоматизированные производства, станки-роботы, в которых ЭВМ решает задачи сбора, хранения, обработки, преобразования информации и на основе ее анализа вырабатывает соответствующие решения. В более сложных задачах человек, используя электронную технику, берет ответственность за принятие решения на себя. В этом случае ЭВМ анализирует огромные объемы информации и предлагает возможные варианты. Человек, познакомившись с этими вариантами, либо выбирает лучший с его точки зрения, либо ставит перед машиной новые условия и ждет следующего совета. Так — в режиме диалога — происходит процесс принятия решения.

Проведение любого эксперимента связано с получением информации, регистрируемой различными датчиками или непосредственно органами чувств человека. В ходе эксперимента информацию надо принять и записать, обработать по специальным алгоритмам, преобразовать к удобному для анализа виду. Далее, исследуя полученные результаты, необходимо сделать выводы.

При этом не имеет значения, какой это эксперимент — физический, биологический, химический и т.д., передают ли датчики данные прямо в ЭВМ или показания приборов сначала записывают в тетрадь, а потом вводят их в машину. Главное в том, что нужны алгоритмы сбора данных и записи их в запоминающие устройства в таком виде, который позволяет находить эти данные повторно, считывать и анализировать.

Другой важнейшей составной частью эксперимента является обработка данных по разработанным алгоритмам и составленным на их основе программам для вычислительной машины.

На следующем этапе активно используются программы преобразования данных к удобному для исследования виду (построение графиков, таблиц, рабочих чертежей и т.д.) и их выдача (отображение информации) или передача другим участникам эксперимента, находящимся на значительном расстоянии. Как

правило, такие программы не ориентированы на конкретную предметную область, они достаточно универсальны.

Таким образом, мы выделили задачи, которые являются общими для всех наук при обработке информации с помощью ЭВМ. Научным фундаментом для их решения и стала новая наука — информатика.

В этом смысле слово «информатика» второй раз появляется в научной среде. Теперь — как перевод с французского informatique. Французский термин informatique (информатика) образован путем слияния слов information (информация) и automatique (автоматика) и означает «информационная автоматика, или автоматизированная переработка информации». В англоязычных странах этому термину соответствует синоним computer science (наука о вычислительной технике).

Появление ЭВМ сыграло решающую роль в оформлении информатики как науки, но и сама ЭВМ, ее создание, функционирование и применение — тоже предмет изучения информатики. Практики показала, что использование ЭВМ резко повысило производительность труда на производстве и в науке, оказало сильное влияние ил научно-технический прогресс. В то же время существует и обратное влияние — задачи науки и практики предъявляют конструкторам и разработчикам программ требования для создания новых, более высокопроизводительных ЭВМ, ориентированных на решение конкретных проблем.

Раннее употребляемый в русском языке термин «информатика», связанный лишь с областью изучения структуры и общих свойств научной информации, передаваемой с помощью научной литературы, в современных условиях приобретает более широкое значение. Сейчас это — название комплексной научно-технической дисциплины, призванной создавать новые информационные технологии и средства для решения проблем информатизации в различных областях человеческой деятельности: производстве, управлении, науке, образовании, торговле, финансовой сфере, медицине и др.

Информатика — комплексная научно-техническая дисциплина, занимающаяся изучением структуры и общих свойств и и формации, информационных процессов, разработкой на этой основе информационной техники и технологии, а также решением

научных и инженерных проблем создания, внедрения и эффективного использования компьютерной техники и технологии во всех сферах общественной практики.

1.6 Структура информатики и ее связь с другими науками

Структура современной информатики включает три составные части, каждая из которых может рассматриваться как относительно самостоятельная научная дисциплина (рис. 1.1).

Рис. 1.1 — Структура информатики

Теоретическая информатика — часть информатики, занимающаяся изучением структуры и общих свойств информации и информационных процессов, разработкой общих принципов построения информационной техники и технологии. Она основана

на использовании математических методов и включает в себя такие основные математические разделы, как теория алгоритмов и автоматов, теория информации и теория кодирования, теория формальных языков и грамматик, исследование операций и др.

Средства информатизации (технические и программные) — раздел, занимающийся изучением общих принципов построения вычислительных устройств и систем обработки и передачи данных, а также вопросов, связанных с разработкой систем программного обеспечения.

Информационные системы и технологии — раздел информатики, связанный с решением вопросов по анализу потоков информации, их оптимизации, структурированию в различных сложных системах, разработкой принципов реализации в данных системах информационных процессов.

Иногда информационные технологии называют компьютерными технологиями или прикладной информатикой. Само слово «компьютер» произошло от английского computer, переводимого на русский язык как «вычислитель», или электронная вычислительная машина — ЭВМ.

Технические (аппаратные) средства, или аппаратура компьютеров, в английском языке обозначаются словом hardware, которое переводится как «твердые изделия». Для обозначения программных средств, под которыми понимается совокупность всех программ, используемых компьютерами, и область деятельности по их созданию и применению, используется слово software (в переводе — «мягкие изделия»), которое подчеркивает способность программного обеспечения модифицироваться, приспосабливаться и развиваться.

Изучением закономерностей и форм движения информации в обществе, возникающих в современном обществе информационных, психологических, социально-экономических проблем и методов их решения, занимается новое направление исследований в области информатики — социальная информатика.

Информатика — очень широкая сфера научных знаний, возникшая на стыке нескольких фундаментальных и прикладных дисциплин.

Фундаментальная наука — наука, изучающая объективные законы природы и общества, осуществляющая теоретическую

систематизацию знаний о действительности.

К фундаментальным принято относить те науки, основные понятия которых носят общенаучный характер, используются во многих других науках и видах деятельности.

Как комплексная научная дисциплина информатика связана (рис. 1.2):

с *философией и психологией*	через учение об информации и теорию познания;
с *математикой*	через теорию математического моделирования, дискретную математику, математическую логику и теорию алгоритмов;
с *лингвистикой*	через учение о формальных языках и знаковых системах;
с *кибернетикой*	через теорию информации и теорию управления;
с *физикой, химией, электроникой, радиотехникой*	через «материальную» часть компьютера и информационных систем.

Рис. 1.2 — Связь информатики с другими науками

Роль информатики в развитии общества чрезвычайно велика. Она является научным фундаментом процесса информатизации общества. С ней связано прогрессивное увеличение возможностей компьютерной техники, развитие информационных сетей, создание новых информационных технологий, которые приводят к значительным изменениям во всех сферах общества: в производстве, науке, образовании, медицине и т.д.

1.7 Общая структура процессорных устройств обработки информации и принципы фон Неймана

Со времени появления в 40-х гг. XX в. первых электронных цифровых вычислительных машин технология их производства были значительно усовершенствована. В последние годы благодаря развитию интегральной технологии существенно улучшились их характеристики, значительно снизилась стоимость. Однако, несмотря на успехи, достигнутые в области технологии, существенных изменений в базовой структуре и принципах работы вычислительных машин не произошло. Так, в основу построения подавляющего большинства современных компьютеров положены общие принципы функционирования универсальных вычислительных устройств, сформулированные еще в 1945 г. американским ученым *Джоном фон Нейманом.*

Согласно фон Нейману, для того чтобы ЭВМ была универсальным и эффективным устройством обработки информации, она должна строиться в соответствии со следующими принципами.

1. Информация кодируется в двоичной форме и разделяется на единицы (*элементы*) *информации, называемые словами.*

Использование в ЭВМ двоичных кодов продиктовано в первую очередь спецификой электронных схем, применяемых для передачи, хранения и преобразования информации. Как уже отмечалось, в этом случае конструкция ЭВМ предельно упрощается и ЭВМ работает наиболее надежно (устойчиво). Совокупности нулей и единиц (битов информации), используемые для представления отдельных чисел, команд и т.п., рассматриваются как самостоятельные информационные объекты и называются словами.

Слово обрабатывается и ЭВМ как одно целое — как машинный элемент информации.

2. Разнотипные слова информации хранятся в одной и той же памяти и различаются по способу использования, но не по способу кодирования.

Все слова, представляющие числа, команды и прочие объекты, выглядят в ЭВМ совершенно одинаково и сами по себе неразличимы. Только порядок использования слов в программе вносит различия в слова. Благодаря такому «однообразию» слов оказывается возможным использовать одни и те же операции для обработки слов различной природы, например для обработки и чисел, и команд, т.е. команды программы становятся в такой же степени доступными для отработки, как и числа.

3. Слова информации размещаются в ячейках памяти машины и идентифицируются номерами ячеек, называемыми адресами слов.

Структурно основная память состоит из перенумерованных ячеек. Ячейка памяти выделяется для хранения значения величины, в частности константы или команды. Чтобы записать слово в память, необходимо указать адрес ячейки, отведенной для хранения соответствующей величины. Чтобы выбрать слово из памяти (прочитать его), следует опять же указать адрес ячейки памяти, т.е. адрес ячейки, в которой хранится величина или команда, становится *машинным идентификатором* (именем) величины и команды. Таким образом, единственным средством для обозначения величин и команд в ЭВМ являются адреса, присваиваемые величинам и командам в процессе составления программы вычислений. При этом выборка (чтение) слова из памяти не разрушает информацию, хранимую в ячейке. Это позволяет любое слово, записанное однажды, читать какое угодно число раз, т.е. из памяти выбираются не слова, а копии слов.

4. Алгоритм представляется в форме последовательности управляющих слов, называемых командами, которые определяют наименование операции и слова информации, участвующие в операции. Алгоритм, представленный в терминах машинных команд, называется программой.

В общем случае алгоритм в ЭВМ представляется в виде упорядоченной последовательности команд следующего вида:

$$\underbrace{bb...b}_{\text{КОП}}\ \underbrace{bb...b}_{A_1}\ \underbrace{bb...b}_{A_2}\ ...\ \underbrace{bb...b}_{A_k}$$

Здесь b — двоичная переменная, принимающая значение 0 или 1. Определенное число первых разрядов команды характеризует *код операции* (КОП). Например, операция сложения может представляться в команде кодом 001010. Последующие наборы двоичных переменных $bbb…b$ определяют адреса $A_1,…, A_к$ операндов (аргументов и результатов), участвующих в операции, заданной КОП. На рис.1.3 в общем виде представлен формат (структура) команды.

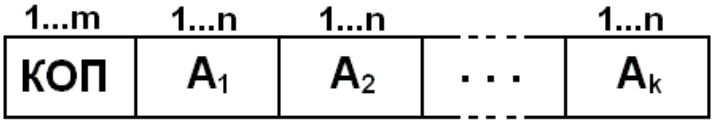


Рис. 1.3 — Общая структура команды

Составные части команды называют *полями*. Так, КОП, $A_v...A_k$ — поля команды, представляющие соответственно код операции и адреса операндов, участвующих в операции. Сверху указаны но мера разрядов полей: поле КОП состоит из m двоичных разрядов, каждое поле $A_p...A_k$ содержит n двоичных разрядов. С учетом это го представленная на рис. 1.3 команда позволяет инициировать, одну из $2^м$ операций, и каждый адрес может принимать до 2" раз личных значений, обеспечивая ссылку на любую из *2"* величин или команд. Требуемый порядок вычислений предопределяется алгоритмом и описывается последовательностью команд, образующих программу вычислений.

5. Выполнение вычислений, предписанных алгоритмом, сводится к последовательному выполнению команд в порядке, однозначно определяемом программой.

Первой выполняется команда, заданная пусковым адресом про граммы. Обычно это адрес первой команды программы. Адрес следующей команды однозначно определяется в процессе выполнения текущей команды и может быть либо адресом следующей по порядку команды, либо адресом любой другой команды.

Процесс вычислений продолжается до тех пор, пока не будет выполнена команда, предписывающая прекращение вычислений.

Необходимо подчеркнуть, что вычисления, производимые машиной, определяются программой. Именно программа «настраивает» ЭВМ на получение требуемых результатов. Замена программы приводит к изменению функций, реализуемых ЭВМ. Следовательно, многообразие программ, которые могут быть выполнены ЭВМ, определяет класс функций, который способна реализовать данная ЭВМ.

Перечисленные принципы функционирования ЭВМ предполагают, что компьютер должен иметь следующие устройства:

- *арифметико-логическое устройство* (АЛУ), выполняющее арифметические и логические операции;
- *устройство управления* (УУ), которое организует процесс выполнения программы;
- *запоминающее устройство* (ЗУ), или память для хранения программ и данных;
- *внешние устройства* для ввода (устройства ввода) и вывода (устройства вывода) информации.

При рассмотрении компьютерных устройств принято различать их *архитектуру* и *структуру*.

Под ***архитектурой ЭВМ*** понимают ее логическую организацию, состав и назначение ее функциональных средств, принципы кодирования и т.п., т.е. все то, что однозначно определяет процесс обработки информации на данной ЭВМ.

ЭВМ, построенные в соответствии с принципами фон Неймана, называют фоннеймановскими, или компьютерами фоннеймановской (классической) архитектуры.

Структура ЭВМ — совокупность элементов компьютера и связей между ними.

Ввиду большой сложности современных ЭВМ принято представлять их структуру иерархически, т.е. понятие «элемент» жестко не фиксируется. Так, на самом высоком уровне сама ЭВМ может считаться элементом. На следующем (программном) уровне иерархии элементами структуры ЭВМ являются память, процессор, устройства ввода-вывода и т.д. На более низком уровне (микропрограммном) элементами служат узлы и блоки, из которых строятся память, процессор и т.д. Наконец, на самых низких

уровнях элементами выступают интегральные логические микросхемы и электронные приборы.

Несмотря на разнообразие современных компьютеров, обобщенная структурная схема подавляющего большинства из них может быть представлена схемой, изображенной на рис..

Все устройства ЭВМ соединены линиями связи, по которым передаются информационные и управляющие сигналы, а синхронизация процессов передачи осуществляется при помощи тактовых импульсов, вырабатываемых генератором тактовых импульсов.

2 СТРУКТУРА И СОСТАВ ПЕРСОНАЛЬНОГО КОМПЬЮТЕРА

Анализ истории развития и применения средств вычислительной техники (компьютеров) показывает, что назначение этих средств состоит в сборе, накоплении, хранении, обработке и передаче (распространении) информации. Отсюда следует, что средства вычислительной техники вместе со средствами телекоммуникации являются основными инструментами реализации информационных процессов, т.е. основой новых информационных технологий.

О массовом применении компьютеров в информационных технологиях стало возможным говорить только после появления и распространения персональных компьютеров (ПК). Отличительными чертами ПК являются:

- относительно невысокая стоимость при высокой вычислительной мощности;
- компактность, малогабаритность;
- возможность гибкого изменения состава и модернизации в соответствии с решаемыми задачами;
- оснащенность программными средствами, дружественными по отношению к пользователю и основанными на средствах графического интерфейса.

В этой главе даны общие представления о структуре и составе компьютера, прежде всего, с целью грамотного и правильного (с точки зрения задач, которые предполагается решать) выбора комплекса технических средств.

Обычно выделяют так называемую *базовую конфигурацию* (состав) ПК, которая наращивается или изменяется в соответствии с конкретными потребностями. Конструктивно ПК представляет собой набор блоков (узлов), каждый из которых выбирается пользователем (покупателем), если они технически совместимы. Это означает на практике, что отдельный блок может быть удален из состава ПК и заменен на другой с соответствующим назначением, но другими характеристиками. Заметим, что на самом деле такая замена блоков требует грамотного подхода.

Конструктивно в настоящее время в базовую конфигурацию ПК входят системный блок, монитор, клавиатура, манипулятор (мышь). Системный блок представляет собой ящик (корпус — *case*), внутри которого располагаются основные узлы типовой структуры ПК.

Традиционно все устройства ПК разделяют на группы (классы) таким образом: *центральный процессор* (ЦП), содержащий *устройство управления* и *арифметико-логическое устройство; постоянное запоминающее устройство; оперативное запоминающее устройство* (ОЗУ); *внешние запоминающие устройства* (ВнЗУ); *устройства ввода* (УВвД) и *вывода данных* (УВывД) (рис. 2.1).

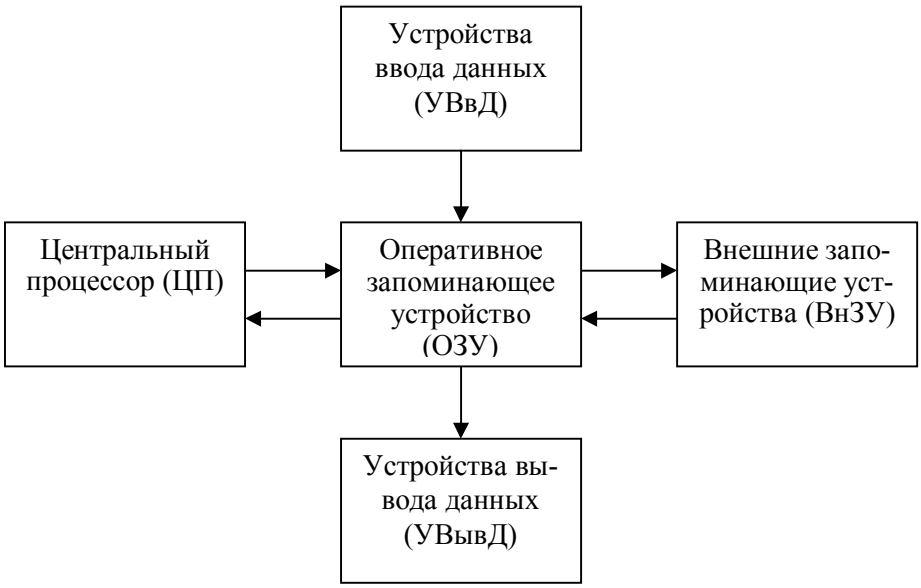


Рис. 2.1

В реальности взаимодействие устройств ПК и информационные связи между ними основаны на специальных устройствах — шинах. *Шина* содержит набор проводников для передачи сигналов и устройство (микросхемы) управления шиной. Каждая шина содержит шину адреса, шину команд и шину данных, назначение которых следует из их названия. Шины являются «узким» местом любого ПК, и скорость передачи сигналов по шинам определяет быстродействие компьютера в целом.

Типовая шинная структура современного ПК на базе процессора Pentium представлена на рис. 2.2.

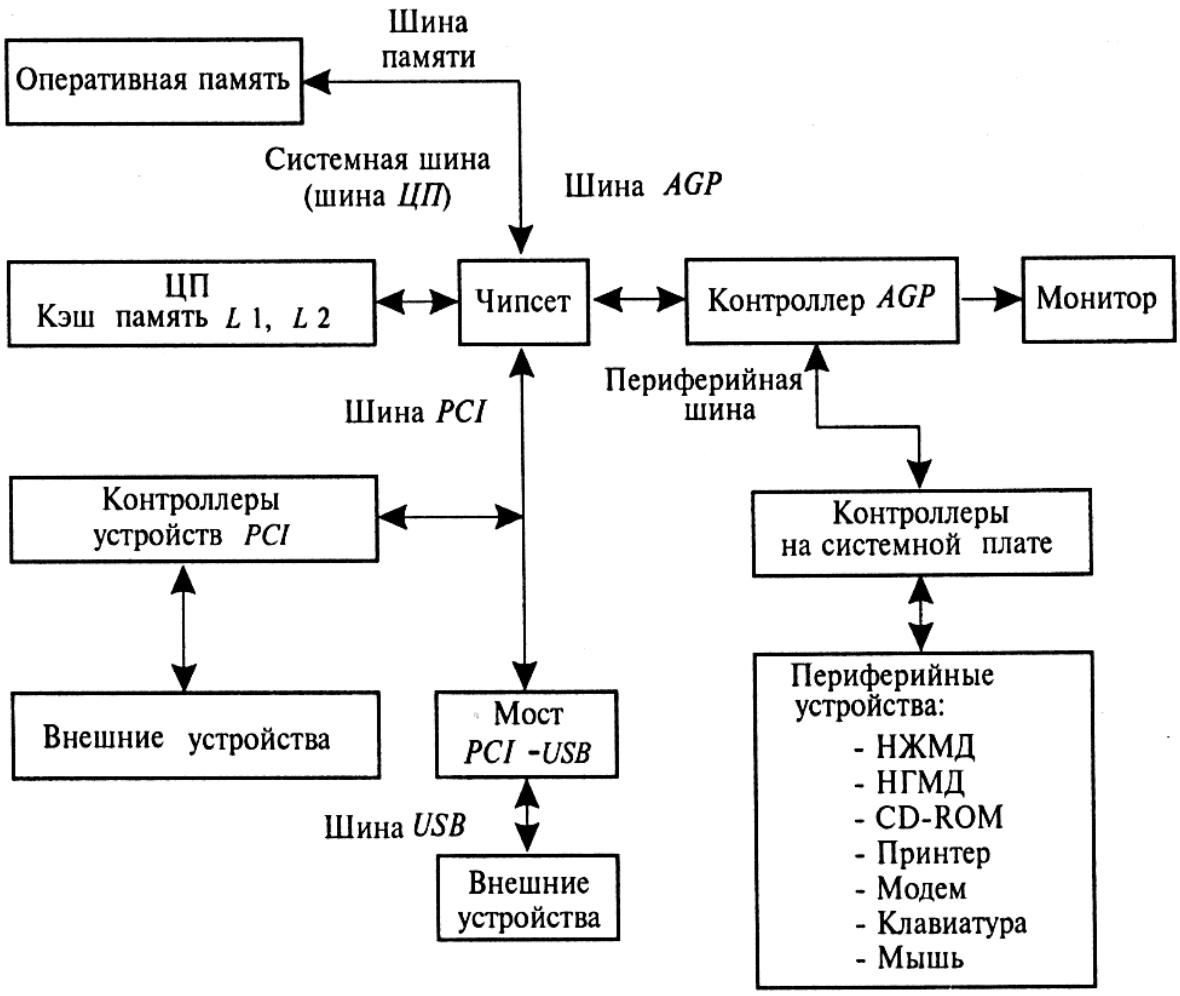


Рис. 2.2 — Типовая шинная структура современного ПК

В состав ПК сегодня входит несколько шин, отличающихся пропускной способностью (и стоимостью). Наивысшей пропускной способностью должна обладать шина, связывающая процессор с блоками памяти (внутренняя или системная шина). Характеристики других (локальных) шин определяются свойствами и характеристиками тех устройств, для подсоединения которых они используются.

Основные тенденции в развитии структуры ПК являются:

- включение в состав ПК локальных шин, обеспечивающих для высокоскоростных внешних устройств по возможности прямой (без участия центрального процессора) доступ к памяти (ОЗУ);
- включение в состав контроллеров внешних устройств (или в состав самих устройств) модулей промежуточной памяти

(кэш-памяти), которые используются для накопления информации для обработки в следующие моменты времени.

- использование периферийных процессоров в составе контроллеров внешних устройств.

Следует заметить, что технические характеристики устройств ПК постоянно совершенствуются. Поэтому приводимые в литературе данные очень быстро устаревают.

В составе многих ПК пока еще остается низкоскоростная шина *ISA* (Industry Standart Architecture), к которой подключаются звуковая и сетевая карты (старые) и порты ввода/вывода. В настоящее время разработаны и реализованы на современных материнских платах стандарты, пришедшие на смену *ISA*.

Разработан разъем (шина) *ACR* (Advanced Communications Riser) и аналогичный *CNR* (Communication and Networking Riser), которые предназначены для подключения многофункциональных плат, образующих вместе с контроллерами на материнской плате аудиосистему, сетевой адаптер и *HSP-uojxtu* (Host Signal Processing — модем, в котором обработка сигналов реализована программно, т.е. возложена на центральный процессор компьютера).

Подключение внешних запоминающих устройств (накопителей на жестких магнитных дисках — НЖМД; CD-ROM) производится через специальный интерфейс (контроллер) *IDE,* который на основе современных микросхем управления и согласования (чипсетов) обеспечивает прямой метод доступа к памяти DMA (Direct Memory Adressed) или Ultra-DMA.

В ряде случаев для подключения внешних запоминающих устройств используются высокоскоростные контроллеры интерфейса SCSI (читается «сказзи»).

Шина *PCI* первоначально была предназначена для связи процессора с оперативной памятью, в которую были врезаны разъемы для подключения внешних устройств. Пропускная способность 32-разрядной шины *PCI* на частоте 66 МГц составляла 132 Мбайт/с. Новые версии 64-разрядной шины *PCI* поддерживают частоту 66 МГц и обеспечивают пропускную способность 528 Мбайт/с. В течение нескольких лет шина *PCI* в основном использовалась для подсоединения видеосистемы.

В настоящее время вместо шины *PCI* для подсоединения видеоадаптера применяют высокоскоростную локальную шину

AGP (Advanced Graphic Port), которая работает на частоте 33 или 66 МГц и обладает пропускной способностью до 1066 Мбайт/с.

Наконец, все больше внешних устройств ввода/вывода подсоединяются к универсальной последовательной шине *USB* (Universal System Bus). Шина *USB* позволяет подключить к компьютеру до 256 различных устройств, имеющих последовательный интерфейс. Устройства соединяются последовательно (цепочкой), т.е. каждое следующее устройство подключается к предыдущему. Шина *USB* практически исключает конфликты между различными устройствами.

Таким образом, шинная структура ПК постоянно развивается и совершенствуется по пути увеличения пропускной способности в соответствии с совершенствованием других устройств ПК. Электронные устройства, обеспечивающие сопряжение локальных шин компьютера и реализующие методы доступа к памяти компьютера со стороны внешних устройств, представляют собой набор микросхем — *чипсет.* Свойства чипсета полностью определяют характеристики материнской (системной) платы ПК. В большинстве случаев чипсет состоит из двух микросхем, которые называют «*северный мост*» и «*южный мост*».

«Северный мост» управляет взаимодействием основных устройств: центрального процессора, оперативной памяти, порта *AGP* и шины *PCI.*

«Южный порт» выполняет функции контроллера дисководов жестких и гибких дисков, функции моста *PCI—ISA,* контроллера клавиатуры, мыши, шины *USB.*

Электронные мосты предназначены для согласования характеристик шины с нагрузкой (с подключаемыми к ней устройствами) или согласования шин различных типов. Структурная схема ПК показывает, что возможности компьютера существенно определяются характеристиками системной шины *FSB* (Front Side Bus) в основном тактовой частотой ее работы. Эта шина работает на частоте 66, 100 и 133 МГц. В настоящее время наиболее распространены чипсеты, поддерживающие частоту работы системной шины 133 МГц, однако имеются чипсеты, поддерживающие частоту 800 МГц системной шины. Пропускная способность шины *FSB* на частоте 100 МГц равна примерно 800 Мбайт/с. Конеч-

но, необходимо учитывать способность процессора и модулей ОЗУ работать с такими системными шинами.

Характеристики ПК определяются в основном характеристиками процессора и чипсета. Поскольку конструктивно чипсет является неотъемлемой частью материнской платы, а процессор является съемным устройством, постольку характеристики ПК определяются сочетанием материнской платы и процессора, которые должны быть совместимы. Большинство современных материнских плат автоматически определяют тип установленного процессора и производят необходимые настройки, обеспечивающие их совместимость. Некоторые возможности настройки можно выполнить программным путем через меню установки Setup BIOS.

Следует отметить стремление разработчиков ПК интегрировать на системной плате контроллеры ряда основных внешних устройств (возможно, вместе с самими устройствами), таких как видеоконтроллер, звуковая система, сетевой адаптер, факс-модем. Это приводит к существенному упрощению конструкции ПК. Однако у этой тенденции есть и существенные недостатки:

- низкая ремонтопригодность — выход из строя любого контроллера приводит к необходимости замены всей системной платы; в этом отношении модульная структура, в которой каждому контроллеру соответствует от дельная печатная плата, подсоединяемая к материнской (системной) плате через соответствующий разъем, представляется более рациональной;
- невозможность совершенствования (замены) отдельных устройств без замены всей материнской платы.

Однако рост интеграции элементов, уменьшение их габаритов при совершенствовании их характеристик и снижении цены приводит к широкому распространению интегрированных материнских плат, особенно в массовых недорогих ПК.

Объявленная фирмой IBM в самом начале развития ПК *открытость системы,* т.е. опубликование протоколов совместимости различных устройств, привела к массовому производству различными фирмами как контроллеров периферийных устройств, так и самих периферийных устройств, обладающих различными характеристиками. Это способствовало быстрому про-

грессу в области IBM-подобных ПК, быстрому росту их возможностей и производительности.

Можно выделить три основных направления развития ПК:

- прогресс в разработке и производстве процессоров: увеличение тактовой частоты в соответствии с совершенствованием технологии производства интегральных схем;
- совершенствование архитектуры процессора;
- развитие шинной структуры ПК и разработка новых поколений чипсетов;
- разработка новых периферийных устройств и соответствующих контроллеров.

Рассмотрим более подробно указанные направления развития ПК.

2.1 Процессоры

Главным элементом компьютера является центральный процессор (ЦП), который представляет собой выращенный по специальной технологии кристалл кремния, содержащий в себе множество отдельных элементов — транзисторов. Процессор включает в себя несколько устройств: арифметико-логическое устройство («вычислитель»); сопроцессор — устройство для выполнения операций «с плавающей точкой»; устройство управления; кэш-память — сверхбыстрая память, предназначенная для хранения промежуточных результатов.

Лидирующее положение в области производства процессоров давно и прочно занимает фирма «Intel». Однако наличие конкурентов в лице фирм AMD и «Cyrix» стимулирует быстрое развитие этой отрасли.

Процессоры отличаются друг от друга типом (моделью) и тактовой частотой. Каждой модели процессора соответствует набор команд, выполняемых процессором. Чем выше тактовая частота, тем выше при прочих равных условиях производительность (и цена) процессора. Тактовая частота, измеряемая в мегагерцах (МГц), показывает, сколько элементарных операций (тактов) выполняется процессором за одну секунду. Следует заметить, что разные модели процессоров выполняют одни и те же операции

(например, деление или умножение) за разное число тактов, что существенно сказывается на их производительности.

Совершенствование внутренней структуры (архитектуры) процессоров происходит параллельно с ростом тактовой частоты их работы. Серия процессоров Pentium фирмы «Intel» занимает доминирующее место на рынке процессоров. Для процессоров Pentium IV в настоящее время характерными являются частоты 1500...2800 МГц.

Важной характеристикой процессора является наличие встроенной (промежуточной, буферной) кэш-памяти второго уровня (L2), которую так называют в отличие от кэш-памяти (внутренних регистров памяти) первого уровня (L1). Объем внутренней кэш-памяти второго уровня составляет от 128 до 512 КБ.

Конструктивной особенностью, на которую необходимо обращать внимание, является тип разъема (гнезда), в который вставляется микросхема процессора на материнской (системной) плате. К сожалению, здесь достаточно большое разнообразие вариантов. Сегодня наиболее распространены материнские платы с разъемом для процессора типа FC-PGA, для процессоров Pentium III, Pentium IV и Celeron новой серии (технология 0,18 мкм). Ранее выпускались материнские платы с разъемом Slot1 для процессоров Pentium II, Pentium III, некоторых типов Celeron, и разъемом типа Slot370 для процессоров Celeron. Фирмы-производители предлагают специальные устройства — переходники с одного типа разъема на другой.

2.2 Чипсеты

Характеристики чипсета (набора микросхем) полностью определяют свойства материнской платы. При выборе материнской платы необходимо выяснить, какие возможности предоставляет чипсет, установленный на ней, в частности, какую частоту работы системной шины он поддерживает, какую скорость обмена данными с дисковыми накопителями он обеспечивает.

2.3 Контроллеры периферийных устройств

Значительное влияние на характеристики ПК в целом оказывает видеосистема, включающая в себя видеоконтроллер (*видеокарту*) и *монитор*.

Видеокарта содержит собственно контроллер монитора и видеопамять. Современные видеокарты способны хранить значительный объем данных, предназначенных для вывода на экран монитора, и в определенной степени обрабатывать их для ускорения формирования изображения. Такие видеокарты получили название *графических ускорителей* (*акселераторов*).

Свойства и возможности графического ускорителя определяются типом *графического процессора,* установленного в нем, и объемом видеопамяти. Из общих соображений ясно, что чем больше объем видеопамяти, тем лучше, что существенно сказывается на цене видеокарты.

Графические процессоры постоянно совершенствуются по пути роста внутренней частоты и развития применяемых алгоритмов обработки трехмерной графики.

Большинство выпускаемых в настоящее время графических ускорителей подключаются к порту (шине) *AGP* (Accelerated Graphic Port), однако достаточно распространены также акселераторы (иногда в виде дополнительной платы, подсоединяемой к обычной видеокарте), работающие с шиной *PCI.*

Видеоконтроллеры, расположенные на интегрированных системных платах, используют для работы с графическими данными часть оперативной памяти ПК и ресурсы центрального процессора, что может существенно сказаться на производительности системы.

Контроллеры других внешних устройств (звуковой системы, сети, модемы) обычно совмещаются с самими устройствами на одной печатной плате или интегрированы (размещены) на системной плате ПК.

2.4 Модули оперативной памяти

На всех современных материнских платах имеется не менее двух разъемов для установки *модулей оперативной памяти*

DIMM (168 контактов). Эти модули не требуют установки их одинаковыми парами, как было ранее. Один модуль может обладать объемом памяти 16, 32, 64, 128 Мбайт и более. Поэтому обычно достаточно одного модуля, установка которого не вызывает проблем. Принципиальной является такая характеристика модуля памяти, как время доступа. Необходимо убедиться, что чипсет материнской платы соответствует скоростным качествам модуля памяти. Не следует стремиться устанавливать высокоскоростные модули памяти, если материнская плата недостаточно современна.

2.5 Внешние запоминающие устройства

Следующей не менее важной проблемой является выбор и установка *устройств внешней памяти* (внешних запоминающих устройств или *накопителей*).

Сегодня считается обязательным для большинства ПК наличие накопителя на гибких магнитных дисках (НГМД) размером 3,5 дюйма и накопителя на жестком магнитном диске (винчестер), а также устройства для чтения данных с компакт-диска (CD-ROM).

При подключении НГМД необходимо обратить внимание на аккуратное подсоединение провода питания. При подсоединении сигнального шлейфа нужно правильно поместить его нулевой провод. На шлейфе нулевой провод помечен краской, на плате и устройстве обычно имеется соответствующая надпись. Чаще всего нулевой провод на устройстве расположен вблизи разъема питания. К НГМД шлейф подсоединяется тем разъемом, который расположен после переворота на 180° части проводников шлейфа.

При подключении винчестеров имеется несколько вариантов. Если винчестер (накопитель на жестком магнитном диске — НЖМД) один, то он с помощью перемычки устанавливается в режим Master (M). Если подключаются два НЖМД к одному разъему интерфейса IDE на материнской плате с помощью соответствующего шлейфа (с тремя разъемами), то один винчестер устанавливается в режим Master, а другой — в режим Slave (S). Если винчестеры подсоединяются к разным разъемам интерфейса IDE (IDE0 и IDE1), то они оба устанавливаются в режим Master.

Устройства, работающие с лазерными дисками (в частности, CD-ROM), которые подсоединяются к интерфейсу IDE, либо подключаются в режиме Slave тем же шлейфом, что и винчестер, либо в режиме Master отдельным шлейфом ко второму разъему (IDE1) интерфейса IDE на материнской плате.

При выборе винчестера, кроме желаемого объема памяти и цены, нужно руководствоваться возможностями чипсета материнской платы. В настоящее время почти все винчестеры имеют интерфейс UDMA-100 (Ultra Directed Memory Access). Скорость обмена данными существенно зависит от скорости вращения привода диска (5400 или 7200 об/мин).

Кроме накопителей с интерфейсом IDE/EIDE, возможно использование устройств с интерфейсом SCSI (читается: «сказзи»), которые обладают значительно более высокими скоростными характеристиками, но требуют наличия специального адаптера и значительно дороже стоят. Поэтому их применение скорее оправдано в серверах и компьютерах специального назначения.

2.6 Звуковые карты

Современная звуковая карта содержит аналого-цифровой (АЦП) и цифроаналоговый (ЦАП) преобразователи, обеспечивающие необходимые преобразования звукового сигнала, процессоры обработки звукового сигнала и усилители. Звуковой сигнал, поступающий на звуковую плату с аналогового входа (например, от микрофона), преобразуется в цифровую форму (оцифровывается) и может быть записан, в частности, на машинный носитель (диск). При воспроизведении звука выполняется обратная процедура.

Свойства и характеристики звуковых карт определяются частотой дискретизации звукового сигнала и применяемыми алгоритмами обработки и коррекции звука. Возможности, предоставляемые звуковыми картами пользователям, в основном определяются программными средствами, которые обеспечивают генерацию необходимых сигналов, содержат обширные библиотеки сигналов (в частности, звучания отдельных музыкальных инструментов). Все это вместе с соответствующим графическим интерфейсом таких программ позволяет пользователю эффективно

синтезировать необходимое звучание и, например, сочинять музыкальные произведения.

Звуковые карты подсоединяются в основном к шине *PCI,* но иногда все еще используется шина *ISA*.

2.7 Сетевые карты

Сетевые карты (адаптеры сети) преобразуют данные, поступающие с шины *PCI* (или *ISA*) для передачи их в сеть по двухпроводной линии, осуществляют согласование с применяемым кабелем и выполняют обратное преобразование при принятии от сети поступающих данных.

В локальных компьютерных сетях чаще всего применяются либо коаксиальный кабель (типа телевизионного антенного кабеля), либо кабель «витая пара» (два скрученных провода, помещенных в защитную оболочку). Для этих кабелей требуются различные разъемы, которые, естественно, должны быть на сетевой карте. Обычно на сетевой карте имеется либо один разъем для витой пары (UTP), либо два разъема (Combo) — для витой пары и коаксиального кабеля.

Выбор сетевой карты зависит от используемой в сети скорости передачи данных. В настоящее время в локальных сетях реализуются два стандарта: 100 и 10 Мбит/с.

2.8 Факс-модемы

Факс-модемы выполняют преобразование сигнала для передачи его по телефонной линии. Они выпускаются в двух исполнениях — внешние и внутренние. Внешние модемы как отдельные устройства подсоединяются к ПК через один из портов с помощью кабеля, и имеют автономное электропитание. Внутренний модем представляет собой печатную плату, вставляемую в разъем (слот) соответствующей шины на материнской плате.

2.9 Мониторы и видеокарты

Большое внимание при выборе компьютера должно быть уделено *видеосистеме,* включающей в себя видеокарту (видеоадаптер), графический ускоритель (акселератор), монитор.

Монитор — это устройство визуального отображения данных. *Мониторы* характеризуются принципом действия и размером экрана.

Размер экрана измеряется (обычно в дюймах) по диагонали между противоположными углами.

Жидкокристаллические мониторы (LCD) обладают абсолютно плоским экраном, высокой разрешающей способностью и являются наиболее перспективными. Ранее LCD мониторы применялись в основном в переносных (*notebook*) ПК, однако, сейчас они включаются в состав настольных компьютеров. Если на начальном этапе внедрения таких мониторов их распространение сдерживалось относительно высокой стоимостью, то сейчас с развитием технологии изготовления LCD-панелей цена их снизилась, практически приблизившись к цене традиционных мониторов на электронно-лучевых трубках. Выпускаются жидкокристаллические мониторы с размером экрана по диагонали 100 см и более. При этом габариты их по толщине таковы, что их можно повесить на стену. Успешно преодолен главный недостаток таких мониторов, заключавшийся в том, что естественные цвета изображения сохранялись лишь при небольших отклонениях угла зрения от нормали к экрану.

Традиционные мониторы основаны на использовании электронно-лучевых трубок (ЭЛТ). В настоящее время производство таких мониторов с размером экрана 14 дюймов по диагонали практически свернуто. Мониторы с черно-белым изображением применяются только в специальных целях (кассы, банки и т.п.). Для домашних применений в основном приобретаются цветные мониторы с размером 15" или 17" по диагонали. Следует отметить, что видимая часть экрана, которая используется для вывода изображения, меньше размера, указанного в паспорте монитора, примерно на 1 см.

Изображение на экране монитора на базе ЭЛТ получается в результате облучения светящегося (люминофорного) покрытия направленным пучком электронов. Для получения цветного изображения люминофорное покрытие содержит точки трех типов, светящихся красным (*R*), зеленым (*G*) и синим (*B*) цветами. Чтобы на экране все три луча, которые генерируют три цвета, сходились строго в одну точку, и изображение было четким, перед све-

тящимся слоем ставят маску — панель с регулярно расположенными отверстиями или щелями, выполненную, например, из вертикальных проволочек. Чем меньше шаг между отверстиями или щелями (шаг маски), тем четче и точнее получается изображение. Шаг маски измеряется в миллиметрах. В настоящее время наиболее распространены мониторы с шагом маски 0,24...0,28 мм.

Частота регенерации (обновления) изображения показывает, сколько раз в течение секунды на экране полностью сменяется изображение. Иногда частоту регенерации называют частотой кадров. Частоту регенерации измеряют в герцах (Гц). Чем выше частота, тем четче и устойчивее изображение, тем меньше утомление глаз при длительной работе. Минимальным для комфортной работы считается частота регенерации 75 Гц и выше.

Важной характеристикой монитора является класс защиты с точки зрения требований техники безопасности. Класс защиты определяется действующими стандартами, ограничивающими, прежде всего, уровень электромагнитного излучения монитора. Существуют также эргономические и экологические нормы по параметрам, определяющим качество изображения (яркость, контрастность, мерцание, антибликовые свойства покрытия).

Для профессиональных применений используются мониторы с размером экрана 17, 19, 21 дюймов по диагонали. Главной проблемой ЭЛТ большого размера являются искажения изображения по краям экрана. Для уменьшения этих искажений применяют сложные системы управления электронным пучком, формирующим изображение.

С увеличением разрешения монитора (количества точек по вертикали и горизонтали) области видеопамяти в ОЗУ оказалось недостаточно для хранения графических данных, а центральный процессор перестал справляться с построением и обновлением изображения. Поэтому операции, связанные с графическими данными, переданы отдельному устройству, получившему название *видеоадаптера* или *видеокарты*. Видеокарта вместе с монитором образует видеосистему компьютера. На видеокарту возложены функции видеоконтроллера, видеопроцессора и видеопамяти.

Известны различные поколения видеоадаптеров: MDA (монохромный), CGA (4 цвета), EGA (16 цветов), VGA (256 цветов). Видеоадаптеры SVGA обеспечивают воспроизведение до

16,7 млн цветов с возможностью выбора разрешения экрана из стандартного ряда значений: 640×480, 800×600, 1024×768, 1280×1024, 1600×1400 точек.

Разрешение экрана — один из важнейших параметров видеосистемы. Чем выше разрешение, тем больше информации можно отобразить на экране; при этом меньше становится размер каждой отдельной точки, и тем самым меньше видимый размер элементов изображения.

Для каждого размера экрана монитора, для каждой программы можно выбрать наиболее подходящее разрешение. Использование высокого разрешения на мониторе малого размера приводит к тому, что элементы изображения становятся мелкими и неразборчивыми. При использовании низкого разрешения на экранах большого размера элементы изображения становятся чрезмерно крупными. Если программа имеет сложный и насыщенный элементами интерфейс, то они не полностью умещаются на экране, что затрудняет работу.

Многие современные (особенно игровые) программы требуют высокого разрешения экрана и значительных объемов видеопамяти. В настоящее время обычным является объем 64 Мбайт, но в ряде случаев уже требуется больший объем видеопамяти.

2.10 Конфигурация системного блока компьютера

Конструкция и габариты системного блока, так называемый *форм-фактор* корпуса (*case*) являются важными с точки эксплуатационных качеств компьютера в целом. В течение длительного времени на рынке доминировали корпуса вертикального исполнения формата AT или *mini-tower* («малая башня»). Корпуса горизонтального исполнения в основном поставляют очень известные и потому дорогие фирмы (*«Brand Name»*).

Корпус формата AT достаточно компактен, занимает не слишком много места на рабочем столе. Однако недостаток места внутри корпуса приводит к трудностям при модернизации ПК и размещении в нем дополнительных устройств. Теснота внутри корпуса является причиной неудовлетворительного теплового

режима работы размещенных в нем устройств. В настоящее время выпуск корпусов формата AT практически прекращен.

Корпуса формата ATX (*midi-tower*) имеют достаточные размеры для размещения как основных, так и дополнительных устройств, в том числе дополнительных вентиляторов, что очень важно в связи с большой теплоотдачей современных процессоров. Блоки питания в корпусах ATX обычно обладают большей мощностью и обеспечивают возможности программного управления режимами питания.

Выбор размера корпуса определяет выбор форм-фактора системной (материнской) платы. В названиях (типах) материнских плат всегда присутствует указание на форм-фактор AT или ATX.

В корпусах AT и ATX вертикального исполнения материнская плата с помощью пластмассовых ножек и винтов устанавливается на специальной вертикальной (боковой) стенке, на которой имеется стандартизованный набор отверстий. Обычно плата привинчивается к стенке после установки на нее процессора с вентилятором и модулей памяти.

Блок питания в корпусе AT имеет два жгута с разъемами, которые (оба) вставляются в соответствующий разъем на материнской плате. Необходимо обратить внимание на то, чтобы черные провода в жгутах были расположены рядом друг с другом (к центру разъема на материнской плате).

В корпусе формата ATX разъем у жгута блока питания один, и подключение его к материнской плате не вызывает проблем. Выпускаются также материнские платы, которые могут быть установлены как в корпус AT, так и ATX. Соответственно на них имеется два разъема для подключения жгутов питания.

Следующим принципиальным моментом в выборе конфигурации ПК является выбор *процессора.* С учетом моды и престижности многие пользователи ориентируются на самые последние достижения науки и техники в области процессоров. В качестве обоснования обычно указывают на то, что такой компьютер не слишком быстро морально устареет.

Однако на самом деле главными факторами здесь являются круг задач, которые предполагается решать с помощью ПК, и цена. Если ПК используется только в качестве «пишущей машин-

ки», то основным фактором является его цена. Нужно учитывать также то, что, как показывают специально проводимые тесты, при решении «массовых» задач самые мощные и дорогостоящие процессоры не дают заметного преимущества перед моделями, стоящими значительно меньше.

В настоящее время оптимальным для большинства пользователей (особенно домашних) является процессор Celeron фирмы «Intel» с тактовой частотой, которая позволяет использовать наиболее современные чипсеты материнских плат. Сейчас этому соответствуют тактовые частоты больше 667 МГц (до 2400 МГц). Однако необходимо иметь в виду, что на сегодняшний день процессоры Celeron работают с частотой 66 МГц системной шины.

Необходимо обратить внимание также на наличие встроенной в процессор кэш-памяти второго уровня (*L*2) объемом не менее 128 Кбайт. Обычно (начиная с Pentium II) это условие выполняется.

Важным моментом является *тип* разъема, который необходим тому или иному процессору. Если разъем на материнской плате не совпадает с требуемым для процессора, то можно использовать специальные переходники (в виде небольших печатных плат). Также необходимо обратить внимание на тип вентилятора, охлаждающего процессор, который должен конструктивно соответствовать выбранному процессору.

Современные материнские платы в большинстве случаев автоматически «распознают» установленный процессор, и в первом приближении дополнительных настроек не требуется.

2.11 Принтеры

Поскольку подготовка документов с последующим выводом на бумагу является одной из основных областей применения ПК, в состав компьютера (домашнего или рабочего) включается *принтер***.** Существуют три основных типа принтеров: матричные, струйные и лазерные.

В *матричных принтерах* изображение на бумаге формируется при ударе печатающей головки принтера по красящей ленте. Печатающая головка содержит набор (матрицу) иголок (9 или 24), которые выдвигаются в нужной комбинации в соответствии с

сигналами от компьютера. Печатающая головка перемещается вдоль бумаги в горизонтальном направлении. Скорость печати матричных принтеров относительно невелика, и достаточно высок уровень шума при печати.

Несмотря на кажущуюся устарелость матричных принтеров, они продолжают совершенствоваться и широко применяться в связи с высоким качеством печати, относительно невысокой стоимостью и возможностью печатать несколько экземпляров документов «под копирку».

Принцип печати в *струйных принтерах* похож на тот, который используется в матричных устройствах, только вместо иголок из печатающей головки на бумагу «выстреливают» мельчайшие капли красящих чернил. Эти принтеры почти бесшумны при печати, скорость печати достаточно высока. Одним из главных достоинств струйных принтеров является простая возможность цветной печати, которая реализуется путем смешения чернил трех основных цветов (*RGB* — красный, зеленый, голубой).

Струйные принтеры наиболее предпочтительны для домашнего применения в связи с их невысокой стоимостью и высоким качеством печати. Однако нужно иметь в виду, что расходные материалы к ним (*картриджи*) достаточно дорогие.

В *лазерных принтерах* изображение на бумаге формируется в результате сложного технологического процесса, схожего с применяемым в копировально-множительных аппаратах. Луч лазера разной интенсивности в соответствии с выводимым изображением попадает на барабан, участки поверхности которого в различной степени электризуются в зависимости от освещенности. Барабан помещается в среду красящего порошка, который прилипает к нему в разной плотности в зависимости от степени его наэлектризованности. Затем барабан прокатывается по бумаге, и изображение высушивается.

Качество выводимых лазерным принтером изображений очень высоко, производительность даже у массовых принтеров составляет 5...8 страниц в минуту. Лазерные принтеры, обеспечивающие цветную печать, очень дороги и не находят пока широкого распространения.

При выборе печатающего устройства большое значение имеет стоимость расходных материалов и, прежде всего, сменных

картриджей. Картридж матричного принтера, содержащий лишь красящую ленту, очень дешев. Картриджи струйных принтеров (для черно-белой, цветной и фотопечати) достаточно дорогие (20...30 $ и выше). При ресурсе картриджа 300...1000 листов среднезаполненных листов формата А4 стоимость печати одного листа составляет 0,05...0,1 $ (примерно 15...30 коп.). Фотопечать, т.е. вывод изображений фотографического качества, обходится на порядок дороже, в том числе за счет высокой стоимости специальной бумаги.

Картриджи лазерных принтеров стоят более 50 $ и имеют ресурс около 3000 листов формата А4 (текст средней плотности) и, следовательно, стоимость печати одного листа составляет приблизительно 0,5 коп.

Картриджи струйных и лазерных принтеров могут быть вторично заправлены (соответственно чернилами и порошком), что снижает затраты на печать.

Итак, рассмотрены логическая структура и состав персонального компьютера. Приведены основные характеристики устройств ПК. Основное внимание уделено тенденциям развития технических средств и принципам выбора конфигурации компьютера.

3 КОМПОНЕНТЫ ПРИКЛАДНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

3.1 Текстовой процессор Microsoft Word

Текстовыми процессорами называют программные средства, предназначенных для создания, редактирования и форматирования простых и комплексных текстовых документов. В настоящее время в России наибольшее распространение имеет текстовый процессор Microsoft Word. Это связано, прежде всего, с тем, что его создатели относительно давно предусмотрели локализацию программы в России путем включения в нее средств поддержки работы с документами, исполненными на русском языке.

Первоначальные версии текстового процессора Microsoft Word относятся к восьмидесятым годам и, соответственно, к операционной системе MS-DOS. Поскольку MS-DOS не является графической, в первых версиях программы не было возможности реализовать принятый ныне принцип соответствия экранного изображения печатному (принцип WYSIWYG, сокращение от англ. *What You See Is What You Get* — «что ты видишь, то ты получишь).

Современная версия текстового процессора — Microsoft Word XP (Word 10.0), входящая в состав пакета Microsoft Office XP работает под управлением операционной системы Microsoft Widows XP. В ней имеются развитые средства работы со стилями и шаблонами, введены механизмы, позволяющие автоматически обеспечить единство оформления документа.

3.2 Рабочее окно процессора Microsoft Word

Рабочее окно процессора Microsoft Word XP имеет следующие основные элементы управления: *строка меню, панель инструментов, рабочее поле* и *строка состояния*, включающая *индикаторы*. Панель инструментов является настраиваемой.

3.3 Режимы отображения документов

Текстовый процессор Microsoft Word поддерживает несколько режимов представления документов.

В режиме *Обычный* представляется только содержательная часть документа без реквизитных элементов оформления, относящихся не к тексту, а к печатным страницам (колонтитулы, колонцифры, подстраничные сноски и т.п.). Этот режим удобен на ранних этапах разработки документа (ввод текста, редактирование, рецензирование), а также во всех случаях, когда содержательная часть документа имеет более высокое значение, чем внешнее представление. В этом режиме операции с объемными документами проходят быстрее, что важно при работе на малопроизводительных компьютерах.

В режиме *Web-документ* экранное представление не совпадает с печатным. Это отступление от принципа WYSIWYG, но оно характерно для электронных публикаций в World Wide Web, поскольку заранее не известно, каким средством просмотра и на каком оборудовании будет отображаться документ. Понятие печатной страницы для электронных документов не имеет смысла, поэтому назначенные параметры страницы не учитываются, а форматирование документа на экране является относительным. В этом режиме разрабатывают электронные публикации.

В режиме *Разметка страницы* экранное представление документа полностью соответствует печатному, вплоть до назначенных параметров печатной страницы. Этот режим удобен для большинства работ, связанных с форматированием текста, предназначенного для печати.

В режиме *Структура* можно отобразить только заголовки документа. Режим полезен в тех случаях, когда разработку документа начинают с создания плана содержания. Если предполагаемый размер документа превышает 5—7 печатных страниц следует начинать работу именно с создания первичного плана. Режим структуры отличается тем, что при его включении автоматически открывается вспомогательная панель инструментов *Структура*, элементы управления которой позволяют править структуру документа.

Выбор одного из четырех указанных режимов представления документа выполняют с помощью командных кнопок, расположенных в левом нижнем углу окна приложения, или командами меню *Вид*.

Через меню *Вид* доступно также специальное представление

(пятый режим) *Схема документа*, при котором окно приложения имеет две рабочие панели. На левой панели представляется структура документа, а на правой — сам документ. Этот режим, сочетающий достоинства режима разметки и режима структуры, полезен при навигации по объемному документу — его удобно использовать не при создании, а при просмотре документов сложной структуры.

Через меню *Файл* доступны еще два режима представления документа, используемые для предварительного просмотра. Для электронных документов используют команду *Файл* ► *Предварительный просмотр Web-страницы,* а для печатных документов — *Файл* ► *Предварительный просмотр*. В первом случае созданный документ отображается как Web-страница в окне браузера, зарегистрированного операционной системой в качестве принятого по умолчанию (желательно, чтобы это был браузер Microsoft Internet Explorer 6.0). Во втором случае документ представляется в специальном окне.

3.4 Приемы работы с командами строки меню

Как и в большинстве других приложений, корректно соблюдающих идеологию Windows, строка меню текстового процессора Microsoft WordXP как элемент управления отличается тем, что обеспечивает доступ ко всем функциональным возможностям программы. Не всегда этот доступ самый удобный, во многих случаях другие элементы управления использовать проще, но строка меню удовлетворяет принципу функциональной полноты.

Меню, открывающиеся из строки меню, обладают свойством функциональной автонастройки. Расширенные возможности приложения не могли не отразиться в изобилии элементов управления, открываемых через строку меню. В нем не всегда удобно ориентироваться. Поэтому пункты строки меню открываются в два приема. На первом этапе открывают сокращенное меню, и, если необходимого элемента управления в нем нет, открывают расширенное меню наведением указателя мыши на пункт раскрытия. Использованные пункты расширенной части меню далее открываются в составе сокращенного меню.

3.5 Панели инструментов Microsoft Word XP

Программа Microsoft Word поддерживает возможность самостоятельной настройки панелей инструментов. Настройку выполняет пользователь путем подключения функциональных панелей, необходимых ему по роду деятельности (*Вид* ► *Панели инструментов*). Расширение общей панели инструментов сопровождается некоторым уменьшением площади рабочего окна документа. Перемещение функциональных панелей производят методом перетаскивания за рубчик, расположенный на левом краю панели.

В последних версиях текстового процессора панели инструментов не только допускают настройку, но и обладают контекстной чувствительностью. Так, при выделении в поле документа какого-либо объекта, автоматически открывается панель инструментов, предназначенная для его редактирования. Назначение панелей инструментов приведено в таблице 3.1.

Таблица 3.1 — **Панели инструментов программы Word XP**

Панель инструментов	Состав, назначение	Примечание
Стандартная	Элементы управления файловыми операциями, редактированием, экранным отображением	Устанавливается по умолчанию
Форматирование	Элементы управления форматированием документа	Устанавливается по умолчанию
Visual Basic	Доступ к средствам создания и редактирования макросов и Web-сценариев, а также к настройке средств обеспечения безопасности при запуске макросов	Макросы служат для автоматизации типовых операций. Web-сценарии обеспечивают динамичный характер просмотра Web-страниц
Word-Art	Элементы управления для создания художественных заголовков	

Продолжение табл. 3.1

Панель инструментов	Состав, назначение	Примечание
Автотекст	Средство быстрого доступа к настройке функции автотекста	Одновременно предоставляет быстрый доступ к средствам настройки функций автозамены и автоформата
База данных	Элементы управления, характерные для работы с базами данных (сортировка, поиск, управление структурой таблиц и прочее)	В качестве базы данных могут выступать как таблицы Access, так и собственные таблицы Word
Веб-компоненты	Комплект готовых компонентов для создания элементов управления Web-страницы или электронной формы	Применяются для создания обратной связи с потребителем документа (опросные листы, анкеты, бланки заказов и прочее)
Веб-узел	Элементы управления для навигации в Web-структурах данных	В качестве Web-структур могут выступать World Wide Web, корпоративные сети intranet, системы Web-документов локального компьютера
Настройка изображения	Элементы управления для основных функций настройки растровых изображений	Позволяют настраивать яркость, контрастность, размер, рамку, режимы обтекания текстом и прочие параметры выделенного растрового объекта
Рамки	Элементы управления для создания **фреймов** ВНИМАНИЕ! — *не путать с рамками, создаваемыми с помощью панели инструментов Таблицы и границы*!	Текстовый процессор Word XP поддерживает два типа **фреймов**. Фреймы в электронных документах представляют собой особые прямоугольные области, предназначенные для вывода нескольких Web-документов в рамках одной Web-страницы.

Продолжение табл. 3.1

Панель инструментов	Состав, назначение	Примечание
Рецензирование	Элементы управления для проведения редактирования и комментирования документов без искажения исходного текста	Измененные данные сохраняются в том же документе на правах новых версий. Автор исходного текста имеет возможность просмотреть замечания и предлагаемые изменения, после чего принять их или отвергнуть
Рисование	Элементы управления и инструменты для выполнения простейших чертежно-графических работ	Графические объекты, создаваемые инструментами данной панели, имеют характер векторных объектов
Слияние	Инструменты для работы с документами слияния, содержащими постоянную и переменную части	Используется при использовании программы Word XP, например, для массовой подготовки писем аналогичного содержания
Статистика	Позволяет получить информацию об объеме документа	Сведения о числе знаков, слов, строк, абзацев, страниц
Структура	Инструменты для работы с логической структурой документа	Позволяет управлять заголовками и порядком следования логических частей текста. Активно, используется при работе с документом в режиме структуры
Таблицы и границы	Элементы управления для создания таблиц и оформления текстовых блоков рамками	Дополнительно предоставляет средства для сортировки данных и проведения итоговых расчетов в таблицах (функция Автосумма)
Формы	Элементы управления для разработки стандартных форм	Программа Word XP позволяет создавать три типа форм: Web-формы, являющиеся объектами Web-страниц; формы Word, распространяемые и

Окончание табл. 3.1

Панель инструментов	Состав, назначение	Примечание
		заполняемые как электронный документ; печатные формы
Элементы управления	Набор готовых компонентов ActiveX для создания элементов управления Web-страниц и Web-форм	Средства данной панели инструментов позволяют не только использовать около 150 готовых компонентов, но и проводить установку и регистрацию дополнительных компонентов ActiveX

В программе WordXP роль специальной контекстно-зависимой информационно-инструментальной панели играет область задач. Эта область открывается на правах панели инструментов и обычно располагается у правого края окна программы. Область задач может использоваться для выполнения разных функций, в зависимости от выбранного режима. Выбор режима осуществляется из меню, открывающегося при щелчке на треугольной кнопке в верхней строке *Область задач*. Кроме того, существуют специальные команды для открытия области задач в нужном режиме (например, *Правка* ► *Буфер обмена Office или Формат* ► *Показать форматирование*). Режимы работы области задач описаны в таблице 3.2.

Таблица 3.2 — **Режимы области задач**

Режим	**Команда**	**Содержание области задач**	**Назначение**
Создание документа	Файл > Создать	Список недавно открывавшихся документов, команды создания новых документов	Открытие существующих и создание новых документов
Буфер обмена	Правка ► Буфер обмена Office	Содержание буфера обмена Office (до 24. объектов)	Выбор объектов для вставки в документ
Поиск	Файл > Найти	Команды для поиска и информация о результатах поиска	Поиск текста в форматированных файлах
Вставка картинки	Вставка ► Рисунок ► Картинки	Команды для поиска клипартов и информация о результатах поиска	Выбор клипартов и других графических изображений для вставки в документ

Окончание табл. 3.2

Режим	Команда	Содержание области задач	Назначение
Стили и форматирование	Формат > Стили и форматирование	Сведения о стилях и форматировании текста и средства для их изменения	Выбор и создание стилей на основе существующего оформления текста
Показать форматирование	Формат > Показать форматирование	Сведения о характеристиках форматирования в месте расположения курсора	Информация о форматировании, сравнение форматов разных фрагментов
Слияние	Сервис > Письма и рассылки ► Мастер слияния	Этапы создания документа слияния	Создание документов слияния (например, писем), содержащих постоянную и переменную части
Перевод	Сервис > Язык > Перевод	Исходный текст и результат перевода	Перевод отдельных слов и коротких фраз

3.6 Форматирование текста

Форматирование текста осуществляется средствами меню Формат или панели Форматирование. Основные приемы форматирования включают:

1) выбор и изменение гарнитуры шрифта;
2) управление размером шрифта;
3) управление начертанием и цветом шрифта;
4) управление методом выравнивания;
5) создание маркированных и нумерованных списков (в том числе многоуровневых);
6) управление параметрами абзаца.

3.6.1 Настройка шрифта

При выборе гарнитуры шрифта следует иметь в виду следующие обстоятельства.

Выбор гарнитуры шрифта действует на выделенный текстовый фрагмент. Если ни один фрагмент не выделен, он действует на весь вводимый текст до очередной смены гарнитуры.

Текстовый процессор Microsoft Word XP ориентирован на работу с многоязычными шрифтовыми наборами (UNICODE). При использовании других шрифтовых наборов возможны проблемы, которые возникают при переключении раскладки клавиатуры с основной (английской) на дополнительную (русскую). В таком случае возможен неконтролируемый автоматический воз-

врат к использованию одного из стандартных шрифтов UNICODE, зарегистрированных в операционной системе.

Напомним, что как операционная система Windows XP, так и сам текстовый процессор Microsoft Word поставляются с наборами шрифтов UNICODE, то есть использование шрифтов, входящих в стандартную поставку, является гарантией от непредвиденных осложнений.

Настройку шрифта выполняют в диалоговом окне *Шрифт* (*Формат* ► *Шрифт*). В версии Microsoft Word XP данное диалоговое окно имеет три вкладки: *Шрифт, Интервал* и *Анимация.*

На вкладке *Шрифт* выбирают:

- гарнитуру шрифта;
- его размер (измеряется в полиграфических пунктах);
- вариант начертания;
- цвет символов;
- наличие подчеркивания;
- характер видоизменения.

При выборе гарнитуры шрифта следует иметь в виду,.что существует две категории шрифтов: с засечками и без засечек (рубленые). Характерными представителями первой категории являются шрифты семейства Times, а второй категории — шрифты семейства Arial. Шрифты, имеющие засечки, легче читаются в больших текстовых блоках — их рекомендуется применять для оформления основного текста.

Шрифты, не имеющие засечек, рекомендуется использовать для заголовков в технических текстах, а также для оформления дополнительных материалов (врезок, примечаний и прочего).

Кроме того, считается, что шрифты с засечками лучше воспринимаются в документах, напечатанных на бумаге. Для электронных документов, которые предполагается читать с экрана, многие предпочитают применять рубленые шрифты.

Большинство гарнитур шрифтов являются пропорциональными. Это означает, что и ширина отдельных символов, и расстояние между соседними символами не являются постоянными величинами и динамически меняются так, чтобы сопряжение символов было наиболее благоприятным для чтения.

Особую группу представляют так называемые моноширинные шрифты. В них каждый символ вместе с окаймляющими его

интервалами имеет строго определенную ширину. Такие шрифты применяют в тех случаях, когда надо имитировать шрифт пишущей машинки, а также при вводе текстов, представляющих листинги программ. Характерными представителями таких шрифтов являются шрифты семейства Courier.

При выборе размера шрифта руководствуются назначением документа, а также вертикальным размером печатного листа. Для документов, имеющих формат типовой книжной страницы, обычно применяют шрифт размером 10 пунктов. Для документов, готовящихся для печати на стандартных листах формата А4 (210×297 мм), выбирают размер 12 пунктов. При подготовке документов, предназначенных для передачи средствами факсимильной связи, применяют увеличенный размер — 14 пунктов и больше (факсимильные документы часто воспроизводятся с искажениями, и увеличенный размер шрифта улучшает удобство их чтения).

При подготовке электронных документов, распространяемых в формате Microsoft Word, размер шрифта выбирают, исходя из разрешения экрана. В настоящее время наиболее распространены компьютеры, видеосистема которых настроена на экранное разрешение 800×600 точек или 1024×768 точек. Для этих параметров целесообразно готовить электронные документы с размером шрифта 12 пунктов. На этот размер по умолчанию настроены последние версии процессора Microsoft WordXP.

Из прочих, не рассмотренных здесь средств управления шрифтами надо отметить управление интервалом между символами и возможность использования эффектов анимации. Интервал задается путем выбора одного из трех значений (Обычный, Разреженный, Уплотненный) на вкладке Формат ► Шрифт ► Интервал.

Эффекты анимации используют очень редко и только при подготовке электронных документов, распространяемых в формате текстового процессора. В печатных документах эти эффекты невоспроизводимы по очевидным причинам, а в Web-документах их нет смысла применять, так как они пока не поддерживаются Web-браузерами.

3.6.2 Настройка метода выравнивания

Все последние версии текстового процессора Microsoft Word поддерживают четыре типа выравнивания:

1) по левому краю;
2) по центру;
3) по правому краю;
4) по ширине.

Выбор метода выполняют соответствующими кнопками панели инструментов Форматирование или из раскрывающегося списка *Формат* ► *Абзац* ► *Отступы и интервалы* ► *Выравнивание.* Избранный метод действует на текущий и последующие вводимые абзацы. Выбор метода выравнивания определяется назначением документа. Так, например, для Web-страниц нет смысла выполнять выравнивание по ширине, поскольку все равно неизвестна ширина окна браузера, в котором документ будет просматриваться, однако выравнивание по центру использовать можно.

Для документов, передаваемых на последующую обработку, все методы выравнивания, кроме тривиального выравнивания по левому краю, являются излишними. Для печатных документов, выполненных на русском или немецком языках, рекомендуется в основном тексте использовать выравнивание по ширине с одновременным включением функции переноса, а для документов на английском языке основной метод выравнивания — по левому полю.

3.6.3 Настройка параметров абзаца

Кроме режима выравнивания настраиваются следующие параметры абзаца:

- величина отступа слева (от левого поля);
- величина отступа справа(от правого поля);
- величина отступа первой строки абзаца («красная строка»);
- величина интервала (отбивки между абзацами) перед абзацем и после него.

Для печатных документов величину отступа для основного текста, как правило, не задают (необходимое положение текста

определяется шириной полей), но ее задают для дополнительных материалов и заголовков, если они не выравниваются по центру. В то же время, для Web-страниц величина отступа для абзацев имеет большое значение. Это один из весьма немногих параметров форматирования, допускаемых для Web-документов, поэтому его используют очень широко.

Роль отбивок между абзацами, как и роль отступа первой строки абзаца, состоит в том, чтобы визуально выделить абзацы. При этом следует помнить, что эти средства несовместимы. То есть, применяя отступ первой строки абзаца, не следует применять отбивки между абзацами, и наоборот. Комбинация этих стилей допускается только для маркированных и нумерованных списков (основной текст оформляется с отступом первой строки, а списки — без него, но с отбивкой между абзацами).

Обычная практика назначения формата состоит в том, что для документов простой структуры (художественных) используют отступ первой строки (это особенно важно для текстов на русском и немецком языках), а для документов сложной структуры (технических) и документов на английском языке используют отбивки между абзацами. Промежуточное положение занимают документы, относящиеся к естественнонаучным и гуманитарным дисциплинам, — при их подготовке кроме точки зрения автора руководствуются сложившейся практикой и устоявшимися традициями.

В Web-документах применяют только отбивки между абзацами. Отступ первой строки в них обычно не используют в связи с повышенными трудностями его создания.

3.6.4 Средства создания маркированных и нумерованных списков

Специальное оформление маркированных и нумерованных списков редко применяют в художественных документах и персональной переписке, но в служебных документах и особенно в Web-документах оно используется очень широко. В Web-документах оформление маркированных списков особо усиливают за счет применения специальных графических маркеров, стиль которых должен тематически сочетаться с содержанием и оформ-

лением документов.

Для создания нумерованных и маркированных списков нужно сначала выполнить настройку, затем вход в список и, наконец, выход из него. Настройку выполняют в диалоговом окне Список, открываемом командой *Формат* ► *Список*. Данное окно имеет четыре вкладки: *Маркированный список*, *Нумерованный список*, *Многоуровневый список* и *Список стилей*. В качестве элементов управления здесь представлены образцы оформления списков. Для выбора нужного достаточно щелкнуть на избранном образце.

Вход в список может осуществляться автоматически или по команде. Чтобы автоматически создать маркированный список, достаточно начать запись строки с ввода символа «*». По завершении строки и нажатии клавиши ENTER символ «*» автоматически преобразуется в маркер, а на следующей строке маркер будет установлен автоматически. Для автоматического создания нумерованного списка достаточно начать строку с цифры, после которой стоят точка и пробел, например «1.», «2.» и т.д. Этот метод позволяет начать нумерацию с любого пункта (не обязательно с единицы).

Для создания списка по команде служат кнопки *Нумерация* и *Маркеры*, представленные на панели *Форматирование*. Как маркированный, так и нумерованный список легко превратить в многоуровневый. Для перехода на новые (или возврата на предшествующие уровни) служат кнопки *Увеличить отступ* и *Уменьшить отступ* на панели *Форматирование*.

Для списков с очень глубоким вложением уровней (более трех) можно настроить стиль оформления каждого из уровней. Для этого служит командная кнопка *Изменить* на вкладке *Многоуровневый* диалогового окна *Список* (*Формат* ► *Список*).

Вкладка *Список* стилей также предназначена для оформления многоуровневых списков. Она позволяет выбрать или определить для каждого уровня списка особый стиль. О работе со стилями мы поговорим чуть позже.

Характерной особенностью процессора Microsoft WordXP, связанной с его ориентацией на создание Web-документов, является возможность использования графических маркеров. Для выбора такого маркера на вкладке *Маркированный* диалогового окна *Список* (*Формат* ► *Список*) выберите один из образцов спи-

ска и щелкните на кнопке *Изменить*. Откроется диалоговое окно Изменение маркированного списка, в котором надо щелкнуть на кнопке *Рисунок*. Эта кнопка открывает диалоговое окно *Рисованный маркер*, в котором можно выбрать подходящее изображение.

Для завершения маркированного или нумерованного списка и выхода из режима его создания достаточно по завершении ввода последней строки дважды нажать клавишу ENTER.

3.7 Приемы и средства автоматизации разработки документов

С рядом приемов автоматизации ввода и редактирования текста мы познакомились выше. К ним относятся средства Автотекст, Автозамена, средства проверки правописания, средства расстановки переносов, средства поиска и замены фрагментов текста.

В этом разделе мы познакомимся с наиболее общими средствами автоматизации разработки и оформления документов, к числу которых относятся стили оформления абзацев, шаблоны документов и темы оформления.

3.8 Работа со стилями

Абзац — элементарный элемент оформления любого документа. Каждый заголовок документа тоже рассматривается как отдельный абзац. Выше мы видели, что в меню Формат > Абзац имеется немало различных элементов управления, и выполнять их настройку для каждого абзаца отдельно — неэффективная и утомительная задача. Она автоматизируется путем использования понятия стиль.

Стиль оформления — это именованная совокупность настроек параметров шрифта, абзаца, языка и некоторых элементов оформления абзацев (линий и рамок). Благодаря использованию стилей обеспечивается простота форматирования абзацев и заголовков текста, а также единство их оформления в рамках всего документа.

Особенностью текстовых процессоров Microsoft Word является то, что они поддерживают четыре типа стилей:

1) стили абзаца;
2) стили знаков (символов);
3) стили списков;
4) стили таблиц.

С помощью стилей абзаца выполняют форматирование абзацев, а с помощью знаковых стилей можно изменять оформление выделенных фрагментов текста внутри абзаца. Стиль списка предполагает наличие в начале абзаца номера или маркера. Стиль таблицы обеспечивает согласование границ, заливки, выравнивания и шрифтов в таблицах.

Наличие разных типов стилей позволяет реализовать довольно сложные приемы форматирования. Например, внутри абзаца, оформленного одним шрифтом, могут содержаться фрагменты текста, оформленные другим шрифтом. В данной книге, например, специальный шрифт использован для записи названий элементов управления.

Работа со стилями состоит в создании, настройке и использовании стилей. Некоторое количество стандартных стилей присутствует в настройке программы по умолчанию, сразу после ее установки. Их используют путем выбора нужного стиля из раскрывающегося списка на панели управления Форматирование.

Все работы по созданию новых стилей и изменению существующих выполняют с помощью Области задач в режиме Стили и форматирование. Если Область задач закрыта или находится в ином режиме, надо дать команду Формат ► Стили и форматирование.

Настройка стиля. Настройку стиля выполняют в диалоговом окне Стиль (Формат > Стиль). Настраиваемый стиль выбирают в списке Стили (при этом на панелях Абзац и Знаки отображаются образцы применения данного стиля). Для изменения стиля служит командная кнопка Изменить, открывающая диалоговое окно Изменение стиля. Каждый из компонентов стиля настраивается в отдельном диалоговом окне. Выбор компонента выполняют в меню, открываемом с помощью командной кнопки Формат.

При проведении настройки стиля важно правильно выбрать исходный стиль. Он должен быть как можно ближе к желаемому, чтобы минимизировать количество необходимых настроек.

Создание стиля. Для создания нового стиля надо щелкнуть на кнопке Создать стиль в Области задач. Откроется диалоговое окно Создание стиля. В данном окне следует:

- ввести название нового стиля в поле Имя;
- выбрать тип стиля (стиль абзаца, знака, списка или таблицы);
- выбрать стиль, на котором основан новый стиль;
- указать стиль следующего абзаца;
- настроить основные элементы стиля, используя средства данного диалогового окна;
- настроить дополнительные элементы стиля с помощью кнопки Формат.

Важной чертой программы является принцип наследования стилей. Он состоит в том, что любой стиль может быть основан на каком-то из существующих стилей. Это позволяет, во-первых, сократить настройку стиля до минимума, сосредоточившись только на отличиях от базового, а во-вторых, обеспечить принцип единства оформления всего документа в целом. Так, например, при изменении базового стиля автоматически произойдут и изменения наследуемых элементов в стилях, созданных на его основе.

Стиль следующего абзаца указывают для обеспечения автоматического применения стиля к следующему абзацу, после того как предыдущий абзац закрывается клавишей ENTER.

Разработка новых стилей и их настройка являются достаточно сложными технологическими операциями. Они требуют тщательного планирования, внимательности и аккуратности, особенно в связи с тем, что согласно принципу наследования свойств стилей желаемые изменения в одном стиле могут приводить к нежелательным изменениям во многих других стилях.

В связи с трудоемкостью изучения и освоения приемов практической работы со стилями начинающие пользователи часто ими пренебрегают. Действительно, при разработке небольших документов (одна-две страницы) можно обойтись без настройки и использования стилей, выполнив все необходимое форматирование вручную средствами меню Формат. Однако при разработке объемных документов вручную очень трудно обеспечить единство оформления, особенно если разные разделы документа разрабатывались разными авторами.

Прийти к использованию стилей надо как можно раньше. Правильное и рациональное использование этого средства является залогом высокой эффективности работы с процессором Microsoft Word и высокого качества разрабатываемых документов. На изучение средств управления стилями может потребоваться несколько часов, но полученные при этом навыки останутся и пригодятся многократно.

3.9 Шаблоны

Совокупность удачных стилевых настроек сохраняется вместе с готовым документом, но желательно иметь средство, позволяющее сохранить их и вне документа. Тогда их можно использовать для подготовки новых документов. Такое средство есть — это шаблоны, причем некоторое количество универсальных шаблонов поставляется вместе с текстовым процессором и устанавливается на компьютере вместе с ним.

По своей сути, шаблоны — это тоже документы, а точнее говоря, заготовки будущих документов. От обычных документов шаблоны отличаются тем, что в них приняты специальные меры, исключающие возможность их повреждения. Открывая шаблон, мы начинаем новый документ и вносим изменения в содержание шаблона. При сохранении же мы записываем новый документ, а шаблон, использованный в качестве его основы, остается в неизменном виде и пригоден для дальнейшего использования.

Использование шаблона для создания документа. По команде Файл ► Создать открывается Область задач в режиме Создание документа. Здесь можно выбрать шаблон, на базе которого документ будет разрабатываться. В этом случае документ сразу получает несколько готовых стилей оформления, которые содержатся в шаблоне. Основные шаблоны перечислены в области задач в разделе Создание. Если их недостаточно, надо щелкнуть на ссылке Общие шаблоны и выбрать подходящий шаблон на одной из вкладок открывшегося диалогового окна Шаблоны.

Изменение шаблона готового документа. Эта достаточно редкая операция выполняется с помощью диалогового окна Шаблоны и настройки (Сервис ► Шаблоны и настройки). Для смены

текущего шаблона следует использовать кнопку Присоединить и в открывшемся диалоговом окне Присоединение шаблона выбрать нужный шаблон в папке C:\Program Files\Microsoft Office\tlla6flOHbi.

Создание нового шаблона на базе существующего шаблона. Открыв диалоговое окно **Шаблон** щелчком на ссылке **Общие шаблоны** в **Области задач (режим Создание документа**), включите переключатель **Шаблон**. Теперь надо выбрать стандартный шаблон, на базе которого создается новый. После настройки стилей и редактирования содержания выполняется сохранение шаблона командой **Сохранить как** с включением пункта **Шаблон документа** в поле **Тип файла**.

Создание нового шаблона на базе документа. Если готовый документ может быть использован в качестве заготовки для создания других документов, его целесообразно сохранить как шаблон. Командой Файл ► Открыть открывают готовый документ, в нем правят содержание и настраивают стили, а потом сохраняют документ как шаблон командой Сохранить как с включением пункта Шаблон документа в поле Тип файла.

3.10 Темы

Последние версии текстового процессора Microsoft Word (начиная с Word 2000) имеют специальное средство автоматического оформления, предназначенное в первую очередь для электронных документов (для Web-документов и документов, распространяемых в формате процессора). Это средство называется темы. Тема представляет собой совокупность следующих элементов оформления:

- фоновый узор;
- стили оформления основного текста и заголовков;
- стиль оформления маркированных списков;
- стиль графических элементов оформления (линий).

Доступ к выбору тем выполняется командой Формат ► Темы.

3.11 Приемы управления объектами Microsoft Word

3.11.1 Особенности объектов Word

Текстовый процессор Word XP обладает развитой функциональностью по работе с объектами нетекстовой природы. Среди встроенных объектов могут быть стандартные объекты, созданные другими программами (рисунки, анимационные и звуковые клипы и многое другое), а также объекты, созданные средствами самого текстового процессора. В частности, программа позволяет создавать и встраивать геометрические фигуры, художественные заголовки, диаграммы, формульные выражения, заготовленные векторные иллюстрации (клипарты), то есть в ней имеются средства, отдаленно напоминающие средства специализированных графических редакторов. Правда, среди этих средств нет ничего для создания и обработки растровых иллюстраций — их можно только импортировать из других программ, но зато есть средства для управления их визуализацией, например для изменения яркости, контрастности и масштаба изображения.

Несмотря на столь разностороннюю природу объектов, с которыми может работать текстовый процессор Word XP, у них есть общие свойства, например такие, как размер, положение на странице, характер взаимодействия с текстом. Сначала мы остановимся на изучении самых общих свойств встроенных объектов, не обсуждая их природу, — это поможет освоить базовые приемы работы с объектами. А с конкретными свойствами конкретных объектов мы познакомимся чуть позже. Но перед тем как приступать к изучению приемов работы с объектами WordXP, необходимо сделать важное замечание о целесообразности их применения. На этот счет существуют весьма противоречивые мнения.

Все объекты Microsoft WordXP безусловно можно использовать, если документ готовится для печати, то есть предполагается, что он будет передаваться заказчику или распространяться в виде бумажной копии, выполненной на принтере. Оформление документов с помощью встроенных объектов позволяет сделать их представительными.

Если документ предполагается передать в виде файла для последующей обработки (а именно так передают рукописи в редакции), то все собственные средства программы по созданию и размещению встроенных объектов не только бесполезны, но и вредны. Это связано с тем, что объекты Microsoft Word XP не стандартны и не поддерживаются профессиональными программами. Компания Microsoft имеет лидирующее положение в отрасли и может не считаться с общепринятыми стандартами и правилами, а внедрять свои. Поэтому объекты, созданные в программах компании Microsoft, могут полноценно использоваться только в программах этой компании.

Из последнего замечания вытекает еще одно направление для использования объектов, созданных в Microsoft Word. Их можно успешно экспортировать через буфер обмена Windows в другие программные продукты, входящие в пакет Microsoft Office XP, например такие, как система управления электронными таблицами Excel, система управления базами данных Access и другие.

Взаимодействие объектов Word с текстом и страницей.

Управление размером и положением объекта, встроенный в текст документа графический объект, обладает рядом свойств. Самое очевидное свойство — его размер. Когда объект выделен, вокруг него видны восемь квадратных маркеров. При наведении указателя мыши на один из маркеров указатель меняет форму и превращается в двунаправленную стрелку. В этот момент размер объекта можно менять методом протягивания мыши. Угловые маркеры позволяют пропорционально изменять размер объекта как по горизонтали, так и по вертикали. Четыре маркера, расположенные на сторонах воображаемого прямоугольника, позволяют управлять размером по одному направлению (по вертикали или горизонтали).

При наведении указателя мыши на сам объект указатель меняет форму и превращается в четырехнаправленную стрелку. В таком состоянии объект можно перетаскивать с помощью мыши по рабочему полю документа. Он займет новое положение в тот момент, когда левая кнопка мыши будет отпущена после перетаскивания.

Расширенное управление свойствами объектов. Вручную

мы можем только управлять размером, поворотом и положением объекта на странице. Для управления всеми остальными свойствами объектов нужны дополнительные средства — их можно найти в двух местах:

- на панели инструментов, соответствующей типу объекта (она открывается автоматически, когда объект выделен);
- в диалоговом окне *Формат объекта*, которое открывают из контекстного меню объекта (после щелчка правой кнопкой мыши на объекте).

С помощью панели инструментов управляют индивидуальными свойствами объектов (у разных типов объектов они различны), а с помощью диалогового окна *Формат объекта* управляют наиболее общими свойствами, имеющимися у объектов любых типов.

Взаимодействие объекта с окружающим текстом. Вставив объект в текст, следует задать характер его взаимодействия с текстом. Средства для этого представлены на вкладке *Положение* диалогового окна *Формат объекта*. Возможны следующие варианты:

- Вариант *В тексте* используют для графических объектов малого размера, сопоставимого с размерами символов текста. В этом случае объект вставляется в текстовую строку на правах графического символа и далее перемещается постранице только вместе с текстом.
- Вариант *Вокруг рамки* Текст располагается вокруг воображаемой прямоугольной рамки, охватывающей весь контур объекта.
- Вариант По контуру отличается от предыдущего тем, что воображаемая прямоугольная рамка не проводится и текст плавно обтекает контур объекта (если он криволинейный).
- Вариант *Перед текстом* — это прием вставки объекта без обтекания. Текст и объект лежат на разных слоях, причем объект лежит выше и загораживает часть текста. Этим приемом пользуются, когда оформление важнее содержания.
- Вариант *За текстом* — это еще один прием вставки объекта без обтекания. Текст и объект тоже лежат на разных слоях, но в данном случае объект лежит на нижнем слое и загораживается текстом. Этот вариант используют для размещения текста на тематическом художественном фоне.

Дополнительные варианты взаимодействия текста со встроенным объектом можно найти в диалоговом окне Дополнительная разметка, которое открывают с помощью кнопки Дополнительно.

- Вариант *Сквозное* — это прием обтекания, аналогичный обтеканию *По контуру*, но в данном случае текст обтекает объект не только снаружи, но и изнутри.

Там же, в диалоговом окне *Дополнительная разметка* можно выбрать вариант обтекания Сверху и снизу. Этот прием используют наиболее часто — его считают основным для объектов, ширина которых составляет более половины ширины страницы.

Прочие параметры взаимодействия объекта с окружающим текстом. Более тонкую настройку взаимодействия объектов с текстом выполняют с помощью элементов управления, имеющихся в диалоговом окне *Дополнительная разметка*. В частности, здесь можно с помощью переключателей конкретно указать, с каких сторон объекта происходит обтекание, а с каких — нет. Здесь же можно указать величину интервала в миллиметрах между текстом и объектом.

Управление горизонтальным положением объекта относительно элементов печатной страницы. Завершив настройку взаимодействия объекта с текстом, приступают к размещению объекта на странице. Как уже говорилось выше, это можно сделать вручную методом перетаскивания объекта с помощью мыши, но более точную настройку выполняют с помощью рассмотренного диалогового окна *Формат объекта* ► *Положение.*

Варианты горизонтального размещения объекта:

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

Варианты По левому краю и По правому краю обычно используют при обтекании По контуру или Вокруг рамки. Вариант По центру часто сочетают с обтеканием Сверху и снизу, а последний вариант соответствует ручному размещению объекта перетаскиванием с помощью мыши.

Управление вертикальным положением объекта относительно элементов печатной страницы. К объекту, встроенному в

текст, можно подходить с двух позиций: как к элементу оформления страницы или как к элементу оформления содержания, то есть текста. Разница заключается в том, что происходит с объектом во время редактирования текста: он перемещается вместе с ним (с абзацами, к которым он примыкает) или он неподвижен, а текст перемещается, обтекая объект по заданным правилам.

В первом случае объект надо закрепить относительно абзаца, а во втором случае — относительно страницы. Необходимую настройку выполняют элементами управления вкладки *Положение рисунка* в диалоговом окне *Дополнительная разметка*. Вертикальное положение объекта относительно элементов страницы задают установкой переключателя *Выравнивание* и выбором метода выравнивания и элемента, относительно которого происходит выравнивание. Вертикальное положение относительно текста задают установкой переключателя *Положение* и выбором объекта, относительно которого положение задается, например абзаца.

Чтобы объект был связан с элементом страницы и не перемещался вместе с текстом, устанавливают флажок Установить привязку. Чтобы объект мог перемещаться вместе с текстом, устанавливают флажок Перемещать вместе с текстом.

Управление свойствами объектов Microsoft Word.

Управление размерами объекта. Размерами встроенных объектов можно управлять перетаскиванием графических маркеров с помощью мыши. Это прием ручного управления. Однако существуют и приемы автоматического управления. Их реализуют с помощью элементов управления вкладки Размер рассмотренного выше диалогового окна Формат объекта. Счетчиками Высота, Ширина и Поворот задают вертикальные и горизонтальные размеры объекта, а также его угол поворота по часовой стрелке.

Размерами объектов можно управлять не только в абсолютном исчислении, но и в относительном (в процентах относительно исходного). Для этого служат счетчики группы Масштаб. Чтобы размеры объекта синхронно изменялись по вертикали и горизонтали, надо установить флажок Сохранить пропорции.

Управление свойствами линии. Большинство объектов, создаваемых средствами самой программы Word XP, имеют векторную природу, то есть в их основе лежат простейшие геометрические фигуры — линии. Эти линии, в свою очередь, имеют собст-

венные свойства: толщину, цвет и тип. Управление этими свойствами выполняют с помощью средств вкладки Формат объекта ► Цвета и линии.

Взаимодействие объектов друг с другом.

Выше рассмотрено, как объекты взаимодействуют с текстом и с элементами печатной страницы, но если на одной странице имеется несколько встроенных объектов, то они могут взаимодействовать и друг с другом, и характером этого взаимодействия тоже нужно управлять.

Первое, что нужно решить, — это разрешено ли объектам перекрывать друг друга. Для тех объектов, которым перекрытие разрешено, следует установить флажок Формат объекта ► Положение ► Дополнительно ► Положение объекта ► Разрешить перекрытие. Напомним, что доступ к диалоговому окну Формат объекта открывается командой (для разных объектов она может называться по-разному) контекстного меню объекта.

Управление взаимным положением объектов выполняют с помощью операций:

- группирования;
- задания порядка следования;
- выравнивания;
- распределения.

Группирование объектов. Если на странице представлено несколько объектов и при этом важно строго зафиксировать их взаимное расположение, то их объединяют в один комплексный (групповой) объект с помощью операции группирования. После этой операции свойства группового объекта можно настраивать точно так же, как мы настраивали свойства простейших объектов, — ему может быть задан характер обтекания текстом, метод привязки к абзацу или к элементам печатной страницы и т.п.

Для группирования нескольких объектов их следует выделить (выделение нескольких объектов выполняют при нажатой клавише SHIFT), щелкнуть на любом из объектов группы правой кнопкой мыши и выбрать в контекстном меню команду Группировка ► Группировать. Сгруппированные объекты можно перемещать как единое целое. Чтобы разгруппировать объекты и получить доступ к индивидуальным свойствам каждого из них, надо выделить группу и дать команду Группировка ► Разгруппировать.

Управление порядком следования объектов. Если на странице документа размещается несколько объектов, то предполагается, что у каждого объекта есть свой слой. По умолчанию порядок следования слоев связан с порядком создания объектов, то есть те объекты, которые были созданы раньше, лежат на слоях ниже, чем объекты, созданные позже. Если между объектами нет перекрытия, то мы не замечаем, что существует некий порядок следования объектов, однако, когда объекты перекрывают друг друга, этот порядок становится заметен.

Управляют порядком следования объектов с помощью команды Порядок контекстного меню. Она открывает вложенное меню, средствами которого можно поднять объект на передний план, опустить на задний план, сместить на один слой вверх или вниз и задать положение объекта относительно текста.

Выравнивание объектов. Если объекты, составляющие композицию, не перекрывают друг друга, важно иметь средство их относительного выравнивания между собой. Выравнивание объектов выполняют до группирования, ведь после него объекты уже нельзя сдвинуть друг относительно друга. В этом случае операция группирования закрепляет взаимное расположение объектов. После нее объекты уже не могут сдвинуться друг относительно друга, и положением всей группы на странице можно управлять как единым целым. Чтобы выполнить выравнивание, необходимо предварительно открыть дополнительную панель инструментов Рисование (Вид ► Панели инструментов ► Рисование).

Для выравнивания нескольких объектов между собой их следует выделить при нажатой клавише SHIFT, а затем дать команду *Действия* ► *Выровнять/распределить* (с помощью кнопки Действия панели инструментов Рисование). Существует шесть методов выравнивания. Им соответствуют три команды горизонтального выравнивания (По левому краю, По правому краю, По центру) и три команды выравнивания вертикального (По верхнему краю, По нижнему краю, По середине). Следует обратить внимание на особенность действия команд выравнивания. Так, например, если два объекта выравниваются по нижнему полю, значит, они выравниваются по нижнему полю нижнего объекта. Выравнивание по правому полю — это выравнивание по правому полю самого правого объекта из числа выделенных и так далее.

Если необходимо выполнить выравнивание относительно полей страницы, следует предварительно установить флажок меню *Действия* ► *Выровнять/распределить* ► *Относительно страницы.*

Распределение объектов. Эта операция родственна выравниванию. Ее суть в том, что между объектами устанавливаются равные интервалы по горизонтали или (и) вертикали. Соответственно, в меню команды *Действия* ► *Выровнять/распределить* имеются команды: Распределить по горизонтали и Распределить по вертикали.

Равномерное распределение объектов обычно выполняют после выравнивания, но, разумеется, до группирования. Нередко объекты выравнивают по вертикали и одновременно равномерно распределяют по горизонтали или, соответственно, наоборот. Дополнительное отличие команд распределения от команд выравнивания заключается еще и в том, что для взаимного выравнивания достаточно иметь два выделенных объекта, а для команд распределения должно быть выделено не менее трех объектов.

Ввод формул.

Необходимость в наличии средства для ввода математических выражений в текстовый документ характерна для научно-технической документации.

В программе Microsoft Word средством ввода формул является редактор формул Microsoft Equation 3.0. Он позволяет создавать формульные объекты и вставлять их в текстовый документ. При необходимости вставленный объект можно редактировать непосредственно в поле документа.

3.11.2 Запуск и настройка редактора формул

Для запуска редактора формул служит команда Вставка ► Объект. В открывшемся диалоговом окне Вставка объекта следует выбрать пункт Microsoft Equation 3.0 в списке Тип объекта на вкладке Создание. Откроется панель управления Формула, представленная на рис. 11.8. При этом строка меню текстового процессора замещается строкой меню редактора формул.

Прежде чем пользоваться редактором формул, следует выполнить его настройку. Настройка состоит в назначении шрифтов

для различных элементов, входящих в формулы., Она выполняется в диалоговом окне Стили, открываемом командой Стиль ► Определить. Эта настройка является обязательной — без нее редактор формул работать не будет, но выполнить ее достаточно только один раз.

Прочие (необязательные) настройки редактора формул выполняют в диалоговом окне Интервал (Формат ► Интервал). Многочисленные средства настройки, присутствующие в нем, предназначены для задания размеров различных элементов формул.

Панель инструментов редактора формул содержит два ряда кнопок. Кнопки нижнего ряда создают своеобразные шаблоны, содержащие поля для ввода символов. Так, например, для ввода обыкновенной дроби следует выбрать соответствующий шаблон, имеющий два поля: числитель и знаменатель. Заполнение этих полей может производиться как с клавиатуры, так и с помощью элементов управления верхней строки. Переходы между полями выполняются с помощью клавиш управления курсором.

Ввод и редактирование формул завершается нажатием клавиши ESC или закрытием панели редактора формул. Можно также щелкнуть левой кнопкой мыши где-либо в поле документа вне области ввода формулы. Введенная формула автоматически вставляется в текст в качестве объекта. Далее ее можно переместить в любое иное место документа через буфер обмена (CTRL+X — вырезать; CTRL+V — вставить). Для редактирования формулы непосредственно в документе достаточно выполнить на ней двойной щелчок. При этом автоматически открывается окно редактора формул.

3.11.3 Особенности редактора формул

Редактор формул Microsoft Equation 3.0 представляет собой отдельный компонент, поэтому при установке текстового процессора требуется специально указать необходимость его подключения.

При работе с редактором формул следует стремиться к максимальной полноте вводимых выражений. Так, например, выражение (формула) может содержать компоненты, ввод которых

возможен и без использования редактора формул, но для удобства работы и простоты дальнейшего редактирования следует вводить всю формулу целиком только в редакторе формул, не используя иные средства.

При вводе формул и выражений не рекомендуется использовать символы русского алфавита. В тех случаях, когда они необходимы, например, в качестве описательных индексов переменных, им следует назначать стиль Текст.

В редакторе формул не работает клавиша ПРОБЕЛ, поскольку необходимые интервалы между символами создаются автоматически. Однако если необходимость ввода пробелов все-таки возникнет, то их можно вводить с помощью кнопки Пробелы и многоточия панели инструментов Формула. Всего предусмотрено пять разновидностей пробелов различной ширины.

3.12 Работа с таблицами

Данные, представленные в табличной форме, отличаются наглядностью. Таблицы всегда были неотъемлемым атрибутом печатной научно-технической документации, а в последние годы стали и эффективным средством оформления Web-страниц Интернета. Это связано с тем, что в силу естественных причин возможности форматирования Web-страниц весьма ограниченны. Поэтому многие Web-дизайнеры используют таблицы (в том числе и скрытые), чтобы принудительно управлять отображением данных на экране клиента и не доверять этот ответственный процесс средству просмотра Web (браузеру). Так, например, таблицы — это простейшее средство для имитации на Wei-странице газетного или журнального текста, имеющего две и более колонок.

Ячейки таблиц могут содержать не только текст, но и графические и прочие объекты. Благодаря этому можно размещать несколько иллюстраций по ширине Web-страницы (обычные средства форматирования Web-страниц не позволяют это сделать).

При создании страниц — можно управлять методом представления ячеек и рамок, как внешних, так и внутренних. При создании печатных документов таблицы оформляют так, чтобы они соответствовали стилю и содержанию документа. При созда-

нии Web-страниц существует прием, когда рамки вообще не отображают, а между ячейками делают зазор. В результате этого объекты, находящиеся в ячейках, образуют ровные регулярные структуры на экране, в то время как никаких следов таблиц на экране не видно.

Текстовый процессор Microsoft Word обладает мощными средствами создания таблиц как для печатных, так и для электронных документов.

Три основных средства создания таблиц — это:

1) кнопка *Добавить таблицу* на панели инструментов *Стандартная;*

2) диалоговое окно Вставка таблицы (Таблица ► Вставить ► Таблица);

3) средство рисования таблиц Таблицы и границы (Таблица ► Нарисовать таблицу).

4 ЭЛЕКТРОННАЯ ТАБЛИЦА EXCEL

Для представления данных в удобном виде используют таблицы. Компьютер позволяет представлять их в электронной форме, а это дает возможность не только отображать, но и обрабатывать данные. Класс программ, используемых для этой цели, называется *электронными таблицами.*

Особенность электронных таблиц заключается в возможности применения формул для описания связи между значениями различных ячеек. Расчет по заданным формулам выполняется автоматически. Изменение содержимого какой-либо ячейки приводит к пересчету значений всех ячеек, которые с ней связаны формульными отношениями и, тем самым, к обновлению всей таблицы в соответствии с изменившимися данными.

Применение электронных таблиц упрощает работу с данными и позволяет получать результаты без проведения расчетов вручную или специального программирования. Наиболее широкое применение электронные таблицы нашли в экономических и бухгалтерских расчетах, но и в научно-технических задачах электронные таблицы можно использовать эффективно, например:

- для проведения однотипных расчетов над большими наборами данных;
- решения задач путем подбора значений параметров, табулирования формул;
- обработки результатов экспериментов;
- проведения поиска оптимальных значений параметров;
- подготовки табличных документов;
- построения диаграмм и графиков по имеющимся данным.

Одним из наиболее распространенных средств работы с документами, имеющими табличную структуру, является программа *Microsoft Excel.*

4.1 Основные понятия электронных таблиц

Программа *Microsoft Excel* предназначена для работы с таблицами данных, преимущественно числовых. При формировании таблицы выполняют ввод, редактирование и форматирование

текстовых и числовых данных, а также формул. Наличие средств автоматизации облегчает эти операции. Созданная таблица может быть выведена на печать.

Рабочая книга и рабочий лист. Строки, столбцы, ячейки. Документ *Excel* называется рабочем *книгой.* Рабочая книга представляет собой набор *рабочих листов,* каждый из которых имеет табличную структуру и может содержать одну или несколько таблиц. В окне документа в программе *Excel* отображается только *текущий* рабочий лист, с которым и ведется работа. Каждый рабочий лист имеет *название,* которое отображается на ярлычке *листа,* отображаемом в его нижней части. С помощью ярлычков можно переключаться к другим рабочим листам, входящим в ту же самую рабочую книгу. Чтобы переименовать рабочий лист, надо дважды щелкнуть на его ярлычке.

Рабочий лист состоит из *строк* и *столбцов.* Столбцы озаглавлены прописными латинскими буквами и, далее, двухбуквенными комбинациями. Всего рабочий лист может содержать до 256 столбцов, пронумерованных от A до IV. Строки последовательно нумеруются цифрами, от 1 до 65 536 (максимально допустимый номер строки).

Ячейки и их адресация. На пересечении столбцов и строк образуются *ячейки* таблицы. Они являются минимальными элементами для хранения данных. Обозначение отдельной ячейки сочетает в себе номера столбца и строки (в этом порядке), на пересечении которых она расположена, например: A1 или DE234. Обозначение ячейки (ее номер) выполняет функции ее адреса. Адреса ячеек используются при записи формул, определяющих взаимосвязь между значениями, расположенными в разных ячейках.

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

Диапазон ячеек. На данные, расположенные в соседних ячейках, можно ссылаться в формулах как на единое целое. Такую группу ячеек называют *диапазоном.* Наиболее часто используют прямоугольные диапазоны, образующиеся на пересечении группы последовательно идущих строк и группы последователь-

но идущих столбцов. Диапазон ячеек обозначают, указывая через двоеточие номера ячеек, расположенных в противоположных углах прямоугольника, например: A1:C15.

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.

4.2 Ввод, редактирование и форматирование данных

Отдельная ячейка может содержать данные, относящиеся к одному из трех типов: *текст, число* или *формула,* — а также оставаться пустой. Программа *Excel* при сохранении рабочей книги записывает в файл только прямоугольную область рабочих листов, примыкающую к левому верхнему углу (ячейка A1) и содержащую все заполненные ячейки.

Тип данных, размещаемых в ячейке, определяется автоматически при вводе. Если эти данные можно интерпретировать как число, программа *Excel* так и делает. В противном случае данные рассматриваются как текст. Ввод формулы всегда начинается с символа «=» (знака равенства).

Ввод текста и чисел. Ввод данных осуществляют непосредственно в текущую ячейку или в *строку формул,* располагающуюся в верхней части окна программы под панелями инструментов. Место ввода отмечается текстовым курсором. Если начать ввод нажатием алфавитно-цифровых клавиш, данные из текущей ячейки 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 формул.

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.

4.3 Содержание электронной таблицы

Формулы. Вычисления в таблицах программы *Excel* осуществляются при помощи *формул.* Формула может содержать числовые константы, ссылки на ячейки и *функции Excel,* соединенные знаками математических операций. Скобки позволяют изменять стандартный порядок выполнения действий. Если ячейка содержит формулу, то в рабочем листе отображается текущий результат вычисления этой формулы. Если сделать ячейку текущей, то сама формула отображается в строке формул.

Правило использования формул в программе *Excel* состоит в том, что, если значение ячейки *действительно* зависит от других ячеек таблицы, *всегда* следует использовать формулу, даже если операцию легко можно выполнить в «уме». Это гарантирует, что последующее редактирование таблицы не нарушит ее целостности и правильности производимых в ней вычислений.

4.4 Ссылки на ячейки

Формула может содержать *ссылки,* то есть адреса ячеек, содержимое которых используется в вычислениях. Это означает, что результат вычисления формулы зависит от числа, находящегося в другой ячейке. Ячейка, содержащая формулу, таким обра-

зом, является *зависимой.* Значение, отображаемое в ячейке с формулой, пересчитывается при изменении значения ячейки, на которую указывает ссылка.

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

Все диалоговые окна программы *Excel,* которые требуют указания номеров или диапазонов ячеек, содержат кнопки, присоединенные к соответствующим полям. При щелчке на такой кнопке диалоговое окно сворачивается до минимально возможного размера, что облегчает выбор нужной ячейки (диапазона) с помощью щелчка или протягивания.

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.

4.5 Абсолютные и относительные ссылки

По умолчанию, ссылки на ячейки в формулах рассматриваются как *относительные* Это означает, что при копировании формулы адреса в ссылках автоматически изменяются в соответствии с относительным расположением исходной ячейки и создаваемой копии.

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. Если формула будет скопирована в другую ячейку, то такое относительное указание ссылки сохранится. Например, при копировании формулы в ячейку EA27 ссылка будет продолжать указывать на ячейку, располагающуюся левее и ниже, в данном случае на ячейку DZ28.

При *абсолютной адресации* адреса ссылок при копировании не изменяются, так что ячейка, на которую указывает ссылка,

рассматривается как нетабличная. Для изменения способа адресации при редактировании формулы надо выделить ссылку на ячейку и нажать клавишу F4. Элементы номера ячейки, использующие абсолютную адресацию, предваряются символом $. Например, при последовательных нажатиях клавиши F4 номер ячейки A1 будет записываться как A1, A1, A$1 и $A1. В двух последних случаях один из компонентов номера ячейки рассматривается как абсолютный, а другой — как относительный.

4.5.1 Копирование содержимого ячеек

Копирование и перемещение ячеек в программе *Excel* можно осуществлять методом перетаскивания или через буфер обмена. При работе с небольшим числом ячеек удобно использовать первый метод, при работе с большими диапазонами — второй.

Метод перетаскивания. Чтобы методом перетаскивания скопировать или переместить текущую ячейку (выделенный диапазон) вместе с содержимым, следует навести указатель мыши на рамку текущей ячейки (он примет вид стрелки с дополнительными стрелочками). Теперь ячейку можно перетащить в любое место рабочего листа (точка вставки помечается всплывающей подсказкой).

Для выбора способа выполнения этой операции, а также для более надежного контроля над ней рекомендуется использовать *специальное перетаскивание* с помощью правой кнопки мыши. В этом случае при отпускании кнопки мыши появляется специальное меню, в котором можно выбрать конкретную выполняемую операцию.

Применение буфера обмена. Передача информации через буфер обмена имеет в программе *Excel* определенные особенности, связанные со сложностью контроля над этой операцией. Вначале необходимо выделить копируемый (вырезаемый) диапазон и дать команду на его помещение в буфер обмена: Правка ► Копировать или Правка ► Вырезать. Вставка данных в рабочий лист возможна лишь немедленно после их помещения в буфер обмена. Попытка выполнить любую другую операцию приводит к отмене начатого процесса копирования или перемещения. Однако утраты данных не происходит, поскольку «вырезанные»

данные удаляются из места их исходного размещения только в момент выполнения вставки.

Место вставки определяется путем указания ячейки, соответствующей верхнему левому углу диапазона, помещенного в буфер обмена, или путем выделения диапазона, который по размерам в точности равен копируемому (перемещаемому). Вставка выполняется командой Правка > Вставить. Для управления способом вставки можно использовать команду Правка ► Специальная вставка. В этом случае правила вставки данных из буфера обмена задаются в открывшемся диалоговом окне.

4.6 Автоматизация ввода

Так как таблицы часто содержат повторяющиеся или однотипные данные, программа *Excel* содержит средства автоматизации ввода. К числу предоставляемых средств относятся: *автозавершение, автозаполнение числами* и *автозаполнение формулами.*

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 *Excel* проверяет соответствие введенных символов строкам, имеющемся 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 числами. При работе с числами используется метод *автозаполнения.* В правом нижнем углу рамки текущей ячейки имеется черный квадратик — *маркер заполнения.* При наведении на него указатель мыши (он обычно имеет вид толстого белого креста) приобретает форму тонкого черного крестика. Перетаскивание маркера заполнения рассматривается как операция «размножения» содержимого ячейки в горизонтальном

или вертикальном направлении.

Если ячейка содержит число (в том числе дату, денежную сумму), то при перетаскивании маркера происходит копирование ячеек или их заполнение арифметической прогрессией. Для выбора способа автозаполнения следует производить специальное перетаскивание с использованием правой кнопки мыши.

Пусть, например, ячейка **A1** содержит число 1. Наведите указатель мыши на маркер заполнения, нажмите правую кнопку мыши и перетащите маркер заполнения так, чтобы рамка охватила ячейки **A1, B1** и **C1**, и отпустите кнопку мыши. Если теперь выбрать в открывшемся меню пункт *Копировать ячейки*, все ячейки будут содержать число 1. Если же выбрать пункт Заполнить, то в ячейках окажутся числа 1, 2 и 3.

Чтобы точно сформулировать условия заполнения ячеек, следует дать команду *Правка* ► *Заполнить* ► *Прогрессия*. В открывшемся диалоговом окне *Прогрессия* выбирается тип прогрессии, величина шага и предельное значение. После щелчка на кнопке OK программа *Excel* автоматически заполняет ячейки в соответствии с заданными правилами.

Автозаполнение формулами. Эта операция выполняется так же, как автозаполнение числами. Ее особенность заключается в необходимости копирования ссылок на другие ячейки. В ходе автозаполнения во внимание принимается 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 ячейках столбцов **A** и **B**. Введем в ячейку **C1** формулу **=A1+B1**. Теперь скопируем эту формулу методом автозаполнения во все ячейки третьего столбца таблицы. Благодаря относительной адресации формула будет правильной для всех ячеек данного столбца.

В таблице 4.1 приведены правила обновления ссылок при автозаполнении вдоль строки или вдоль столбца.

Таблица 4.1 — **Правила обновления ссылок при автозаполнении**

Ссылка в исходной ячейке	**Ссылка в следующей ячейке**	
	заполнение вправо	заполнение вниз
A1 (относительная)	B1	A2
$A1 (абсолютная по столбцу)	$A1	$A2
A$1 (абсолютная по строке)	B$1	A$1
A1 (абсолютная)	A1	A1

Использование стандартных функций

Стандартные функции используются в программе *Excel* только в формулах. *Вызов функции* состоит в указании в формуле *имени функции,* после которого в скобках указывается *список параметров.* Отдельные параметры разделяются в списке точкой с запятой. В качестве параметра может использоваться число, адрес ячейки или произвольное выражение, для вычисления которого также могут использоваться функции.

В режиме ввода формулы в левой части строки формул, где раньше располагался номер текущей ячейки, появляется раскрывающийся список функций. Он содержит десять функций, которые использовались последними, а также пункт Другие функции.

Использование мастера функций. При выборе пункта *Другие функции* запускается *Мастер функций*, облегчающий выбор нужной функции. В раскрывающемся списке *Категория* выбирается категория, к которой относится функция (если определить категорию затруднительно, используют пункт *Полный алфавитный перечень*), а в списке *Выберите функцию* — конкретная функция данной категории. После щелчка на кнопке ОК имя функции заносится в строку формул вместе со скобками, ограничивающими список параметров. Текстовый курсор устанавливается между этими скобками. Вызвать *Мастер функций* можно щелчком на кнопке *Вставка* функции в строке формул.

Аргументы функции. Как только имя функции выбрано, на экране появляется диалоговое окно. Аргументы функции (в предыдущих версиях Excel это окно рассматривалось как палитра формул). Это окно, в частности, содержит значение, которое получится, если немедленно закончить ввод формулы.

Правила вычисления формул, содержащих функции, не от-

личаются от правил вычисления более простых формул. Ссылки на ячейки, используемые в качестве параметров функции, также могут быть относительными или абсолютными, что учитывается при копировании формул методом автозаполнения.

4.7 Построение диаграмм и графиков

В программе *Excel* термин «диаграмма» используется для обозначения всех виде; графического представления числовых данных. Построение графического изображения производится на *основе ряда данных.* Так называют группу ячеек с данным; в пределах отдельной строки или столбца. На одной диаграмме можно отображать несколько рядов данных.

Диаграмма представляет собой вставной объект, внедренный на один из листов рабочей книги. Она может располагаться на том же листе, на котором находятся данные, или на любом другом листе (часто для отображения диаграммы отводят отдельный лист). Диаграмма сохраняет связь с данными, на основе которых она построена, и при обновлении этих данных немедленно изменяет свой вид.

Для построения диаграммы обычно используют Мастер диаграмм, запускаемый щелчком на кнопке Мастер диаграмм на стандартной панели инструментов. Чаете удобно заранее выделить область, содержащую данные, которые будут отображаться на диаграмме, но задать эту информацию можно и в ходе работы мастера.

Выбор типа диаграммы. На первом этапе работы мастера выбирают форму диаграммы. Доступные формы перечислены в списке Тип на вкладке Стандартные. Для выбранного типа диаграммы справа указывается несколько вариантов представления данных (палитра Вид), из которых следует выбрать наиболее подходящий. На вкладке Нестандартные отображается набор полностью сформированных типов диаграмм с готовым форматированием. После задания формы диаграммы следует щелкнуть на кнопке Далее.

Выбор данных. Второй этап работы мастера служит для выбора данных, по которым будет строиться диаграмма. Если диапазон данных был выбран заранее, то в области предвари-

тельного просмотра в верхней части окна мастера появится приблизительное отображение будущей диаграммы. Если данные образуют единый прямоугольный диапазон, то их удобно выбирать при помощи вкладки Диапазон данных. Если данные не образуют единой группы, то информацию для рисования отдельных рядов данных задают на вкладке Ряд. Предварительное представление диаграммы автоматически обновляется при изменении набора отображаемых данных.

Оформление диаграммы. Третий этап работы мастера (после щелчка на кнопке *Далее*) состоит в выборе оформления диаграммы. На вкладках окна мастера задаются:

- название диаграммы, подписи осей (вкладка *Заголовки*);
- отображение и маркировка осей координат (вкладка *Оси*);
- отображение сетки линий, параллельных осям координат (вкладка *Линии сетки*);
- описание построенных графиков (*вкладка Легенда*);
- отображение надписей, соответствующих отдельным элементам данных на графике (вкладка *Подписи данных*);
- представление данных, использованных при построении графика, в виде таблицы (вкладка Таблица данных).

В зависимости от типа диаграммы некоторые из перечисленных вкладок могут отсутствовать.

Размещение диаграммы. На последнем этапе работы мастера (после щелчка на кнопке *Далее*) указывается, следует ли использовать для размещения диаграммы новый рабочий лист или один из имеющихся. Обычно этот выбор важен только для последующей печати документа, содержащего диаграмму. После щелчка на кнопке *Готово* диаграмма строится автоматически и вставляется на указанный рабочий лист.

4.8 Редактирование диаграммы

Готовую диаграмму можно изменить. Она состоит из набора отдельных элементов, таких, как сами графики (ряды данных), оси координат, заголовок диаграммы, область построения и прочее. При щелчке на элементе диаграммы он выделяется маркерами, а при наведении на него указателя мыши — описывается всплывающей подсказкой. Открыть диалоговое окно для форма-

тирования элемента диаграммы можно через меню Формат (для выделенного элемента) или через контекстное меню (команда Формат). Различные вкладки открывшегося диалогового окна позволяют изменять параметры отображения выбранного элемента данных.

5 БАЗЫ ДАННЫХ И СИСТЕМЫ УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ

База данных — это организованная структура, предназначенная для хранения информации. Данные и информация — понятия взаимосвязанные, но не тождественные, поэтому можно заметить некоторое несоответствие в этом определении. Его причины чисто исторические. В те годы, когда формировалось понятие баз данных, в них действительно хранились только данные. Однако сегодня большинство систем управления базами данных (СУБД) позволяют размещать в своих структурах не только данные, но и методы (то есть программный код), с помощью которых происходит взаимодействие с потребителем или с другими программно-аппаратными комплексами. Таким образом, мы можем говорить, что в современных базах данных хранятся отнюдь не только данные, но и информация.

С понятием базы данных тесно связано понятие системы управления базой данных. Это комплекс программных средств, предназначенных для создания структуры новой базы, наполнения ее содержимым, редактирования содержимого и визуализацией информации. Под визуализацией информации базы понимается отбор отображаемых данных в соответствии с заданным критерием, их упорядочение, оформление и последующая выдача на устройство вывода или передача по каналам связи.

Существует множество систем управления базами данных. Несмотря на то, что они могут по-разному работать с разными объектами и предоставляют пользователю различные функции и средства, большинство СУБД опираются на единый устоявшийся комплекс основных понятий. Это дает возможность рассмотреть одну систему и обобщить ее понятия, приемы и методы на весь класс СУБД.

В качестве учебного объекта мы выберем СУБД Microsoft Access, входящую в пакет Microsoft Office наряду с рассмотренными ранее пакетами Microsoft Word'и Microsoft Excel. В тех случаях, когда конкретные приемы операций зависят от используемой версии программы, мы будем опираться на версию Microsoft

Access 2002, хотя в основном речь будет идти о таких обобщенных понятиях и методах, для которых различия между конкретными версиями программ второстепенны.

5.1 Структура базы данных

Отметим, что если в базу еще не занесено никаких данных (пустая незаполненная база), то это все равно полноценная база данных. В «пустой» базе *нет собственно данных,* но *информация* в ней есть — это *структура базы*. Структура базы определяет методы занесения данных и их хранения в базе.

Основными объектами любой базы данных являются ее *таблицы*. Любая база данных имеет хотя бы одну таблицу. Соответственно, структура простейшей базы данных тождественно равна структуре ее таблицы. Известно, что структуру двумерной таблицы образуют *столбцы* и *строки*. В таблицах базы данных это, соответственно, *поля* и *записи*. Если записей в таблице пока нет, значит, ее структура образована только набором полей. Изменив состав полей базовой таблицы или их свойства, мы изменяем структуру базы данных и, соответственно, получаем новую базу данных.

5.2 Свойства полей базы данных

Поля базы данных не просто определяют структуру базы — они еще определяют групповые свойства данных, записываемых в ячейки, принадлежащие каждому из полей. Ниже перечислены основные свойства полей таблиц баз данных на примере СУБД Microsoft Access.

Имя поля — определяет, как следует ссылаться на эту часть данных при операциях с базой (по умолчанию имена полей используются в качестве заголовков столбцов таблиц).

Тип поля — определяет тип данных, которые могут содержаться в данном поле.

Размер поля — определяет предельную длину (в символах) данных, которые могут размещаться в данном поле.

Формат поля — определяет способ форматирования данных в ячейках, принадлежащих полю.

Маска ввода — определяет форму, в которой вводятся данные в поле (средство автоматизации ввода данных).

Подпись — определяет заголовок столбца таблицы для данного поля (если подпись не указана, то в качестве заголовка столбца используется свойство Имя поля).

Значение по умолчанию — то значение, которое вводится в ячейки поля автоматически (средство автоматизации ввода данных).

Условие на значение — ограничение, используемое для проверки правильности ввода данных (средство автоматизации ввода, которое используется, как правило, для данных, имеющих числовой тип, денежный тип или тип даты).

Сообщение об ошибке — текстовое сообщение, которое выдается автоматически при попытке ввода в поле ошибочных данных (проверка ошибочности выполняется автоматически, если задано свойство Условие на значение).

Обязательное поле — свойство, определяющее обязательность заполнения данного поля при наполнении базы;

Пустые строки — свойство, разрешающее ввод пустых строковых данных (от свойства Обязательное поле отличается тем, что относится не ко всем типам данных, а лишь к некоторым, например к текстовым).

Индексированное поле — если поле обладает этим свойством, все операции, связанные с поиском или сортировкой записей по значению, хранящемуся в данном поле, существенно ускоряются. Кроме того, для индексированных полей можно сделать так, что значения в записях будут проверяться по этому полю на наличие повторов, что позволяет автоматически исключить дублирование данных.

В разных полях могут содержаться данные разного типа, и свойства у полей могут различаться в зависимости от типа данных. Так, например, список вышеуказанных свойств полей относится в основном к полям текстового типа. Поля других типов могут иметь или не иметь эти свойства, но могут добавлять к ним и свои. Например, для данных, представляющих действительные числа, важным свойством является количество знаков после десятичной запятой, а для полей, используемых для хранения рисунков, звукозаписей, видеоклипов и других объектов OLE, ряд вышеуказанных свойств не имеет смысла.

5.3 Типы данных Microsoft Access

Базы данных Microsoft Access работают со следующими типами данных.

- **Текстовый** — тип данных, используемый для хранения обычного неформатированного текста ограниченного размера (до 255 символов).
- **Поле Мемо** — специальный тип данных для хранения больших объемов текста (до 65 535 символов). Физически текст не хранится в поле. Он хранится в другом месте базы данных, а в поле хранится указатель на него, но для пользователя такое разделение заметно не всегда.
- **Числовой** — тип данных для хранения действительных чисел.
- **Дата/время** — тип данных для хранения календарных дат и текущего времени.
- **Денежный** — тип данных для хранения денежных сумм. Теоретически, для их записи можно было бы пользоваться и полями числового типа, но для денежных сумм есть некоторые особенности (например, связанные с правилами округления), которые делают более удобным использование специального типа данных, а не настройку числового типа.
- **Счетчик** — специальный тип данных для уникальных (не повторяющихся в поле) натуральных чисел с автоматическим наращиванием. Естественное использование — для порядковой нумерации записей
- **Логический** — тип для хранения логических данных (могут принимать только два значения, например *Да* или *Нет*).
- **Поле объекта OLE** — специальный тип данных, предназначенный для хранения объектов OLE, например, мультимедийных. Реально, конечно, такие объекты в таблице не хранятся. Как и в случае полей MEMO, они хранятся в другом месте внутренней структуры файла базы данных, а в таблице хранятся только указатели на них (иначе работа с таблицами была бы чрезвычайно замедленной).
- **Гиперссылка** — специальное поле для хранения адресов URL для Web-объектов Интернета. При щелчке на ссылке автоматически происходит запуск браузера и воспроизведение объек-

та в его окне.

- **Мастер подстановок** — это не специальный тип данных. Это объект, настройкой которого можно автоматизировать ввод данных в поле так, чтобы не вводить их вручную, а выбирать из раскрывающегося списка.

5.3.1 Безопасность баз данных

Базы данных — это тоже файлы, но работа с ними отличается от работы с файлами других типов, создаваемых прочими приложениями. Обычно всю работу по обслуживанию файловой структуры берет на себя операционная система,.но для баз данных предъявляются особые требования с точки зрения безопасности, поэтому в них реализован другой подход к сохранению данных.

При работе с обычными приложениями для сохранения данных мы выдаем соответствующую команду, задаем имя файла и доверяемся операционной системе. Если мы закроем файл, не сохранив его, то вся работа по созданию или редактированию файла пропадет безвозвратно.

Базы данных — это особые структуры. Информация, которая в них содержится, очень часто имеет общественную ценность. Нередко с одной и той же базой (например, с базой регистрации автомобилей в ГИБДД) работают тысячи людей по всей стране. От информации, которая содержится в некоторых базах, может зависеть благополучие множества людей. Поэтому целостность содержимого базы не может и не должна зависеть ни от конкретных действий некоего пользователя, забывшего сохранить файл перед выключением компьютера, ни от перебоев в электросети.

Проблема безопасности баз данных решается тем, что в СУБД для сохранения информации используется двойной подход. В части операций, как обычно, участвует операционная система компьютера, но некоторые операции сохранения происходят в обход операционной системы.

Операции изменения структуры базы данных, создания новых таблиц или иных объектов происходят при сохранении файла базы данных. Об этих операциях СУБД предупреждает пользова-

теля. Это, так сказать, глобальные операции. Их никогда не проводят с базой данных, находящейся в коммерческой эксплуатации, — только с ее копией. В этом случае любые сбои в работе вычислительных систем не страшны.

С другой стороны, операции по изменению содержания данных, не затрагивающие структуру базы, максимально автоматизированы и выполняются *без предупреждения*. Если, работая с таблицей данных, мы что-то в ней меняем в составе данных, то изменения *сохраняются немедленно и автоматически*.

Обычно, решив отказаться от изменений в документе, его просто закрывают без сохранения и вновь открывают предыдущую копию. Этот прием работает почти во всех приложениях, но только не в СУБД. Все изменения, вносимые в таблицы базы, сохраняются на диске без нашего ведома, поэтому попытка закрыть базу «без сохранения» ничего не даст, так как все уже сохранено. Таким образом, редактируя таблицы баз данных, создавая новые записи и удаляя старые, мы как бы работаем с жестким диском напрямую, минуя операционную систему.

По указанным выше причинам нельзя заниматься учебными экспериментами на базах данных, находящихся в эксплуатации. Для этого следует создавать специальные учебные базы или выполнять копии структуры реальных баз (без фактического наполнения данными).

5.4 Формирование баз данных

5.4.1 Режимы работы с базами данных

Обычно с базами данных работают две категории исполнителей.

Первая категория — разработчики. Их задача состоит в разработке структуры таблиц базы данных. Кроме таблиц разработчики создают и другие объекты базы данных, предназначенные, для автоматизации работы с базой и для ограничения функциональных возможностей работы с базой, если это необходимо из соображений безопасности. Разработчик, как правило, не наполняет базу конкретными данными, которые зачастую могут быть конфиденциальными.

Вторая категория работающих с базами данных, — пользователи. Они получают базу данных от разработчика и занимаются ее наполнением и обслуживанием. В общем случае пользователи не имеют средств доступа к управлению структурой базы — только к данным, да и то не ко всем, а лишь к тем, с которыми они уполномочены работать.

Соответственно, СУБД имеет два режима работы: режим разработки и пользовательский режим.

5.5 Объекты базы данных

В СУБД версии Microsoft Access 2002 используются объекты семи различных типов:

1) таблицы;
2) запросы;
3) формы;
4) отчеты;
5) страницы;
6) макросы;
7) модули.

Таблицы — это основные объекты любой базы данных. В таблицах хранятся данные, имеющиеся в базе. Таблицы хранят структуру базы: набор полей, их типы и свойства.

Запросы. Эти объекты служат для извлечения данных из таблиц и предоставления их пользователю в удобном виде. С помощью запросов выполняют такие операции, как отбор данных, их сортировку и фильтрацию. С помощью запросов можно выполнять преобразование данных по заданному алгоритму, создавать новые таблицы, выполнять автоматическое наполнение таблиц данными, импортированными из других источников, выполнять простейшие вычисления в таблицах и многое другое.

Начинающие пользователи не сразу понимают роль запросов, поскольку все те же операции можно делать и с таблицами. Да, действительно, это так, но есть соображения удобства (в первую очередь быстродействия) и соображения безопасности.

Из соображений безопасности, чем меньше доступа к базовым таблицам имеют конечные пользователи, тем лучше. Во-первых, снижается риск того, что неумелыми действиями они по-

вредят данные в таблицах. Во-вторых, предоставив разным пользователям разные запросы, можно эффективно разграничить их доступ к данным в строгом соответствии с кругом персональных обязанностей.

Особенность запросов состоит в том, что они черпают данные из базовых таблиц и создают на их основе временную результирующую таблицу. Когда мы работаем с основными таблицами базы, мы физически имеем дело с жестким диском, то есть с очень медленным устройством (напомним, что это связано с особенностью сохранения данных, описанной выше). Когда же на основании запроса мы получаем результирующую таблицу, то имеем дело с электронной таблицей, не имеющей аналога на жестком диске, — это только образ отобранных полей и записей. Разумеется, работа с «образом» происходит гораздо быстрее и эффективнее — это еще одно основание для того, чтобы широко использовать запросы.

Недостатком упорядоченных табличных структур является сложность их обновления, поскольку при внесении новых записей нарушается упорядоченность, и приходится переделывать всю таблицу. В системах управления базами данных и эта проблема решается благодаря запросам.

Основной принцип состоит и том, что от базовых таблиц никакой упорядоченности не требуется. Все записи в основные таблицы вносятся только в естественном порядке по мере их поступления, то есть в неупорядоченном виде. Если же надо получить данные, отсортированные или отфильтрованные по тому или иному принципу, нужно просто создать соответствующий запрос.

Формы. Если запросы — это специальные средства для отбора и анализа данных, то формы — это средства для ввода данных. Смысл их тот же — предоставить пользователю средства для заполнения только тех полей, которые ему заполнять положено. Одновременно с этим в форме можно разместить специальные элементы управления (счетчики, раскрывающиеся списки, переключатели, флажки и прочие) для автоматизации ввода. Преимущества форм раскрываются особенно наглядно, когда происходит ввод данных с заполненных бланков. В этом случае форму делают графическими средствами так, чтобы она повторя-

ла оформление бланка, — это заметно упрощает работу наборщика, снижает его утомление и предотвращает появление печатных ошибок. На сопроводительном рисунке приведен пример простейшей формы для ввода данных.

С помощью форм данные можно не только вводить, но и отображать. Запросы тоже отображают данные, но делают это в виде результирующей таблицы, не имеющей почти никаких средств оформления. При выводе данных с помощью форм можно применять специальные средства оформления).

Отчеты. По своим свойствам и структуре отчеты во многом похожи на формы, но предназначены только для вывода данных, причем для вывода не на экран, а на печатающее устройство (принтер). В связи с этим отчеты отличаются тем, что в них приняты специальные меры для группирования выводимых данных и для вывода специальных элементов оформления, характерных для печатных документов (верхний и нижний колонтитулы, номера страниц, служебная информация о времени создания отчета и т.п.)

Страницы. Это специальные объекты баз данных, реализованные в последних версиях СУБД Microsoft Access. Правда, более корректно их называть страницами доступа к данным. Физически это особый объект, выполненный в коде HTML, размещаемый на Web-странице и передаваемый клиенту вместе с ней. Сам по себе этот объект не является базой данных, но содержит компоненты, через которые осуществляется связь переданной Web-страницы с базой данных, остающейся на сервере. Пользуясь этими компонентами, посетитель Web-узла может просматривать записи базы в полях страницы доступа.

Таким образом, страницы доступа к данным осуществляют интерфейс между клиентом, сервером и базой данных, размещенной на сервере. Эта база данных не обязательно должна быть базой данных Microsoft Access. Страницы доступа, созданные средствами Microsoft Access, позволяют работать также с базами данных Microsoft SQL Sewer.

Макросы. В СУБД Microsoft Access макросы, состоят из последовательности внутренних команд СУБД и являются одним из средств автоматизации работы с базой.

Модули. Модули создаются средствами внешнего языка программирования, в данном случае языка Visual Basic for

Applications(VBA). Это одно из средств, с помощью которых разработчик базы может заложить в нее нестандартные функциональные возможности, удовлетворить специфические требования заказчика, повысить быстродействие системы управления, а также уровень ее защищенности.

5.6 Работа с СУБД Microsoft Access. Общие замечания

Ниже мы рассмотрим, как в программе Microsoft Access 2002 реализованы средства разработки основных объектов базы данных, и в упражнениях познакомимся с конкретными приемами работы. Однако прежде чем приступать к освоению системы, следует учесть ряд важных замечаний, связанных с особенностями ее автоматизации.

СУБД Microsoft Access 2002 предоставляет несколько средств создания каждого из основных объектов базы. Эти средства можно классифицировать как:

- ручные (разработка объектов в режиме Конструктора);
- автоматизированные (разработка с помощью программ-мастеров);
- автоматические — средства ускоренной разработки простейших объектов.

Соотношения между этими средствами понятны: ручные средства являются наиболее трудоемкими, но обеспечивают максимальную гибкость; автоматизированные и автоматические средства являются наиболее производительными, но и наименее гибкими. Методической особенностью изучения программы Microsoft Access является тот факт, что в учебных целях для создания разных объектов целесообразно пользоваться разными средствами.

При разработке учебных таблиц и запросов рекомендуется использовать ручные средства — работать в режиме Конструктора. Использование мастеров ускоряет работу, но не способствует освоению понятий и методов.

При разработке учебных форм, отчетов и страниц доступа, наоборот, лучше пользоваться автоматизированными средствами, предоставляемыми мастерами. Это связано с тем, что для данных

объектов большую роль играет внешний вид. Дизайн этих объектов весьма трудоемок, поэтому его лучше поручить программе, а учащемуся сосредоточиться на содержательной части работы.

Разработку макросов и модулей в данном пособии мы не рассматриваем. Эти средства ориентированы на профессиональных разработчиков баз данных, поэтому в рамках общетехнического курса «Информатики» для них недостаточно места.

5.6.1 Работа с таблицами

Создание таблиц. Работа с любыми объектами начинается с окна *База данных*. На левой панели данного окна сосредоточены элементы управления для вызова всех семи типов объектов программы. Создание таблиц начинается с выбора элемента управления *Таблицы*.

На правой панели представлен список таблиц, уже имеющихся в составе базы, и приведены элементы управления для создания новой таблицы. Чтобы создать таблицу вручную, следует использовать значок *Создание таблицы в режиме конструктора*.

Окно *Конструктора* таблиц является графическим бланком для создания и редактирования структуры таблиц. В первом столбце вводят имена полей. Если свойство *Подпись* для поля не задано, то *Имя поля* станет одновременно и именем столбца будущей таблицы.

Тип для каждого поля выбирают из раскрывающегося списка, открываемого кнопкой выбора типа данных. Эта кнопка — скрытый элемент управления. Она отображается только после щелчка на поле бланка. Это надо иметь в виду — в Microsoft Access очень много таких скрытых элементов управления, которые не отображаются, пока ввод данных не начат.

При изучении приемов работы с программой Microsoft Access целесообразно специально «прощелкивать» пустые поля ее бланков левой кнопкой мыши в поисках «скрытых» элементов управления.

Нижняя часть бланка содержит список свойств поля, выделенного в верхней части. Некоторые из свойств уже заданы по умолчанию. Свойства полей не являются обязательными. Их

можно настраивать по желанию, а можно и не трогать.

При создании таблицы целесообразно (хотя и не обязательно) задать ключевое поле. Это поможет впоследствии, при организации связей между таблицами. Для задания ключевого поля достаточно щелкнуть на его имени правой кнопкой мыши и в открывшемся контекстном меню выбрать пункт *Ключевое поле*.

Выше мы говорили о том, что если первичный ключ необходим для связи с другими таблицами, но ни одно из полей не является уникальным, то первичный ключ можно создать на базе двух (или более полей). Эта операция выполняется точно так же, через контекстное меню, надо только уметь выделить сразу несколько полей. Групповое выделение выполняют при нажатой клавише SHIFT щелчками на квадратных маркерах слева от имен полей.

Закончив создание структуры таблицы, бланк закрывают (при этом система выдает запрос на сохранение таблицы), после чего дают таблице имя, и с этого момента она доступна в числе прочих таблиц в основном окне *База данных*. Оттуда ее и можено открыть в случае необходимости.

Созданную таблицу открывают в окне База данных двойным щелчком на ее значке. Новая таблица не имеет записей — только названия столбцов, характеризующие структуру таблицы. Заполнение таблицы данными производится обычным порядком. Курсор ввода устанавливается в нужную ячейку указателем мыши.

Переход к следующей ячейке можно выполнить клавишей TAB. Переход к очередной записи выполняется после заполнения последней ячейки.

В нижней части таблицы расположена Панель кнопок перехода. Ее элементами управления удобно пользоваться при навигации по таблице, имеющей большое число записей.

Начинающим пользователям Microsoft A ccess доставляет неудобство тот факт, что данные не всегда умещаются в ячейках таблицы. Шириной столбцов можно управлять методом перетаскивания их границ. Удобно использовать автоматическое форматирование столбцов «по содержимому». Для этого надо установить указатель мыши на границу между столбцами (в строке заголовков столбцов), дождаться, когда указатель сменит форму, и выполнить двойной щелчок. Это общесистемный прием Windows,

и им можно пользоваться в данной программе, как и во многих других.

После наполнения таблицы данными сохранять их не надо — все сохраняется автоматически. Однако если при работе с таблицей произошло редактирование ее макета (например, изменялась ширина столбцов), СУБД попросит подтвердить сохранение этих изменений.

Если возникнет необходимость изменить структуру таблицы (состав полей или их свойства), таблицу надо открыть в режиме Конструктора. Для этого ее следует выделить в окне База данных и щелкнуть на кнопке Конструктор.

Если на этапе проектирования базы данных была четко разработана структура таблиц, то создание таблиц с помощью Конструктора происходит очень быстро и эффективно. Даже без использования автоматизированных средств создание основы для достаточно крупных проектов происходит в считанные минуты — это ценное свойство СУБД Microsoft Access, но оно реализуется при непременном условии тщательной предварительной подготовки.

Создание межтабличных связей. Если структура базы данных продумана заранее, а связи между таблицами намечены, то создание реляционных отношений между таблицами выполняется очень просто. Вся необходимая работа происходит в специальном окне Схема данных и выполняется с помощью мыши. Окно *Схема данных* открывают кнопкой на панели инструментов или командой *Сервис* ► *Схема данных* (если в меню Сервис не видно соответствующего пункта, следует раскрыть расширенную часть меню).

Образовавшаяся межтабличная связь отображается в окне Схема данных в виде линии, соединяющей два поля разных таблиц. При этом одна из таблиц считается главной, а другая — связанной. Главная — это та таблица, которая участвует в связи своим ключевым полем (название этого поля на схеме данных отображается полужирным шрифтом).

У связи два основных назначения. Первое — обеспечение *целостности данных*, а второе — автоматизация задач обслуживания базы.

Рассмотрим пример нарушения целостности данных. Пусть

в некой базе данных имеется таблица «Клиенты», содержащая сведения о потребителях товаров и таблица «Заказы», где указаны товары приобретаемые клиентами. В таблице «Клиенты», где каждый клиент уникален, кто-то удалит запись для одного из клиентов, но не сделает этого в таблице «Заказы». Получится, что согласно таблице «Заказы» некто, не имеющий ни имени, ни адреса, а только абстрактный код, делал заказы. Узнать по коду, кто же это был на самом деле, будет невозможно — *произошло нарушение целостности данных*.

В данном случае можно применить три подхода: либо вообще ничего не делать для защиты целостности данных, либо запретить удаление данных из ключевых полей главных таблиц, либо разрешить его, но при этом адекватно обработать и связанные таблицы. Вручную сделать это чрезвычайно трудно, поэтому и нужны средства автоматизации.

Связь между таблицами позволяет:

1) либо исключить возможность удаления или изменения данных в ключевом поле главной таблицы, если с этим полем связаны какие-либо поля других таблиц;

2) либо сделать так, что при удалении (или изменении) данных в ключевом поле главной таблицы автоматически (и абсолютно корректно) произойдет удаление или изменение соответствующих данных в полях связанных таблиц.

Для настройки свойств связи надо в окне *Схема данных* выделить линию, соединяющую поля двух таблиц, щелкнуть на ней правой кнопкой мыши и открыть контекстное меню связи, после чего выбрать в нем пункт *Изменить связь* — откроется диалоговое окно *Изменение связей*. В нем показаны названия связанных таблиц и имена полей, участвующих в связи (здесь же их можно изменить), а также приведены элементы управления для обеспечения условий целостности данных.

Если установлен только флажок *Обеспечение целостности данных*, то удалять данные из ключевого поля главной таблицы нельзя. Если вместе с ним включены флажки *Каскадное обновление связанных полей* и *Каскадное удаление связанных записей*, то, соответственно, операции редактирования и удаления данных в ключевом поле главной таблицы разрешены, но сопровождаются автоматическими изменениями в связанной таблице.

Таким образом, смысл создания реляционных связей между таблицами состоит, с одной стороны, в защите данных, а с другой стороны — в автоматизации внесения изменений сразу в несколько таблиц при изменениях в одной таблице.

5.6.2 Работа с запросами

Если структура базы данных предприятия хорошо продумана, то исполнители, работающие с базой, должны навсегда забыть о том, что в базе есть таблицы, а еще лучше, если они об этом вообще ничего не знают. Таблицы — слишком ценные объекты базы, чтобы с ними имел дело кто-либо, кроме разработчика базы.

Если исполнителю надо получить данные из базы, он должен использовать специальные объекты — запросы. Все необходимые запросы разработчик базы должен подготовить заранее. Если запрос подготовлен, надо открыть панель Запросы в окне База данных, выбрать его и открыть двойным щелчком на значке — откроется результирующая таблица, в которой исполнитель найдет то, что его интересует.

В общем случае результирующая таблица может не соответствовать ни одной из базовых таблиц базы данных. Ее поля могут представлять набор из полей разных таблиц, а ее записи могут содержать отфильтрованные и отсортированные записи таблиц, на основе которых формировался запрос.

В учебных целях запросы лучше готовить вручную, с помощью Конструктора. Как и в случае с таблицами, для этого есть специальный значок в окне *База данных*. Он называется *Создание запроса в режиме конструктора* и открывает специальный бланк, называемый бланком запроса по образцу. За этим длинным названием скрывается тот приятный факт, что, хотя запросы к таблицам баз данных пишутся на специальном языке программирования — SQL, пользователям Microsoft Access изучать его не обязательно, а большинство операций можно выполнить щелчками кнопок мыши и приемом перетаскивания в бланке.

Бланк запроса по образцу состоит из двух областей. В верхней отображается структура таблиц, к которым запрос адресован, а нижняя область разбита на столбцы — по одному столбцу на каждое поле будущей результирующей таблицы.

Идея формирования запроса по образцу чрезвычайно проста. С помощью контекстного меню на верхней половине бланка открывают те таблицы, к которым обращен запрос. Затем в них щелкают двойными щелчками на названиях тех полей, которые должны войти в результирующую таблицу. При этом автоматически заполняются столбцы в нижней части бланка. Сформировав структуру запроса, его закрывают, дают ему имя и в дальнейшем запускают двойным щелчком на значке в окне База данных.

Порядок действий, рассмотренный выше, позволяет создать простейший запрос, называемый запросом на выборку. Он позволяет выбрать данные из полей таблиц, на основе которых запрос сформирован.

5.6.3 Упорядочение записей в результирующей таблице

Если необходимо, чтобы данные, отобранные в результате работы запроса на выборку, были упорядочены по какому-либо полю, применяют сортировку. В нижней части бланка имеется специальная строка Сортировка. При щелчке на этой строке открывается кнопка раскрывающегося списка, в котором можно выбрать метод сортировки: по возрастанию или по убыванию. В результирующей таблице данные будут отсортированы по тому полю, для которого задан порядок сортировки.

Возможна многоуровневая сортировка — сразу по нескольким полям. В этом случае данные сначала сортируются по тому полю, которое в бланке запроса по образцу находится левее, затем по следующему полю, для которого включена сортировка, и так далее слева направо. Соответственно, при формировании запроса надо располагать поля результирующей таблицы не как попало, а-с учетом будущей сортировки. В крайнем случае, если запрос уже сформирован и надо изменить порядок следования столбцов, пользуются следующим приемом:

а) выделяют столбец щелчком на его заголовке (кнопку мыши отпускают);

б) еще раз щелкают на заголовке уже выделенного столбца (но кнопку не отпускают);

в) перетаскивают столбец в другое место.

Управление отображением данных в результирующей таб-

лице. В нижней части бланка запроса по образцу имеется строка *Вывод на экран*. По умолчанию предполагается, что все поля, включенные в запрос, должны выводиться на экран, но это не всегда целесообразно. Например, бывают случаи, когда некое поле необходимо включить в запрос, например, потому, что оно является полем сортировки, но в то же время, нежелательно, чтобы пользователь базы видел его содержание. В таких случаях отображение содержимого на экране подавляют сбросом флажка *Вывод на экран*.

Использование условия отбора. Дополнительным средством, обеспечивающим отбор данных по заданному критерию, является так называемое *Условие отбора*. Соответствующая строка имеется в нижней части бланка запроса по образцу. Для каждого поля в этой строке можно задать индивидуальное условие.

5.6.4 Другие виды запросов

Мы рассмотрели запросы на выборку. Это самые простые и в то же время наиболее распространенные виды запросов. Однако существуют и другие виды запросов, некоторые их которых выполняются на базе предварительно созданного запроса на выборку. К ним относятся, например:

- запросы с параметром (интересны тем, что критерий отбора может задать сам пользователь, введя нужный параметр при вызове запроса);
- итоговые запросы, назначение которых отдаленно напоминает итоговые функции электронных таблиц (производят математические вычисления по заданному полю и выдают результат);
- запросы на изменение — позволяют автоматизировать заполнение полей таблиц;
- перекрестные запросы, позволяющие создавать результирующие таблицы на основе расчетов, полученных при анализе группы таблиц;
- специфические запросы SQL — запросы к серверу базы данных, написанные на языке запросов SQL.

5.7 Работа с формами

С одной стороны, формы позволяют пользователям вводить данные в таблицы базы данных без непосредственного доступа к самим таблицам. С другой стороны, они позволяют выводить результаты работы запросов не в виде скупых результирующих таблиц, а в виде красивых форм. В связи с таким разделением существует два вида организации структуры форм: на основе таблицы и на основе запроса, хотя возможен и комбинированный подход, — это вопрос творчества.

5.7.1 Создание форм с помощью мастера

Автоматизированные средства предоставляет Мастер форм — специальное программное средство, создающее структуру формы в режиме диалога с разработчиком. Мастер форм можно запустить из окна База данных щелчком на значке Создание формы с помощью мастера на панели *Формы*.

На первом этапе работы Мастера форм выбирают таблицы и поля, которые войдут в будущую форму.

На втором этапе выбирается внешний вид формы.

На третьем этапе выбирается стиль оформления формы.

На последнем этапе выполняется сохранение формы под заданным именем. Здесь же можно включить переключатель Изменить макет формы, который открывает только что созданную форму в режиме Конструктора. Этим удобно воспользоваться в учебных целях, чтобы рассмотреть структуру формы на готовом примере.

5.7.2 Структура формы

Форма имеет три основных раздела: область заголовка, область данных и область примечания. Линии, разделяющие разделы, перетаскиваются по вертикали с помощью мыши — это позволяет изменять размеры разделов так, как требуется.

Разделы заголовка и примечания имеют чисто оформительское назначение — их содержимое напрямую не связано с таблицей или запросом, на котором основана форма. Раздел данных имеет содержательное значение — в нем представлены элементы

управления, с помощью которых выполняется отображение данных или их ввод. Разработчик формы может разместить здесь дополнительные элементы управления для автоматизации ввода данных (переключатели, флажки, списки и другие, типичные для приложений Windows).

Элементы управления формы. Элементы управления, которыми может пользоваться разработчик, представлены на Панели элементов. Ее открывают щелчком на соответствующей кнопке панели инструментов Microsoft Access или командой Вид ► Панель элементов.

Выбор элемента управления выполняется одним щелчком на его значке в Панели элементов, после чего следующим щелчком в поле формы отмечается место, куда он должен быть поставлен. Вместе с элементом в поле формы вставляется его при* соединенная надпись. По умолчанию эта надпись стандартная, например, для переключателей это Переключатель1, Переключатель2 и т.д. Редактированием свойства элемента управления (доступ к свойствам открывается через контекстное меню) можно дать элементу управления более содержательную подпись.

Основными элементами оформления формы являются текстовые надписи и рисунки. Для создания в форме текстовых надписей служат два элемента управления — *Надпись* и *Поле*. В качестве надписи можно задать произвольный текст. Элемент *Поле* отличается тем, что в нем может отображаться содержимое одного из полей таблицы, на которой основана форма, то есть при переходе от записи к записи текст может меняться.

Для создания графических элементов оформления служат элементы управления *Рисунок*, *Свободная рамка объекта* и *Присоединенная рамка объекта*. Рисунок выбирается из графического файла и вставляется в форму. Элемент *Свободная рамка объекта* отличается тем, что это не обязательно рисунок — это может быть любой другой объект OLE, например мультимедийный. Элемент *Присоединенная рамка объекта* тоже в какой-то степени может служить для оформления формы, но его содержимое берется не из назначенного файла, а непосредственно из таблицы базы данных (если она имеет поле объекта OLE). Естественно, что при переходе между записями содержимое этого элемента будет меняться.

5.7.3 Дизайн формы

Если таблицы базы данных обычно скрыты в ее недрах, формы базы данных — это средства внешнего интерфейса, с которым работают пользователи. Поэтому к формам предъявляются повышенные требования по дизайну.

В первую очередь, все элементы управления форм должны быть аккуратно выровнены. Это обеспечивается командой *Формат* ► *Выровнять*. Если нужно равномерно распределить элементы управления по полю формы, используют средства меню *Формат* ► *Интервал по горизонтали* или *Формат* ► *Интервал по вертикали.*

Ручное изменение размеров и положения элементов управления тоже возможно, но редко приводит к качественным результатам. При работе вручную используют перетаскивание маркеров, которые видны вокруг элемента управления, когда он выделен. Особый статус имеет маркер левого верхнего угла. Обычно элементы управления перетаскиваются вместе с присоединенными к ним надписями. Перетаскивание с помощью этого маркера позволяет оторвать присоединенную надпись от элемента.

Существенную помощь при разработке дизайна формы оказывает вспомогательная сетка. Ее отображение включают командой *Вид* ► *Сетка*. Автоматическую привязку элементов к узлам сетки включают командой *Форма* ► *Привязать к сетке*.

5.7.4 Управление последовательностью перехода

Пользователь, для которого, собственно, и разрабатывается форма, ожидает, что ввод данных в нее должен происходить по элементам управления слева направо и сверху вниз. Однако при проектировании сложных форм, когда в процессе дизайна элементы управления многократно перемещаются с места на место, очень легко перепутать их последовательность и создать неудобный порядок ввода данных.

Физически последовательность перехода — это порядок перехода к следующему полю по окончании работы с предыдущим. Она легко проверяется с помощью клавиши TAB. Если при последовательных нажатиях этой клавиши фокус ввода «мечется»

по всей форме, то последовательность перехода нерациональна и нуждается в корректировке.

Для управления последовательностью перехода служит диалоговое окно *Последовательность перехода*. В нем представлен список элементов управления формы. Порядок элементов в списке соответствует текущему порядку перехода. Изменение порядка перехода выполняется перетаскиванием в два приема:

- щелчком на кнопке маркера слева от названия выделяется элемент управления (кнопка мыши отпускается);
- после повторного щелчка с перетаскиванием элемент перемещается на новое место.

Закончив разработку макета формы, ее следует закрыть и сохранить под заданным именем. После открытия формы в окне База данных с ней можно работать: просматривать или редактировать данные из базовой таблицы. Проверку последовательности перехода выполняют клавишей TAB.

5.8 Работа со страницами доступа к данным

Страницы (страницы доступа к данным) — новый объект баз данных, вошедший в последние версии Microsoft Access. Как и формы, этот объект служит для обеспечения доступа к данным, содержащимся в базе, но здесь речь идет об удаленном доступе, например о доступе через Интернет или через корпоративную сеть intranet.

С помощью страниц доступа к данным решается вопрос передачи данных из базы удаленному потребителю. Обычно базы данных имеют очень большие размеры, и напрямую передавать их через медленные каналы связи непрактично. В то же время, большинство современных Web-браузеров пока не имеют функций для работы с базами данных, размещенными на серверах. Таким образом, страницы доступа выполняют как бы посредническую функцию. Они имеют небольшой размер, содержат удобные элементы управления для навигации в базе данных, могут быть записаны в формате кода HTML, переданы по медленным каналам связи и воспроизведены в стандартном браузере. В связи с тем, что по формату они являются Web-документами, их нетрудно встроить в любой Web-документ, например разместить на

Web-странице.

От прочих объектов базы данных страницы доступа отличаются тем, что имеют двойную природу. Прочие объекты базы являются внутренними. Так, например, мы не можем выделить ни таблицу, ни запрос, ни форму в виде самостоятельного файла. Эти объекты размещаются где-то внутри файла базы данных, Но операционная система компьютера работать с ними не может, поскольку это не файлы. С ними работает лишь сама система управления базой данных. Страница же представлена двумя объектами — внутренним объектом базы (его можно редактировать) и внешним объектом — файлом в формате HTML. Запись этого файла происходит при сохранении спроектированной страницы доступа.

5.8.1 Создание страницы доступа к данным

Для страниц доступа, как и для форм, важную роль играет внешний вид, поэтому создавать их удобно с помощью мастера. Мастер страниц запускается щелчком на значке *Создание страницы доступа к данным с помощью мастера*.

На первом этапе работы *Мастера форм* выбирают таблицы (или запросы), в их составе — поля, к которым должна обеспечить доступ страница.

Второй этап работы мастера предназначен для управления группировкой данных. Эта возможность предусмотрена для доступа к базам, содержащим большие объемы данных. Если значения в некотором поле часто повторяются, имеет смысл объединить соответствующие им записи в группу. В результате группировки образуется иерархическая структура. Она может иметь несколько уровней вложения.

Вторая страница мастера предоставляет элементы управления для выбора полей, по которым производится группировка, и управления глубиной уровней группировки. Если просмотреть в режиме Конструктора страницу, имеющую уровни группировки, то можно убедиться, что для каждого уровня группировки в структуре объекта образуется отдельный раздел, то есть различные уровни группировки могут быть дополнены различными элементами управления экранной Web-формы.

На третьем этапе выбирается метод упорядочения отображаемых данных. Возможно задание до четырех полей сортировки, причем сортировка возможна как по возрастанию, так и по убыванию.

На последнем этапе выполняется сохранение страницы под заданным именем. Здесь же можно перейти в режим Конструктора, включив переключатель Изменить макет страницы. В случае изменения макета к странице можно применить одну из тем оформления, входящих в состав пакета Microsoft Office XP.Темы оформления представляют собой совокупности стилей оформления текстов, фоновых узоров и специфических элементов оформления страницы (маркеров, линий и прочих).

Редактирование страницы доступа к данным. Редактирование созданной страницы доступа выполняется в режиме Конструктора теми же приемами, которые были описаны для форм. Основными отличиями являются:

- наличие большего количества разделов (связано с возможностью группировки);
- расширенный состав элементов управления на *Панели элементов* (*Вид* ► *Панель элементов*);
- иной механизм перетаскивания элементов управления и присоединенных надписей (элементы управления перетаскиваются вместе с присоединенными надписями, но присоединенные надписи перетаскиваются отдельно от элементов управления).

5.9 Работа с отчетами

Отчеты во многом похожи на формы и страницы доступа к данным, но имеют иное функциональное назначение — они служат для форматированного вывода данных на печатающие устройства и, соответственно, при этом должны учитывать параметры принтера и параметры используемой бумаги.

Большая часть того, что было сказано о формах, относится и к отчетам. Здесь также существуют средства автоматического, автоматизированного и ручного проектирования. Средства автоматического проектирования реализованы автоотчетами (*База данных* ► *Создать* ► *Новый отчет* ► *Автоотчет в столбец*). Кроме автоотчетов «в столбец» существуют «ленточные» автоот-

четы. Разницу между ними нетрудно увидеть, поставив эксперимент.

Средством автоматизированного создания отчетов является *Мастер отчетов*. Он запускается двойным щелчком на значке *Создание отчета с помощью мастера* в окне *База данных*. Мастер отчетов работает в шесть этапов. При его работе выполняется выбор базовых таблиц или запросов, на которых отчет базируется, выбор полей, отображаемых в отчете, выбор полей группировки, выбор полей и методов сортировки, выбор формы печатного макета и стиля оформления.

Структура готового отчета отличается от структуры формы только увеличенным количеством разделов. Кроме разделов заголовка, примечания и данных, отчет может содержать разделы верхнего и нижнего колонтитулов. Если отчет занимает более одной страницы, эти разделы необходимы для печати служебной информации, например номеров страниц. Чем больше страниц занимает отчет, тем важнее роль данных, выводимых на печать через эти разделы. Если для каких-то полей отчета применена группировка, количество разделов отчета увеличивается, поскольку оформление заголовков групп выполняется в отдельных разделах.

Редактирование структуры отчета выполняют в режиме *Конструктора* (режим запускается кнопкой *Конструктор* в окне *База данных*). Приемы редактирования те же, что и для форм. Элементы управления в данном случае выполняют функции элементов оформления, поскольку печатный отчет в отличие от электронных форм и Web-страниц не интерактивный объект, Размещение элементов управления выполняют с помощью *Панели элементов* (*Вид* ► *Панель элементов*), которая по составу практически не отличается от *Панели элементов* формы.

Важной особенностью отчетов является наличие средства для вставки в область верхнего или нижнего колонтитула текущего номера страницы и полного количества страниц. Эту операцию выполняют с помощью диалогового окна *Номера страниц* (*Вставка* ► *Номера страниц*).

6 МАТЕМАТИЧЕСКИЙ ПРОЦЕССОР MATHCAD КАК СИСТЕМА ПРОГРАММИРОВАНИЯ

Программный комплекс **MathCad** (Mathematical Computer Aided Design — математическое автоматизированное проектирование) — разработка фирмы Mathsoft Engineering & Education. Комплекс предназначен для автоматизации решений широкого круга задач, связанных с математическими расчетами. Это многофункциональная вычислительная среда, снабженная дружественным, во многом интуитивно понятным интерфейсом.

При запуске **MathCad** на экране появляется окно интерфейса (рис. 6.1).

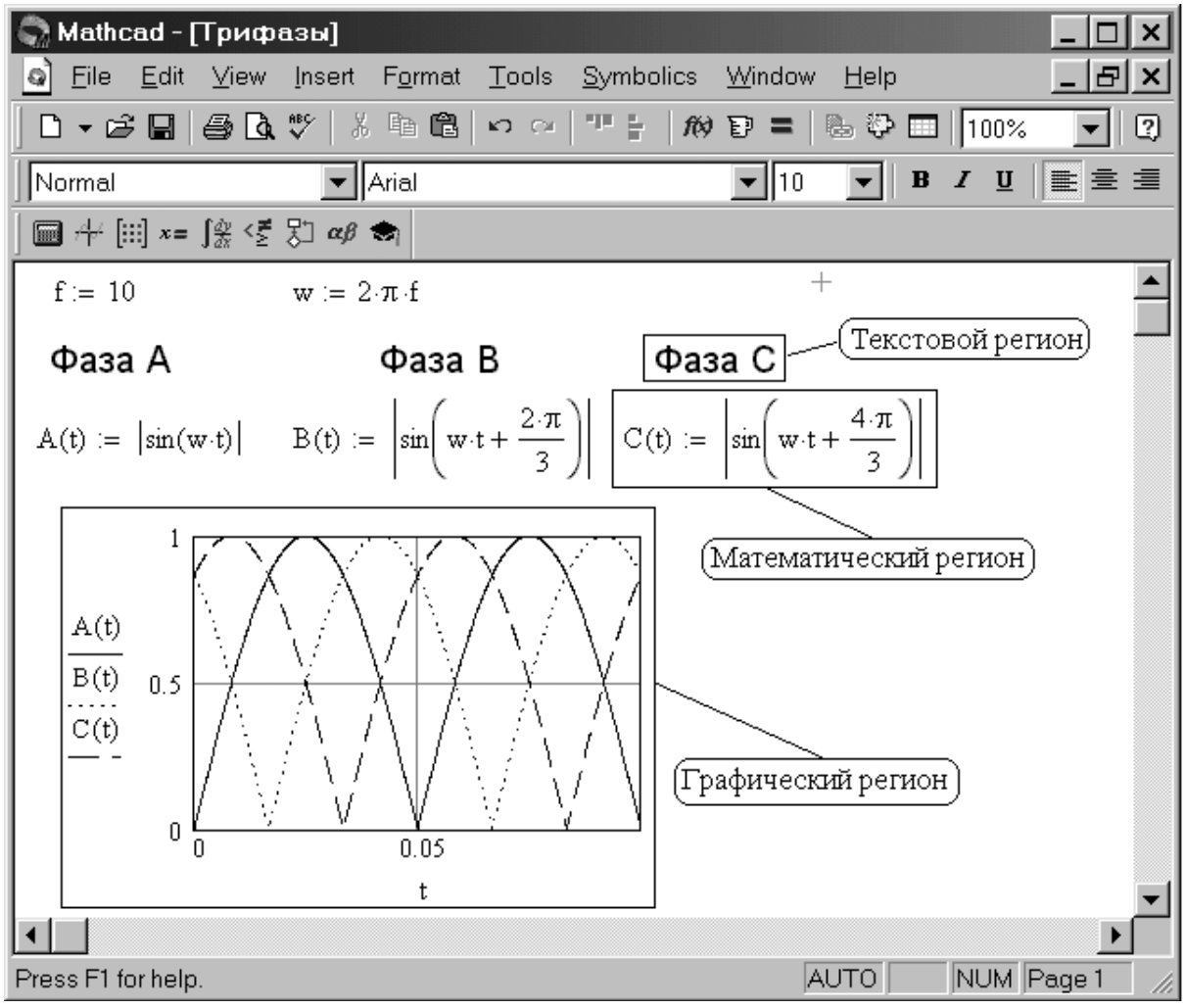


Рис. 6.1 — Окно интерфейса MathCad с загруженным документом

6.1 Рабочий лист и регионы

В поле основного окна отображается продолжаемый вправо и вниз бесконечный рабочий лист (Worksheet). На рабочем листе в произвольном порядке располагаются ***регионы***. Существует три типа регионов:

1) математические,
2) графические,
3) текстовые.

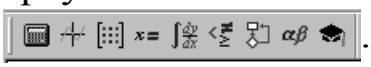
Математические регионы содержат формульные выражения, операторы назначения и индикации результатов. С математическими регионами связаны вычислительные модули — участки исполняемого кода. Преобразование входной информации, занесенной в математический регион, в реализуемый код выполняется ***интерпретатором*** **MathCad**. В отличие от компиляторов, выполняющих просмотр текста программы целиком, интерпретатор обрабатывает каждый регион в отдельности, что, собственно, и дает возможность использовать **MathCad** как мощный калькулятор для разовых вычислений. Последовательность обработки математических регионов — ***слева направо и сверху вниз*** по рабочему листу. К моменту выполнения математического региона все объекты, участвующие в его выражениях, должны быть определены либо непосредственно в этом регионе, либо в предшествующих по времени выполнения регионах. Для удобства ввода информации в математические регионы в **MathCad** широко используется система шаблонов.

Текстовые регионы содержат комментирующий текст и служат для оформления рабочего листа как удобочитаемого документа. Поскольку никакой исполнительный код с текстовыми регионами не связывается, их расположение на рабочем листе ничем не регламентировано. В текстовые регионы можно помещать изображения.

Графические регионы предназначены для вывода информации в виде графиков и изображений. Для программирования графического региона также применяются шаблоны.

6.2 Инструментальные панели и шаблоны

Интерфейс **MathCad** предоставляет пользователю представительный набор шаблонов для программирования математических и графических регионов. Для вызова шаблонов и занесения специфических символов операций используются инструментальные панели с виртуальными кнопками или «горячие» клавиши — акселераторы.

Панели инструментов объединены в 9 тематических групп (палитр). Каждая группа может быть вызвана на экран переводом в «нажатое» состояние виртуальной кнопки — выключателя на линейке вызова палитр:  . Соответствие кнопок линейки вызываемым панелям показано на рисунке 6.2.

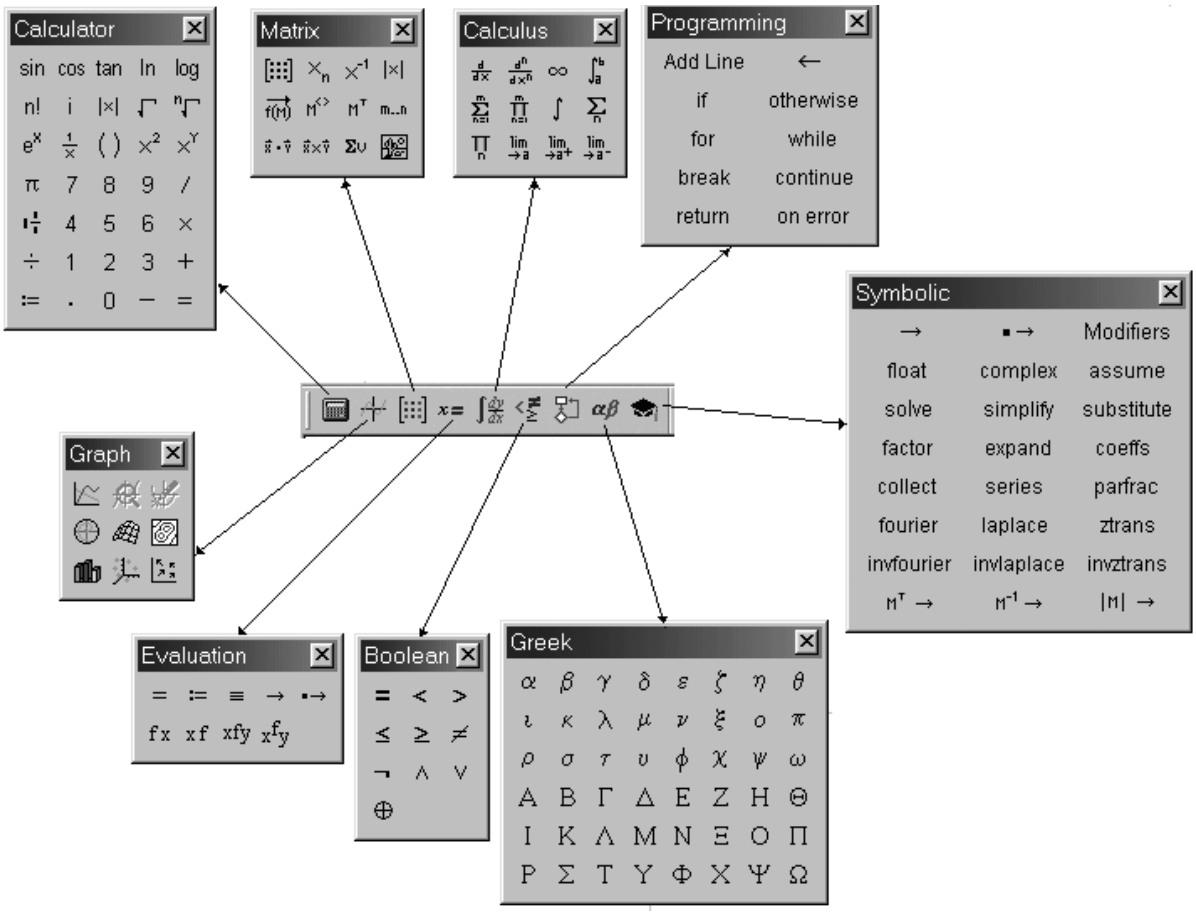


Рис. 6.2 — Панели инструментов и линейка их вызова

Всего на инструментальных панелях представлено 169 позиций, воздействие на которые вызывает вставку определенного

шаблона или символа в точку расположения курсора на рабочем листе. Как видно из рисунка 6.2, некоторые позиции дублируются, присутствуя на разных панелях, некоторые предназначены для вставки обычных символов, имеющихся на клавиатуре, так что более удобно вводить непосредственным набором с клавиатуры, нежели через панель инструментов. Назначение ряда инструментов очевидно из соответствующих пиктограмм и не требует пояснений.

Здесь мы рассмотрим лишь те инструменты, применение которых возможно при выполнении заданий контрольной работы, информацию обо всех инструментах можно получить, воспользовавшись справочной системой **MathCad**.

В составе пакета **MathCad** имеется достаточно обширный набор встроенных функций. Вставка встроенных функций на рабочий лист может быть выполнена двумя путями:

1) набором имени функции на клавиатуре;

2) выбором функции из списка в окне диалога, открывающегося после воздействия на элемент управления с пиктограммой  .

Второй вариант представляется более удобным, так как на рабочий лист помещается шаблон функции с правильным названием и шаблоном списка параметров. Кроме того, в открывающемся окне диалога выбора функции имеется кнопка вызова контекстуально-зависимой справки.

Итак, вызов шаблона может производиться как выбором пиктограммы инструмента на панели, так и с помощью «горячих клавиш». Каким путем вызова шаблона воспользоваться — вопрос удобства. В таблице 6.1 представлены пиктограммы некоторых инструментов и комбинации клавиш быстрого вызова.

Таблица 6.1

Описание операции	**Пиктограмма**	**Клавиши**	**Панель**
Извлечение квадратного корня	$\sqrt{\square}$	\	Calculator
Возведение в степень	$\mathbf{x}^{\mathbf{Y}}$	Shift+6	Calculator
Ввод матрицы или вектора		Ctrl+M	Matrix
Ввод индекса (в элементах массива)	$\mathbf{x_n}$	[	Matrix

Окончание табл. 6.1

Описание операции	Пикто-грамма	Кла-виши	Панель
Определитель, модуль, абсолютное значение	|x|	|	Matrix*
Выделение столбца матрицы	$M^{\langle\rangle}$	Ctrl+6	Matrix
Создание ранжированной переменной	m..n	;	Matrix
Векторизация функции	$\overrightarrow{f(M)}$	Ctrl+ –	Matrix
Оператор назначения — присваивания	:=	:	Evaluation*
Оператор глобального назначения — присваивания	≡	~	Evaluation
Индикация значения в символьной форме	→	Ctrl+.	Evaluation*
График в прямоугольных координатах		Shift+2	Graph
* *Инструмент представлен на нескольких панелях, действие везде одинаковое*			

6.3 Операторы определения (назначения) объектов и индикации значений

Операторы определения или, что эквивалентно, назначения, объектов размещаются в математических регионах и выполняют три функции:

1) создают объект, связывая с ним имя — идентификатор;

2) определяют тип объекта;

3) инициализируют объект значением.

В качестве определяемого объекта могут выступать переменные и функции. Внешне оператор определения выглядит так же, как оператор присваивания языка программирования **Pascal**. Видимо, по этой причине его иногда называют оператором присваивания, хотя с функциональной точки зрения это не совсем так. В отличие от оператора присваивания универсальных языков (**Pascal**, **Си**, **C++** и т.д.), оператор назначения **MathCad** выполняет функцию ***определения типа*** объекта, т.е. ту функцию, которая в универсальных языках реализуется специальными директивами. На рисунке 6.3 приведены примеры применения операторов определения объектов различного типа: вещественного числа **a**,

комплексного числа **z**, вектора **b**, матрицы **A** с инициализацией их значений и функции **conj(x).**

$$a := 17 \qquad b := \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} \qquad A := \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 16 \\ 1 & 3 & 9 & 27 \end{pmatrix}$$

$$z := 3 + 4i$$

$$conj(x) := Re(x) - i \cdot Im(x)$$

Рис. 6.3 — Операторы определения и инициализации объектов различного типа

Тип объекта, соответствующего тому или иному имени-идентификатору, определяется типом правой части. Далее любые операции с объектом интерпретируются в соответствии с его определенным типом.

6.3.1 Функции, определяемые пользователем

В примере на рисунке 6.3 определена функция **conj(x).** В этом примере функция получает в качестве аргумента комплексное число **x** и возвращает в качестве значения комплексно — сопряженное число. Функция, задаваемая подобным образом, называется ***функцией, определяемой пользователем.*** В скобках после имени функции находится список ***формальных аргументов***. В рассмотренном примере этот список состоит из единственной переменной **x**. При вызове функций формальные параметры заменяются фактическими. До вызова функции, переменные, перечисленные в списке формальных аргументов, будут оставаться неопределенными. Тип объекта, возвращаемого функцией, т.е. связанного с ее именем, определяется по контексту определения точно так же, как это происходит при определении переменной. В составе определения функции могут присутствовать переменные, не передаваемые через список формальных параметров. В этом случае такие переменные должны быть определены до оператора назначения функции (рис. 6.4). Этот прием ис-

$$a := 3 \qquad b := 4$$

$$s(t, w) := a \cdot \cos(w \cdot t) + b \cdot \sin(w \cdot t)$$

Рис. 6.4 — Определение функции с передачей части параметров через глобальные переменные

пользован в примере выполнения контрольной работы для сокращения числа формальных параметров. Необходимо отметить, что передача параметров через глобальные переменные в общем случае нежелательна, т.к. нарушает изолированность функции и может служить источником трудно обнаруживаемых ошибок.

6.3.2 Виды операторов назначения MathCad

Операторы назначения в **MathCad** имеют три разновидности:

:= оператор назначения (стандартный);

≡ оператор глобального назначения;

← оператор локального назначения в блоке.

Все три разновидности операторов, помимо операции присваивания значения правого операнда левому, выполняют передачу типа левому операнду.

Оператор:=, схожий по начертанию с оператором присваивания **Pascal**, связывает значение и тип объекта с его идентификатором во всех регионах, выполняемых ***после*** региона, содержащего этот оператор.

Оператор ≡ действует на ***все*** регионы рабочего листа независимо от их расположения относительно региона, содержащего этот оператор.

Оператор ← предназначен для использования в изолированных блоках, формируемых при программировании в пределах одного региона. Действие этого оператора распространяется ***только на тот регион***, в котором он расположен. На рисунке 6.5 показан пример переопределения типа и значения объекта **a**.

Действие оператора глобального назначения ≡ распространяется на все пространство рабочего листа. Появление в регионе оператора назначения:= вызывает переопределение типа объекта, сохраняющееся до появления следующего оператора назначения, которым может быть либо еще один оператор:=, либо (как в примере) оператор глобального назначения ≡.

Оператор локального назначения ← не оказывает никакого действия на тип и значение объекта **a** за пределами блока, выделенного слева жирной вертикальной чертой. Такие блоки созда-

ются при использовании инструментальной панели ***Programming*** и занимают один регион.

Рис. 1.5 — Области действия операторов назначения

Определенные однажды объекты могут быть переопределены в последующих регионах. Таким образом, при интерпретации действий с объектом актуально последнее, ближайшее к точке выполнения определение его типа.

Замечание. Если на рабочем листе есть несколько операторов глобального назначения ≡, то свойство «глобальности» сохраняет ***только последний*** из них.

6.3.3 Операторы индикации значений

Весьма часто при выполнении вычислений требуется контролировать значения того или иного объекта, либо результата выполнения выражения. Эта задача решается операторами инди-

кации значений. В **MathCad** существуют два вида операторов индикации:

= оператор индикации числовых значений;

→ оператор индикации символьных значений.

Пример использования операторов индикации числовых значений имеется на приведенном выше рисунке 6.5. Численные значения компонентов объекта индицируются в соответствии с принятым форматом отображения типа: число, комплексное число, вектор, матрица.

Оператор индикации символьных значений применяется для вывода результатов символьных преобразований, значений объектов, в составе которых имеются компоненты с неопределенными числовыми значениями. Такими объектами являются, например, функции с формальными параметрами, формульные выражения при выполнении над ними операций ***символьных преобразований***. Примеры символической индикации в применении к функциям содержатся в прилагаемом примере выполнения контрольной работы.

6.4 Вывод графиков в MathCad

Вывод графической информации производится в графических регионах с использованием шаблонов, создаваемых инструментами панели **Graph.** Для выполнения контрольной работы нам потребуется только шаблон графика в прямоугольной (декартовой) системе координат (пиктограмма  или комбинация клавиш **Shift+2**). Вид шаблона изменяется по мере его заполнения. Первоначальный вид шаблона показан на рисунке 6.6, *а*. Здесь имеется поле ввода выражения для переменной, значения которой отображаются по вертикальной оси (**Поле Y**) и точка ввода выражения для переменной, отображаемой по горизонтальной оси (**Поле X**), изображена прямоугольная область «экрана», на которой будет строиться график. После занесения выражений отображаемых переменных на шаблоне появляются поля ввода границ диапазонов этих переменных, в пределах которых происходит построение графика (рис. 6.6, *б*). Если выражение для **y(x)** определено к моменту заполнения шаблона, то график выдается сразу после его занесения в поле **Y**, при этом границы

изменения **x** устанавливаются по умолчанию, а границы изменения **y** определяются автоматически на основе его фактического динамического диапазона **y(x)**. Как правило, такое масштабирование по умолчанию не устраивает пользователя. В этом случае нужно вернуться в режим редактирования шаблона, и, поместив курсор внутрь шаблона установить желаемые значения в полях **Min x**, **Max x, Min y, Max y.**

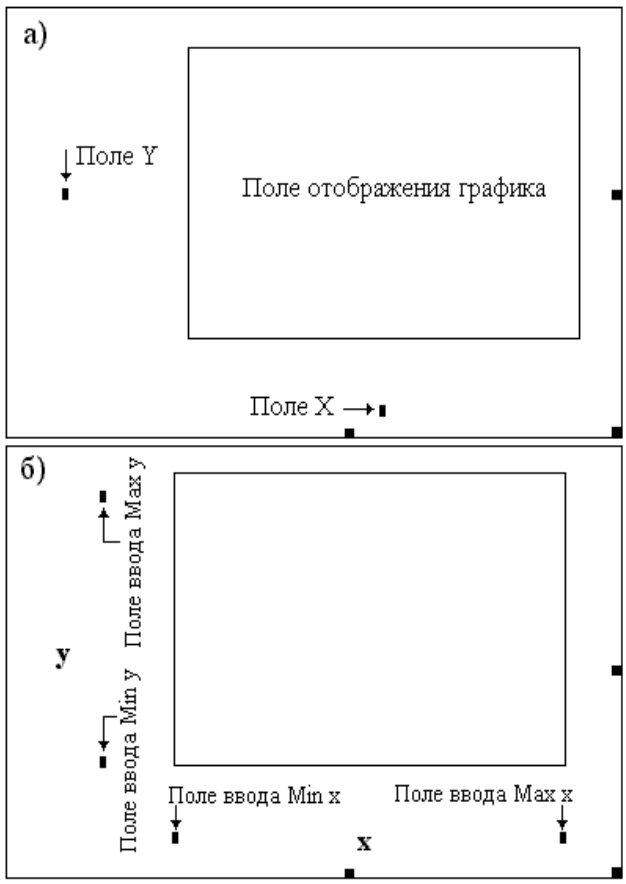


Рис. 6.6 — Начальные фазы формирования шаблона вывода графика

На всех этапах редактирования, когда курсор находится внутри шаблона, отображается внешняя рамка с точками «растяжки» (рисунок 6.6), что позволяет в любой момент изменить размер графика, при этом график автоматически становится более или менее подробным.

В версиях **MathCad** 8 и выше при построении графиков непрерывных зависимостей **y(x)** производится автоматический выбор шага независимой переменной **x**. Если потребуется, процедуру автоматического выбора шага можно исключить. Для этого надо перед графическим регионом задать массив **x** с помощью ранжированных переменных с необходимым шагом изменения, например, **x:= 0,0.1.. 1**. В этом случае при построении графика значения **y(x)** будут вычисляться только в точках **x = 0; x = 0.1; x = 0.2** и т.д. до **x = 1**. Массивы значений $\mathbf{x_i}$ и $\mathbf{y(x_i)}$ будут образовывать узловые точки графика. По умолчанию узловые точки графика будут соединены отрезками прямой.

Графические регионы позволяют строить в одних осях несколько графиков одновременно. Для этого необходимо в **Поле Y** поместить список выводимых переменных, разделенных запятыми. Запятая при этом не отображается, а каждое имя вводимой

переменной помещается на новую строку. На рисунке 6.7 приведен пример готового графика, отображающего две различные зависимости в одних осях.

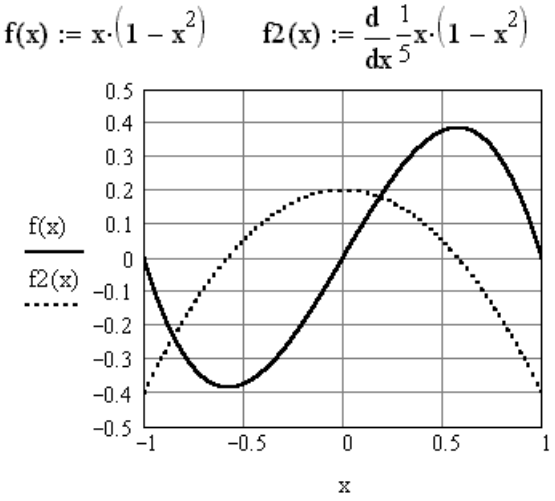


Рис. 6.7 — Вывод нескольких графиков в одних осях

График в **MathCad** допускает гибкую настройку. Двойной клик в поле шаблона вызывает на экран диалоговое окно настройки с четырьмя вкладками, изображение двух из них показано на рисунках 6.8 и 6.9. Назначение большинства элементов управления, размещенных на вкладках окна, понятно из соответствующих подписей.

На вкладке **X-Y Axes** (рисунок 1.8) устанавливаются параметры осей графика и размерной сетки в его поле. При построении амплитудно-частотной характеристики (пункты 4, 5 задания на контрольную работу) необходимо установить логарифмический масштаб оси **X**.

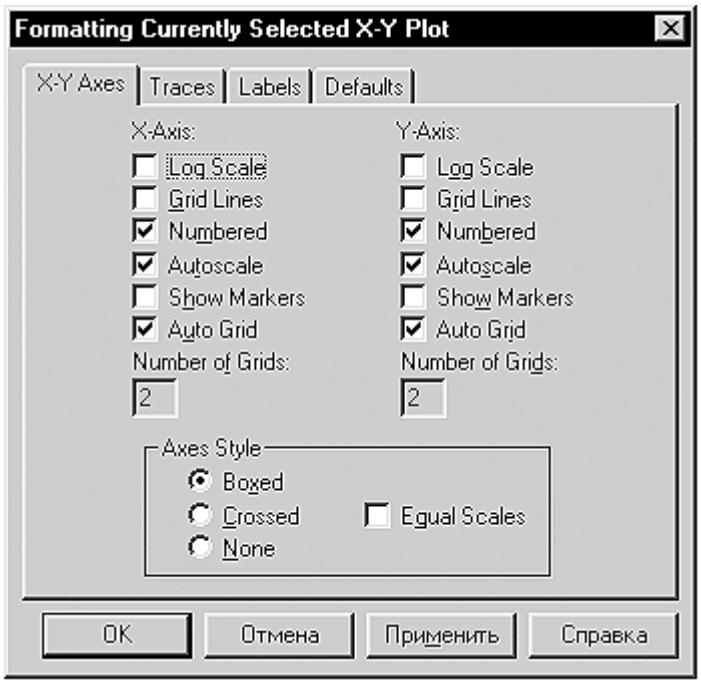


Рис. 6.8 — Вкладка форматирования осей графика

Вкладка **Traces** (рисунок 6.9) позволяет настроить параметры линий, отображающих графики. Здесь можно выбрать тип графика (**Type**), цвет отображения графика (**Color**), вид линии (**Line**), вид символа, помечающего положение узловых точек (**Symbol**).

Пометку узловых точек целесообразно делать лишь в том случае, когда они в достаточной мере удалены друг от друга. При графическом отображении последовательностей можно

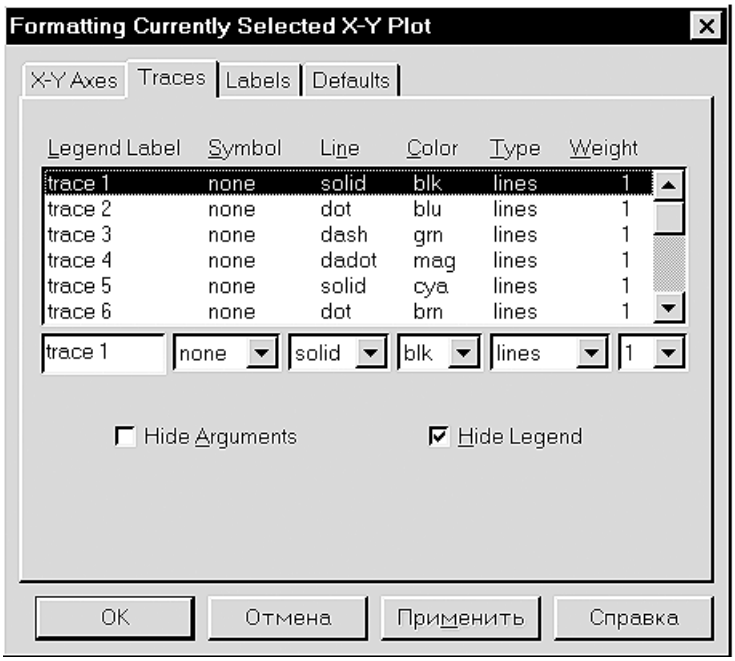


Рис. 6.9 — Вкладка форматирования линии графика

отказаться от соединения соседних узлов линиями, выбрав соответствующий тип графика.

На рисунке 6.10 приведены два разнотипных графика:

- график последовательности, сформированной ранжированной переменной **x** и функцией **y(x).** Тип графика — **points**, отметка узловых точек — **o's**;

- график непрерывной функции **y1(t)**. Тип графика — **line**, отметка узловых точек — **none** (отсутствует).

Абсциссы графиков представлены разнотипными переменными — дискретным **x** и непрерывным **t**. По этой причине обе переменные занесены в список. При построении графика **y1(t)** по непрерывной переменной **t** производится автоматический выбор шага. Шаг последовательности определен изменением ранжированных переменных.

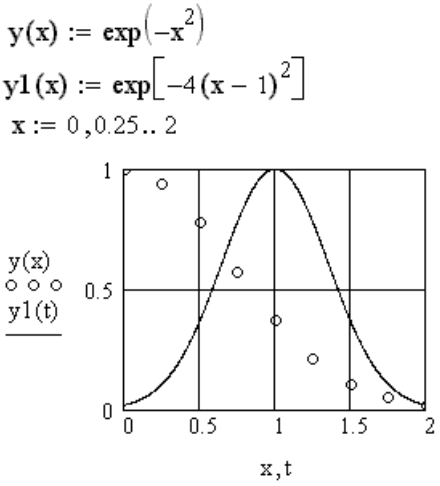


Рис. 6.10 — Воспроизведение разнотипных графиков

7 ОПЕРАЦИИ МАТРИЧНОЙ АЛГЕБРЫ И ИХ РЕАЛИЗАЦИЯ В СРЕДЕ MATHCAD

Напомним некоторые сведения из курса линейной алгебры, полезные для дальнейшего изложения.

7.1 Матрицы и вектора

Матрицей **A** называют прямоугольную таблицу элементов, каждый из которых снабжен парой целочисленных индексов $\mathbf{a_{ij}}$ (рис. 7.1). Матрица имеет строки и столбцы, первый индекс **i** указывает номер строки, второй **j** — номер столбца. Заметим, что в среде **MathCad** по умолчанию нижняя граница индексов — 0, а не 1, как это принято в математической литературе.

$$A:\begin{pmatrix} a_{00} & a_{01} & \bullet & a_{0n} \\ a_{10} & a_{11} & \bullet & a_{1n} \\ \bullet & \bullet & \bullet & \bullet \\ a_{m0} & a_{m1} & \bullet & a_{mn} \end{pmatrix}$$

Рис. 7.1 — Матрица

Матрицы одинаковых размеров можно складывать и вычитать, эти операции производятся покомпонентно, матрица-результат получается того же размера, что и матрицы-операнды. Умножение матрицы на число (скаляр) также выполняется покомпонентно и не изменяет размера исходной матрицы — сомножителя.

Умножение матрицы на матрицу реализуется по известному правилу «строка на столбец». Если $\mathbf{C}=\mathbf{A}\cdot\mathbf{B}$, где **A**, **B** и **C** — матрицы, то элементы результирующей матрицы **C** связаны элементами матриц-сомножителей так:

$$c_{ij} = \sum_{k=0}^{m-1} a_{ik} \cdot b_{kj},$$

где **m** — число столбцов в левой матрице-сомножителе, равное количеству строк в правой матрице-сомножителе. Для существования произведения матриц должно соблюдаться условие: ***число столбцов левого сомножителя должно совпадать с числом строк правого сомножителя.*** Нетрудно заметить, что результирующая матрица **C** содержит одинаковое число строк с матрицей **A** и одинаковое число столбцов с матрицей **B**. Произведение мат-

риц в общем случае некоммутативно, т.е. $\mathbf{A}\cdot\mathbf{B}\neq\mathbf{B}\cdot\mathbf{A}$, и, более того, из факта существования **AB** не следует существование **BA.**

$$X:\begin{pmatrix} x_0 \\ x_1 \\ \bullet \\ x_m \end{pmatrix}$$

Рис. 7.2 — Вектор

Частный вид матрицы — ***вектор***. Это матрица, состоящая из единственного столбца. Компоненты вектора содержат только один индекс (рисунок 7.2). Вектора и матрицы используются для компактной записи систем линейных алгебраических и дифференциальных уравнений.

7.2 Квадратные матрицы

Важную роль в приложениях играют квадратные матрицы, количество строк и столбцов в которых одинаково. Квадратные матрицы одного и того же размера всегда можно перемножать между собой (разумеется, в общем случае свойство некоммутативности произведения сохраняется). Через многократное перемножение квадратной матрицы **A** саму на себя вводится $\mathbf{A}^{\mathbf{n}}$ — целая положительная степень матрицы.

Особым видом квадратной матрицы является единичная матрица, содержащая единицы на главной диагонали и нули вне ее. Для обозначения единичных матриц в математической литературе обычно используют символы **E** или **I**. Единичная матрица в матричной алгебре играет ту же роль, что число 1 в обычной «числовой» алгебре. Допустимое (в указанном выше смысле) умножение любой матрицы справа или слева на единичную не изменяет ее: $\mathbf{E}\cdot\mathbf{A}=\mathbf{A}$, $\mathbf{B}\cdot\mathbf{E}=\mathbf{B}$.

Через единичную матрицу вводится определение отрицательной степени квадратной матрицы **A**. По определению $\mathbf{A}^{-1}$ — это такая квадратная матрица, для которой справедливо $\mathbf{A}^{-1}\cdot\mathbf{A}=\mathbf{A}\cdot\mathbf{A}^{-1}=\mathbf{E}$. Матрицу $\mathbf{A}^{-1}$ называют ***обратной*** по отношению к матрице **A**. Обратная матрица существует не для всякой квадратной матрицы. Те матрицы, для которых не существует обратной, называют ***вырожденными***. Признаком вырожденности матрицы является равенство нулю ее ***определителя*** **|A|=0.** Если существует $\mathbf{A}^{-1}$, то ее многократным умножением саму на себя можно получить любую отрицательную целую степень матрицы $\mathbf{A}^{-n}$. Из определения обратной матрицы следует также, что $\mathbf{A}^{0}=\mathbf{E}$.

7.2.1 Матричные функции от квадратных матриц

Существование целочисленных степеней квадратной матрицы позволяет ввести матричные степенные ряды с числовыми коэффициентами. Из математики известно, что некоторые функции одной переменной **f(x)** представляются степенными рядами, в общем случае бесконечными, по неотрицательным степеням **x**:

$$f(x) = \sum_{k=0}^{\infty} \alpha_k \cdot x^k ,$$

где α_k — постоянные коэффициенты.

Если в таком ряду формально заменить скалярную переменную x на квадратную матрицу A, то получим обобщение функции f(x) в матричную функцию матричного аргумента:

$$f(A) = \sum_{k=0}^{\infty} \alpha_k \cdot A^k .$$

Здесь мы не будем касаться принципиального вопроса сходимости матричного ряда, поскольку это специальный раздел теории матриц, изложение которого выходит за пределы настоящего курса.

В приложении к решению систем дифференциальных уравнений с постоянными коэффициентами важную роль играет матричная экспонента:

$$e^{At} = \sum_{k=0}^{\infty} \frac{A^k \cdot t^k}{k!},$$

где A — квадратная матрица, t — скалярная переменная.

Через матричную экспоненту выражается аналитическое решение системы однородных дифференциальных уравнений с постоянными коэффициентами.

7.3 Системы линейных уравнений и матричные операции

Системы линейных уравнений и матрицы весьма тесно связаны. Матричная алгебра возникла в связи развитием методов решения линейных уравнений (алгебраических, дифференциальных) и отражает основные законы преобразования систем таких

уравнений. Система линейных алгебраических уравнений в матричной форме выглядит так:

$$\mathbf{Ax = b,}$$

где **A** — матрица с **m** строками и **n** столбцами с известными элементами;

x — вектор искомых решений размера **n**;

b — вектор с известными элементами размера **m**.

Если матрица A квадратная **(n = m)** и невырождена, то $\mathbf{x = A^{-1}b}$ — единственное решение. Существуют задачи, в которых количество уравнений меньше числа неизвестных **(n > m)** и необходимо выразить **m** переменных (обозначим их вектором **x1**) через оставшиеся **n – m** переменных (обозначим их вектором **x2**). Это возможно, если матрица **A** имеет **m** независимых строк, иными словами ни одно из уравнений не является следствием других. Систему уравнений можно преобразовать к виду

$$\mathbf{A1x1 = b – A2x2,}$$

где **A1** — квадратная матрица **m×m**, состоящая из коэффициентов при компонентах вектора **x1**,

A2 — матрица, содержащая **m** строк и **n – m** столбцов, образованная коэффициентами при компонентах вектора **x2.**

Тогда решением задачи будет $\mathbf{x1 = A1^{-1}(b–A2x2).}$ Реализация таких вычислений сводится к компоновке матриц A1 и A2 из столбцов исходной матрицы A и взятию обратной матрицы A1, выполнение этих действий в среде **MathCad** не составляет проблем.

7.4 Матричные операции MathCad

В среде **MathCad** операции над матрицами производятся в соответствии с правилами математики, интерпретирующая система блокирует некорректную матричную операцию. При этом некорректный, нереализуемый оператор выделяется измененным цветом шрифта и комментарием в виде всплывающей подсказки. Таким образом, программист может контролировать принципиальную осуществимость матричных операций, связанную, в частности, и с размерностью операндов. Комплекс **MathCad** содержит достаточно представительную библиотеку встроенных функций, в том числе и функций, работающих с матрицами. Ни-

же рассмотрены некоторые функции, осуществляющие матричные операции, используемые при выполнении контрольных работ.

7.4.1 Ввод и редактирование формата матриц с помощью шаблона

Интерфейс процессора MathCad позволяет вводить матрицы с помощью шаблона, задаваемого в диалоговом окне. Диалоговое окно вызывается командой главного меню **Insert\ Matrix…,** либо комбинацией клавиш-акселераторов **Ctrl+M**, либо воздействием («клик») на пиктограмму [⋮⋮] палитры матричных операций. Вид диалогового окна показан на рисунке 7.3. На этом же рисунке показан шаблон, возникающий в точке нахождения курсора после нажатия кнопки «ОК» в поле диалогового окна. В данном примере создан шаблон для занесения матрицы с тремя строками (**Rows**) и четырьмя столбцами (**Columns**). Заполнив знакоместа (выделены черными прямоугольниками) числовыми константами, либо формульными выражениями, получаем готовый объект — матрицу, с которым далее можно производить все допустимые действия.

Рис. 7.3 — Диалоговое окно и шаблон ввода матрицы

С помощью этого диалога можно редактировать формат уже созданных матриц, добавляя/удаляя строки и столбцы. Эти действия вызываются кнопками **Insert** и **Delete**. Для редактирования формата матрицы необходимо:

а) выделить некоторый элемент матрицы, подведя под него курсор;

б) вызвать окно диалога;

в) задать необходимые величины в полях **Rows** и **Columns**;

г) нажать кнопку **Insert** или **Delete**.

Пример редактирования существующих матриц с помощью диалога показан на рисунке 7.4.

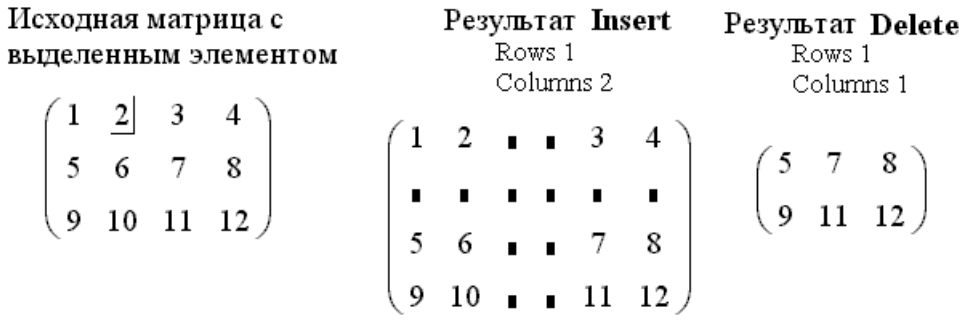


Рис. 7.4 — Редактирование формата матриц с помощью окна диалога

Как видно из приведенного примера, вставка (Insert) заданного количества строк и столбцов происходит *после* строки и столбца с отмеченным элементом. Операция удаления (Delete) убирает заданное число строк и столбцов после строки и столбца с отмеченным элементом, *включая* строку и столбец с выделенным элементом. Если необходимо убрать\добавить только строки, то в поле Columns надо поставить 0, если необходимо убрать\добавить только столбцы, то надо задать Rows=0.

7.4.2 Функции компоновки и декомпозиции матриц

В составе библиотеки встроенных функций MathCad имеются функции, позволяющие производить, формировать матрицу из блоков-подматриц и выделять из заданной матрицы подматрицы нужного размера. Компоновку матрицы из блоков осуществляют функции **augment, stack.** Выделение подматрицы выполняется функцией **submatrix** и оператором выделения столбца матрицы $\mathbf{M}^{\langle\rangle}$.

Функция **augment** объединяет матрицы-аргументы слева направо «столбец к столбцу», как показано на рисунке 7.5. Функция **augment** может вызываться с произвольным количеством аргументов, необходимым условием ее работы является равенство числа строк у всех матриц-аргументов.

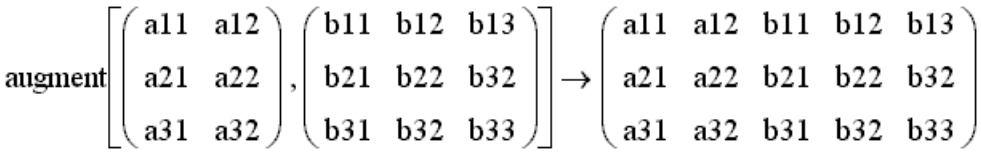


Рис. 7.5 — Действие функции **augment**

Функция **stack** объединяет матрицы-аргументы сверху вниз «строка к строке», как показано на рисунке 7.6. Функция **stack** также может вызываться с произвольным количеством аргументов, а необходимым условием ее работы является равенство числа столбцов у всех матриц-аргументов.

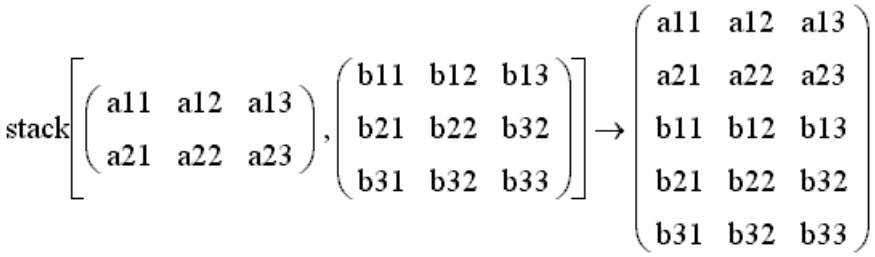


Рис. 7.6 — Действие функции **stack**

Функция выделения подматрицы **submatrix** имеет следующий синтаксис вызова:

submatrix (A, r_b, r_e, c_b, c_e),

где **A** — исходная матрица, из которой извлекается фрагмент;

r_b — номер начальной строки в матрице A, включаемой в выделяемую подматрицу;

r_e — номер конечной строки в матрице A, включаемой в выделяемую подматрицу;

c_b — номер начального столбца в матрице A, включаемого в выделяемую подматрицу;

c_e — номер конечного столбца в матрице A, включаемого в выделяемую подматрицу.

Функция **submatrix (A, r_b, r_e, c_b, c_e)** работоспособна при выполнении условий $\mathbf{0 \ge r_e \ge r_b \ge m-1}$ **и** $\mathbf{0 \ge c_e \ge c_b \ge n-1}$, где **m** и **n** — соответственно число строк и столбцов в матрице **A**. (*Напомним, что нумерация строк и столбцов в MathCad по умолчанию начинается с* 0.) Пример действия функции показан на рисунке 7.7.

$$\text{submatrix}\left[\begin{pmatrix} a11 & a12 & a13 & a14 \\ a21 & a22 & a23 & a24 \\ a31 & a32 & a33 & a34 \\ a41 & a42 & a43 & a44 \\ a51 & b52 & b53 & a54 \end{pmatrix}, 1, 3, 1, 2\right] \rightarrow \begin{pmatrix} a22 & a23 \\ a32 & a33 \\ a42 & a43 \end{pmatrix}$$

Рис. 7.7 — Действие функции **submatrix**

С помощью функции **submatrix** можно выделить любой прямоугольный фрагмент из матрицы. Для частного случая декомпозиции матрицы — выделения столбца с заданным номером — в наборе инструментов векторно-матричных операций MathCad имеется специальный оператор $\mathbf{M}^{\langle\rangle}$, шаблон которого вызывается воздействием на элемент управления с пиктограммой. Пример применения оператора $\mathbf{M}^{\langle\rangle}$ приведен на рисунке 7.8.

$$\begin{pmatrix} a11 & a12 & a13 \\ a21 & a22 & a23 \\ a31 & a32 & a33 \\ a41 & a42 & a43 \end{pmatrix}^{\langle 0 \rangle} \rightarrow \begin{pmatrix} a11 \\ a21 \\ a31 \\ a41 \end{pmatrix} \qquad \begin{pmatrix} a11 & a12 & a13 \\ a21 & a22 & a23 \\ a31 & a32 & a33 \\ a41 & a42 & a43 \end{pmatrix}^{\langle 1 \rangle} \rightarrow \begin{pmatrix} a12 \\ a22 \\ a32 \\ a42 \end{pmatrix}$$

Рис. 7.8 — Действие оператора выделения столбца $\mathbf{M}^{\langle\rangle}$

7.4.3 Создание единичной и диагональной матриц

Для создания единичной матрицы заданного размера в **MathCad** можно использовать функцию **identity(n)**. Эта функция возвращает квадратную единичную матрицу размера **n×n** (рисунок 7.9), размещая ее в точке вызова функции.

$$\text{identity}(3) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \qquad \text{identity}(2) = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

Рис. 7.9 — Создание единичной матрицы функцией **identity**

Квадратная матрица размера **n×n**, элементы которой отличны от нуля только на главной диагонали (**диагональная матри-**

ца), может быть сформирована из вектора **x** размера **n** с помощью встроенной функции **diag (x)**. Размер матрицы определяется автоматически по размеру вектора, а компоненты вектора встраиваются в диагональ в порядке своего расположения (рис. 7.10).

$$\mathrm{diag}\begin{pmatrix}1\\2\\3\end{pmatrix}=\begin{pmatrix}1&0&0\\0&2&0\\0&0&3\end{pmatrix}\qquad \mathrm{diag}\begin{pmatrix}1\\2\end{pmatrix}=\begin{pmatrix}1&0\\0&2\end{pmatrix}$$

Рис. 7.10 — Результат работы функции **diag**

7.4.4 Формирование матрицы на основе заданной функции

В ряде случаев значения элемента матрицы $\mathbf{a_{ij}}$ можно связать со значениями его индексов некоторой функцией от двух целочисленных аргументов: $\mathbf{a_{ij}} = \mathbf{F}(i,j)$. В этом случае матрицу произвольного размера можно сформировать при помощи встроенной функции **matrix(m,n,F)**, где **m** и **n** — число строк и столбцов в создаваемой матрице, **F** — имя функции двух целочисленных аргументов, используемой для расчета элементов. Функция **F**(i,j) должна быть определена к моменту выполнения **matrix**.

Пример формирования матриц с различными функциями формирования значений элементов приведен на рисунке 7.11.

$$F1(i,j) := (i+1)^j \qquad F2(i,j) := 2^{-|i-j|}$$

$$\mathrm{matrix}(4,5,F1)=\begin{pmatrix}1&1&1&1&1\\1&2&4&8&16\\1&3&9&27&81\\1&4&16&64&256\end{pmatrix}\qquad \mathrm{matrix}(4,3,F2)=\begin{pmatrix}1&0.5&0.25\\0.5&1&0.5\\0.25&0.5&1\\0.125&0.25&0.5\end{pmatrix}$$

Рис. 7.11 — Формирование матриц функцией **matrix**

7.4.5 Поэлементное формирование матриц

Матрицу произвольного размера можно сформировать, задавая значение каждому элементу с помощью оператора назначения (рисунок 7.12). Перебор множества индексов может быть

произведен с помощью ранжированных переменных (создание матрицы A1), либо с использованием операторов цикла в программном блоке (создание матрицы A2). Как видно из приведенного на рисунке 7.12 примера, результат абсолютно одинаков.

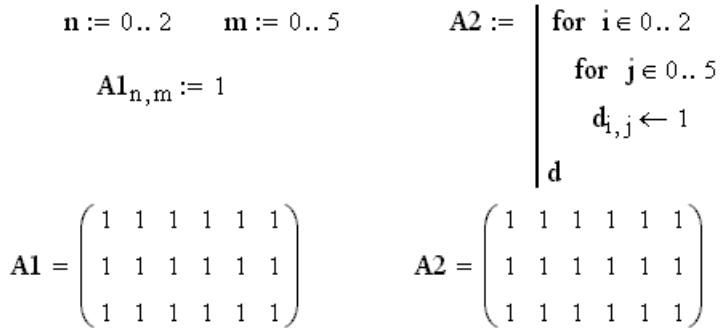


Рис. 7.12 — Поэлементное формирование матриц

7.4.6 Целочисленные степени квадратных матриц

В среде **MathCad** целочисленная степень матрицы (в том числе отрицательная) задается точно так же, как и степень обычных числовых переменных (рисунок 7.13). В том случае, когда матрица не имеет обратной, попытка вычислить ее какую-либо отрицательную степень блокируется с выдачей соответствующего сообщения. Нулевая и положительная степень квадратной матрицы вычисляются всегда.

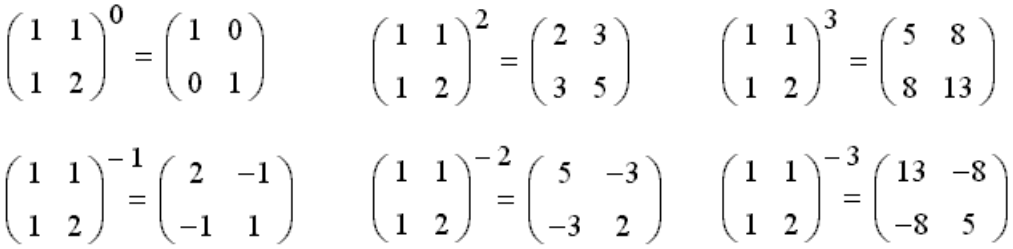


Рис. 7.13 — Целочисленные степени квадратной матрицы

7.4.7 Функции определения размеров матриц

Возможны ситуации, когда для вычислительного процесса необходимы сведения о размерах матриц, которые становятся известны только после их формирования в ходе вычислений. Для получения такой информации используются функции определения количества строк и столбцов в матрицах **rows** и **cols.** На ри-

сунке 7.14 приведен пример довольно сложного формирования матрицы **C** из матриц **A** и **B** с использованием функций **rows** и **cols** для определения размеров промежуточных матриц.

$$A := \begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \end{pmatrix} \quad B := \begin{pmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \end{pmatrix} \quad C := \text{augment}(A \cdot B, \text{identity}(\text{rows}(A \cdot B)))$$

$$C := \text{stack}\left(C, \text{identity}(\text{cols}(C))^{\langle 0 \rangle^T}\right)$$

$$C = \begin{pmatrix} 3 & 6 & 1 & 0 \\ 6 & 14 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

Рис. 7.14 — Использование функций **rows** и **cols**

Применение функций **rows** и **cols** повышает универсальность выражений, использующих матричные операции, они становятся менее зависимыми от конкретных размеров исходных матриц. Так, например, в рассмотренном выше примере при произвольных размерах матриц **A** и **B**, допускающих их перемножение, вычислительные операторы выполнятся корректно.

7.5 Оператор | x |

Вызов шаблона оператора | **x** | может производиться из нескольких инструментальных панелей. Этот оператор многофункционален, результат его действия зависит от типа аргумента **x**. В программировании такое поведение функций называют полиморфизмом. Полиморфизм оператора | x | отражен в таблице 7.1.

Таблица 7.1

Тип аргумента	**Результат**
скаляр	абсолютное значение числа
комплексное число a + ib	модуль комплексного числа $\sqrt{a^2 + b^2}$
вектор $\begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{pmatrix}$	модуль вектора: $\sqrt{\sum_{k=1}^{n} x_k^2}$
квадратная матрица	определитель

Расчет определителя квадратной матрицы в **MathCad** производится оператором **|A|.** Попытка применить этот оператор к неквадратной матрице с количеством столбцов более одного вызывает ошибку.

7.6 Оператор векторизации функций

В наборе инструментов матричных операций имеется оператор векторизации функций. Пусть **f(x)** определена как функция скалярного аргумента и задана **M** — матрица произвольных размеров с элементами $\mathbf{M_{ij}}$. Операция векторизации функции **f(x)** вызывает появление матрицы с элементами $\mathbf{f(M_{ij})}$, т.е. происходит поэлементное функциональное преобразование матрицы — аргумента в матрицу — результат того же размера. Например:

$$f(x) := x^2$$

$$\overrightarrow{f\left(\begin{pmatrix}1&5&4\\2&6&3\\3&7&2\\4&8&1\end{pmatrix}\right)} = \begin{pmatrix}1&25&16\\4&36&9\\9&49&4\\16&64&1\end{pmatrix} \qquad \overrightarrow{\sin\left(\begin{pmatrix}\frac{\pi}{3}&\frac{-\pi}{4}\\\frac{\pi}{6}&\frac{\pi}{2}\end{pmatrix}\right)} = \begin{pmatrix}0.866&-0.707\\0.5&1\end{pmatrix}$$

Векторизация меняет смысл оператора **| x |**, рассмотренного выше:

$$\overrightarrow{\left|\begin{pmatrix}3+4i\\-3\\6-8i\end{pmatrix}\right|} = \begin{pmatrix}5\\3\\10\end{pmatrix} \qquad \overrightarrow{\left|\begin{pmatrix}3+4i&-12\\-3&7\\6&4-3i\end{pmatrix}\right|} = \begin{pmatrix}5&12\\3&7\\6&5\end{pmatrix} \qquad \overrightarrow{\left|\begin{pmatrix}1&-1\\-1&2\end{pmatrix}\right|} = \begin{pmatrix}1&1\\1&2\end{pmatrix}$$

Как видно из приведенных примеров, векторизованный оператор **| x |** выполняет замену элементов матрицы их абсолютными величинами, причем для комплексных чисел берется модуль. Матрица-аргумент может быть произвольного размера.

Операции векторизации нет в математике, это специфическая функция MathCad. Заметим, что ***векторизация функций не имеет ничего общего с матричными функциями от квадратных матриц, рассмотренными в п. 2.3.***

7.7 Интерполяция и регрессия

Этот раздел посвящен методам обработки выборочных данных — интерполяции, экстраполяции и регрессии. Будем считать, что основным объектом исследования будет выборка экспериментальных данных, которые, представляются в виде массива, состоящего из пар чисел **(xi,yi)**. Возникает задача аппроксимации дискретной зависимости **y(xi)** непрерывной функцией **f(x).** Функция **f(x)**, в зависимости от специфики задачи, может отвечать различным требованиям:

- **f(x)** должна проходить через точки (**xi,yi),** т.е**.** **f(xi)=yi, i=1..n.** В этом случае говорят об интерполяции данных функцией **f(x)** во внутренних точках между xi или экстраполяции за пределами интервала, содержащего все **xi**;
- **f(x)** должна некоторым образом (например, в виде определенной аналитической зависимости) приближать **y(xi),** не обязательно проходя через точки **(xi,yi)**. Такова постановка задачи регрессии.

Различные виды построения аппроксимирующей зависимости **f(x)** иллюстрирует рис. 7.15.

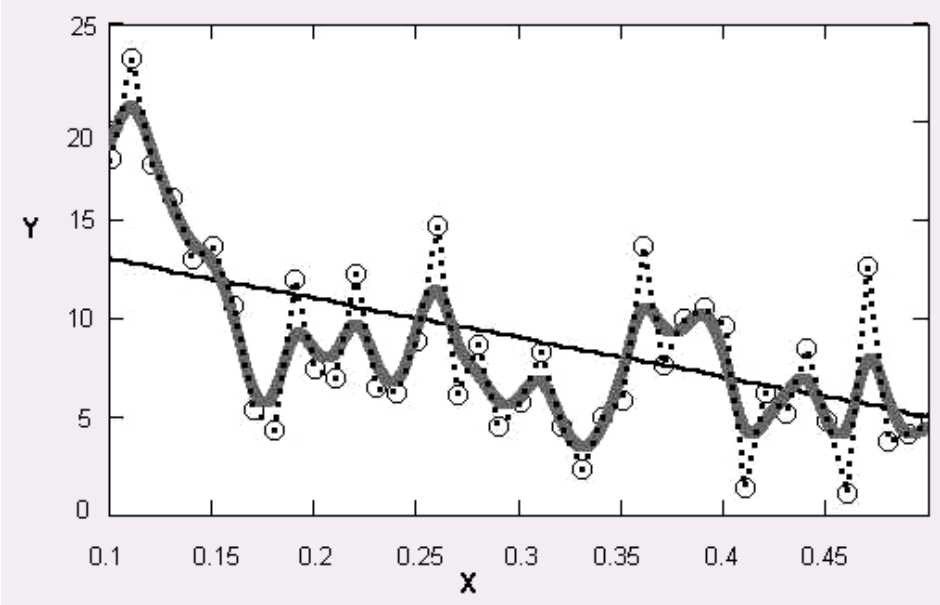


Рис. 7.15 — Разные задачи аппроксимации данных

На нем исходные данные обозначены кружками, интерполяция отрезками прямых линий — пунктиром, линейная регрессия — наклонной прямой линией, а фильтрация — жирной гладкой кри-

вой. Эти зависимости приведены в качестве примера и отражают лишь малую часть возможностей Mathcad по обработке данных. Вообще говоря, в Mathcad имеется целый арсенал встроенных функций, позволяющий осуществлять самую различную регрессию, интерполяцию и экстраполяцию.

7.8 Интерполяция

Для построения интерполяции-экстраполяции в Mathcad имеется несколько встроенных функций, позволяющих «соединить» точки выборки данных (x_i,y_i) кривой разной степени гладкости. По определению интерполяция означает построение функции A(X), аппроксимирующей зависимость y(x) в промежуточных точках (между x_i). Поэтому интерполяцию еще по-другому называют аппроксимацией. В точках x_i значения интерполяционной функции должны совпадать с исходными данными, т.е. $A(x_i)=y(x_i)$.

7.9 Линейная интерполяция

Самый простой вид интерполяции — линейная, которая представляет искомую зависимость A(X) в виде ломаной линии. Интерполирующая функция A(x) состоит из отрезков прямых, соединяющих точки (рис. 7.16).

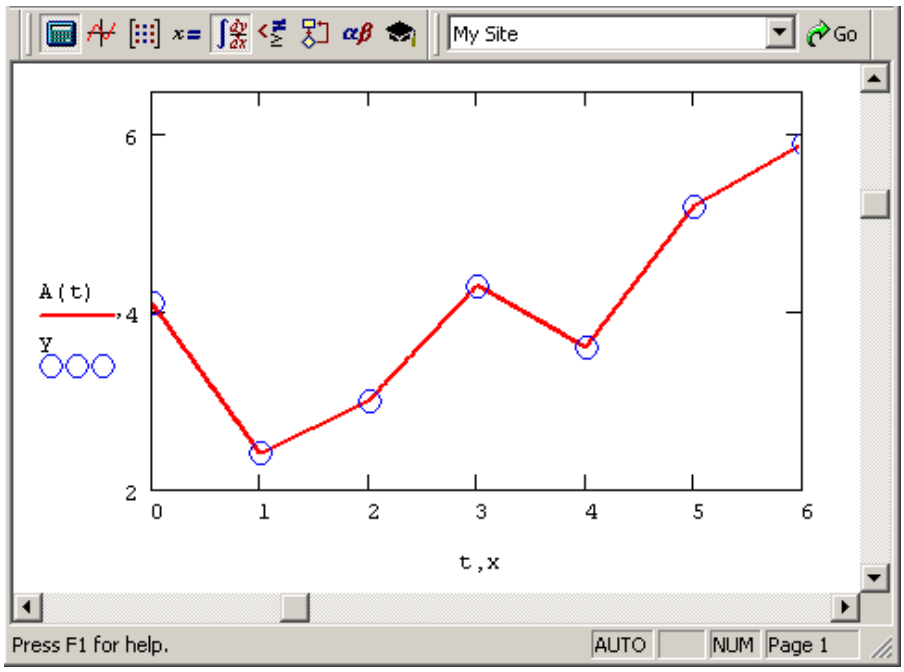


Рис. 7.16 — Линейная интерполяция

Для построения линейной интерполяции служит встроенная функция **linterp**:

- **linterp(x,y,t)** — функция, аппроксимирующая данные векторов **x** и **y** кусочно-линейной зависимостью:
- **x** — вектор действительных данных аргумента;
- **y** — вектор действительных данных значений того же размера;
- **t** — значение аргумента, при котором вычисляется интерполирующая функция.

*ВНИМАНИЕ! Элементы вектора **x** должны быть определены в порядке возрастания, т.е.* $X_1 < X_2 < X_3 < ... < X_N$.

Пример линейной интерполяции.

Листинг 1. Линейная интерполяция

```
x := (0.0  1.0  2.0  3.0  4.0  5.0  6.0)^T

y := (4.1  2.4  3.0  4.3  3.6  5.2  5.9)^T

A(t) := linterp(x, y, t)
```

Как видно из листинга, чтобы осуществить линейную интерполяцию, надо выполнить следующие действия:

Ввести векторы данных **x** и **y** (первые две строки листинга).

Определить функцию **linterp(x,y,t).**

Вычислить значения этой функции в требуемых точках, например,

linterp(x,y,2.4)=3.52, или **linterp(x,y,6)=5.9**, или постройте ее График, как показано на рис. 13.2.

*ПРИМЕЧАНИЕ. Функция **A**(**t**) на графике имеет аргумент t, а не x. Это означает, что функция **A**(**t**) вычисляется не только при значениях аргумента* (*т.е. в семи точках*)*, а при гораздо большем числе аргументов в интервале* (0**,**6)*, что автоматически обеспечивает Mathcad. Просто в данном случае эти различия незаметны, т.к. при обычном построении графика функции **A**(**x**) от векторного аргумента **x*** (*рис. 7.17*) *Mathcad по умолчанию соединяет точки графика прямыми линиями* (*т.е. скрытым образом осуществляет их линейную интерполяцию*).

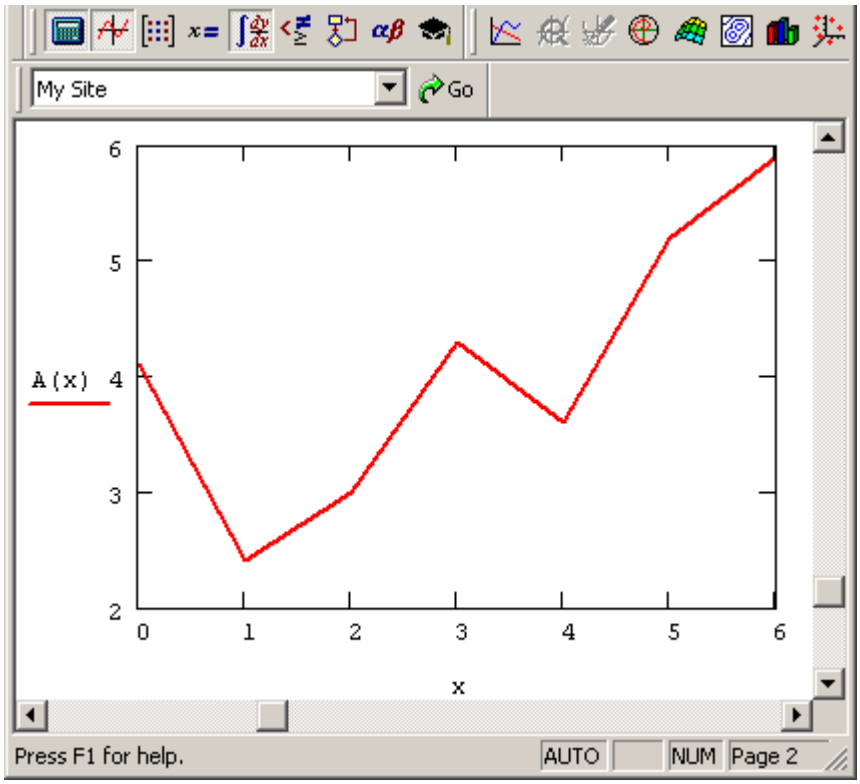


Рис. 7.17 — Обычное построение графика функции от векторной переменной x

7.10 Кубическая сплайн-интерполяция

В большинстве практических приложений желательно соединить экспериментальные точки не ломаной линией, а гладкой кривой. Лучше всего для этих целей подходит интерполяция кубическими сплайнами, т.е. отрезками кубических парабол:

- **`interp(s,x,y,t)`** — функция, аппроксимирующая данные векторов **`x`** и **`y`** кубическими сплайнами:
- **`s`** — вектор вторых производных, созданный одной из сопутствующих функций **`cspline`**, **`pspline`** или **`lspline;`**
- **`x`** — вектор действительных данных аргумента, элементы которого расположены в порядке возрастания;
- **`y`** — вектор действительных данных значений того же размера;
- **`t`** — значение аргумента, при котором вычисляется интерполирующая функция.

Сплайн-интерполяция в Mathcad реализована чуть сложнее линейной. Перед применением функции **`interp`** необходимо

предварительно определить первый из ее аргументов — векторную переменную **s**. Делается это при помощи одной из трех встроенных функций тех же аргументов **(x, y)**:

- **ispline(x,y)** — вектор значений коэффициентов линейного сплайна;
- **pspiine(x,y)** — вектор значений коэффициентов квадратичного сплайна;
- **cspline(x,y)** — вектор значений коэффициентов кубического сплайна:
- **x,y** — векторы данных.

Выбор конкретной функции сплайновых коэффициентов влияет на интерполяцию вблизи конечных точек интервала. Пример сплайн-интерполяции приведен в листинге 2.

Листинг 2. Кубическая сплайн-интерполяция

```
x := (0.0  1.0  2.0  3.0  4.0  5.0  6.0)^T

y := (4.1  2.4  3.0  4.3  3.6  5.2  5.9)^T


s := cspline(x, y)

A(t) := interp(s, x, y, t)
```

Смысл сплайн-интерполяции заключается в том, что в промежутках между точками осуществляется аппроксимация в виде зависимости $\mathbf{A(t)=at^3+bt^2+ ct+d}$. Коэффициенты **a, b, c, d** рассчитываются независимо для каждого промежутка, исходя из значений **y*** в соседних точках. Этот процесс скрыт от пользователя, поскольку смысл задачи интерполяции состоит в выдаче значения **A(t)** в любой точке t (рис. 7.18).

Чтобы подчеркнуть различия, соответствующие разным вспомогательным функциям **cspline, pspline, ispline**, покажем результат действия листинга 2 при замене функции **cspline** в предпоследней строке на линейную **ispline** (рис. 7.19). Как видно, выбор вспомогательных функций существенно влияет на поведение **A(t)** вблизи граничных точек рассматриваемого интервала (0,6) и особенно разительно меняет результат экстраполяции данных за его пределами.

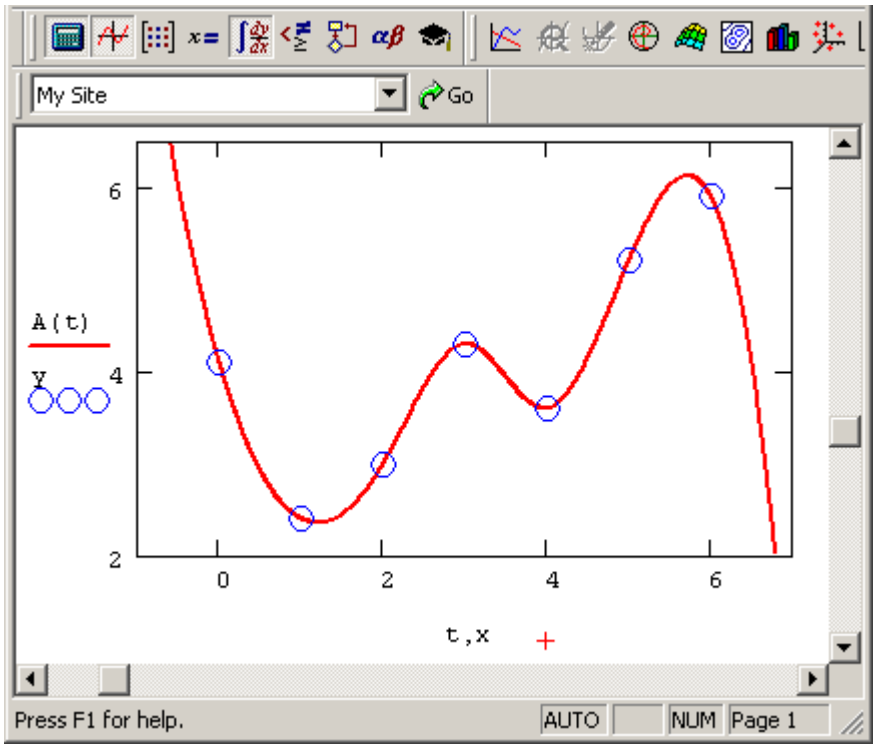


Рис. 7.18 —Сплайн-интерполяция (продолжение листинга 2)

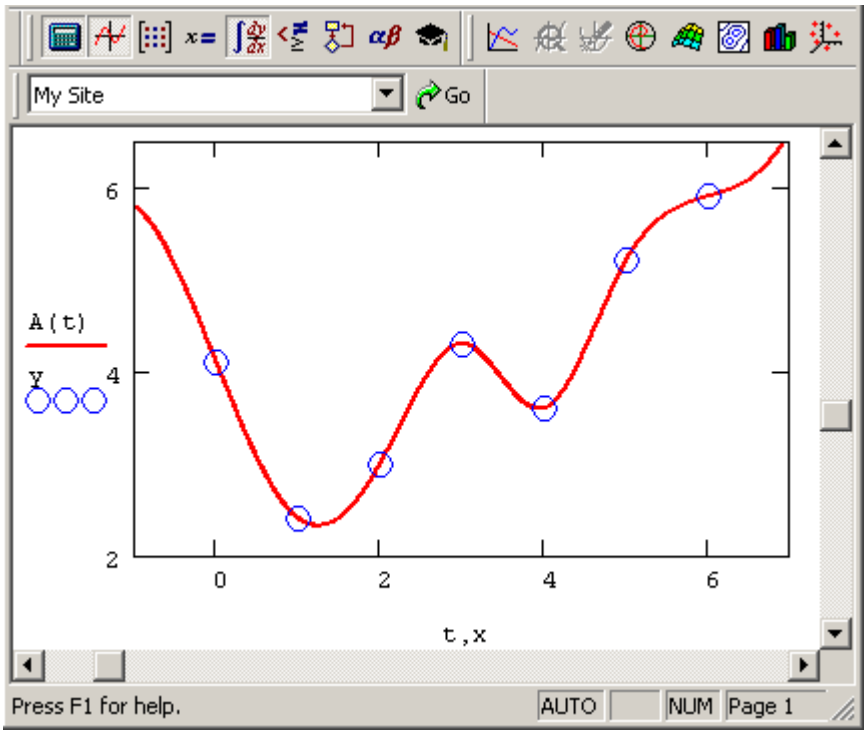


Рис. 7.19 — Сплайн-интерполяция с выбором коэффициентов линейного сплайна lspline

В заключение остановимся на уже упоминавшейся в предыдущем разделе распространенной ошибке при построении графиков интерполирующей функции (см. рис. 7.17). Если на графике, например, являющемся продолжением листинга 2, задать построение функции A(x) вместо A(t), то будет получено просто соединение исходных точек ломаной (рис. 7.20). Так происходит потому, что в промежутках между точками вычисления интерполирующей функции не производятся.

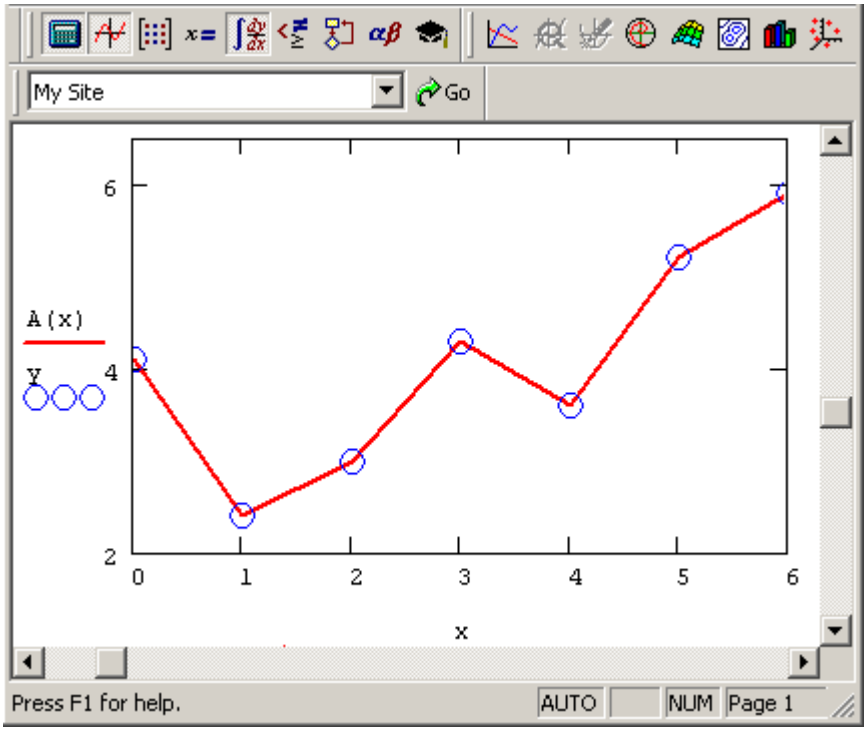


Рис. 7.20 — Ошибочное построение графика сплайн-интерполяции

Полиномиальная сплайн-интерполяция

Более сложный тип интерполяции — так называемая интерполяция B-сплайнами. В отличие от обычной сплайн-интерполяции сшивка элементарных B-сплайнов производится не в точках $\mathbf{x_i}$ а в других точках $\mathbf{u_i}$, координаты которых предлагается ввести пользователю. Сплайны могут быть полиномами 1, 2 или 3 степени (линейные, квадратичные или кубические). Применяется интерполяция B-сплайнами точно так же, как и обычная сплайн-интерполяция, различие состоит только в определении вспомогательной функции коэффициентов сплайна.

- **interp(s,x,y,t)** — функция, аппроксимирующая данные векторов x и y с помощью B-сплайнов;
- **bspiine (x,y,u,n)** — вектор значений коэффициентов B-сплайна;
- s — вектор вторых производных, созданный функцией **bspline**;
- x — вектор действительных данных аргумента, элементы которого расположены в порядке возрастания;
- y — вектор действительных данных значений того же размера;
- t — значение аргумента, при котором вычисляется интерполирующая функция;
- u — вектор значений аргумента, в которых производится сшивка B-сплайнов;
- n — порядок полиномов онлайновой интерполяции (1, 2 или з).

ПРИМЕЧАНИЕ.

Размерность вектора и должна быть на 1, 2 *или* 3 *меньше размерности векторов* ***x*** *и* ***y****. Первый элемент вектора и должен быть меньше или равен первому элементу вектора* ***x****, а последний элемент и — больше или равен последнему элементу x.*

Интерполяция B-сплайнами иллюстрируется листингом 3 и рис. 7.21.

Листинг 3. Интерполяция B-сплайнами

```
x := (0.0  1.0  2.0  3.0  4.0  5.0  6.0)^T

y := (4.1  2.4  3.0  4.3  3.6  5.2  5.9)^T

u := (-0.5  2.2  3.3  4.1  5.5  7)^T

s := bspline(x, y, u, 2)

A(t) := interp(s, x, y, t)
```

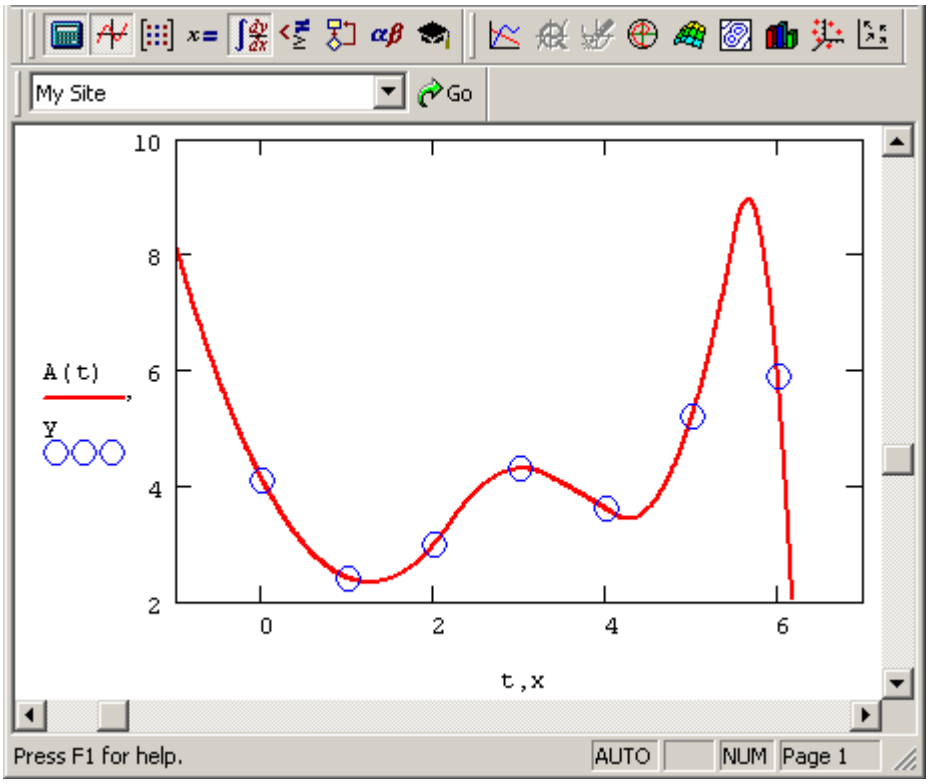


Рис. 7.21 — B-сплайн-интерполяция

7.11 Сплайн-экстраполяция

Все описанные выше функции интерполяции работают также и как функции экстраполяции данных. Для вычисления экстраполяции достаточно просто указать соответствующее значение аргумента, которое лежит за границами рассматриваемого интервала. С этой точки зрения разницы в применении в Mathcad между интерполяцией и экстраполяцией нет.

На практике при построении экстраполяции следует соблюдать известную осторожность, не забывая о том, что ее успех определяется значимостью ближайших к границе интервала точек. Чем дальше от них вы будете пытаться экстраполировать зависимость, заданную экспериментальными точками, тем сомнительнее будет результат. Сказанное иллюстрируется рис. 7.22 и 7.23, на которых изображена линейная (пунктир на обоих графиках) и сплайн — (сплошные кривые) экстраполяция. На рис. 7.22 используется линейная сплайн-экстраполяция при помощи функции **ispline** (см. рис. 7.19 в качестве примера интерполяции), а на рис. 7.23 — функции кубического сплайна **cspline** (что соответствует листингу 2 и рис. 7.18). Видно, что вдали от рассматриваемого

интервала результаты экстраполяции совершенно различны, что, конечно, объясняется тем, что она является ни чем иным как параболической зависимостью.

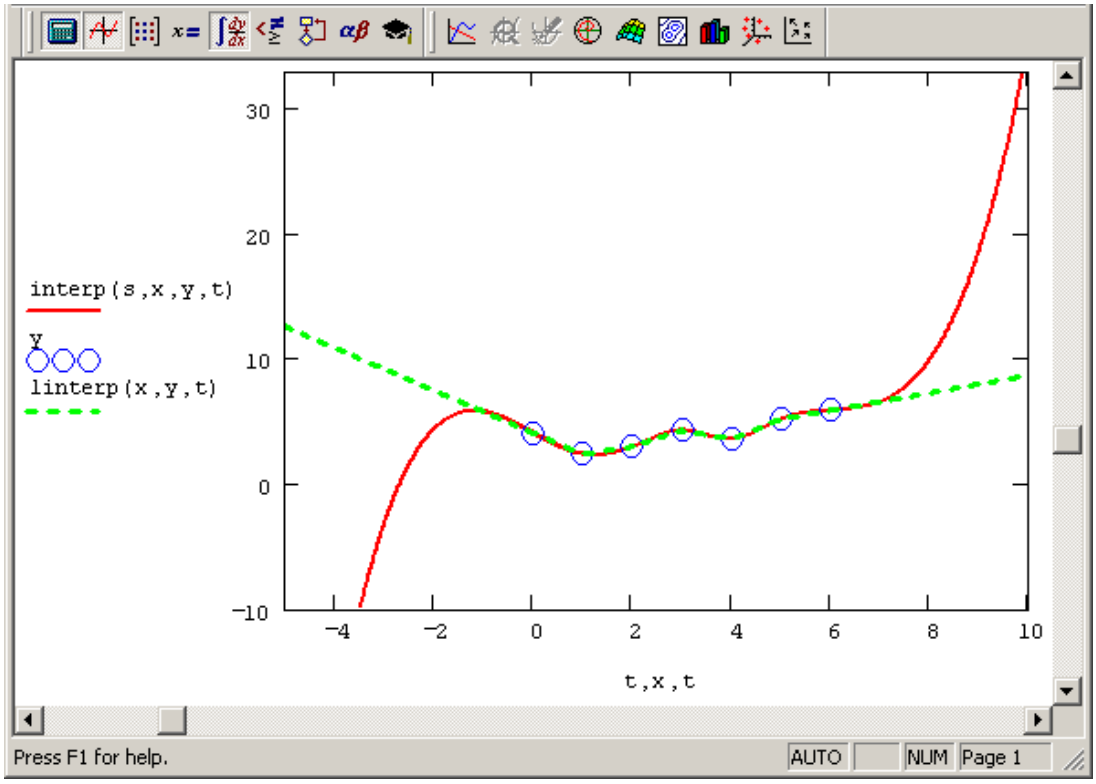


Рис. 7.22 — Линейная сплайн-экстраполяция

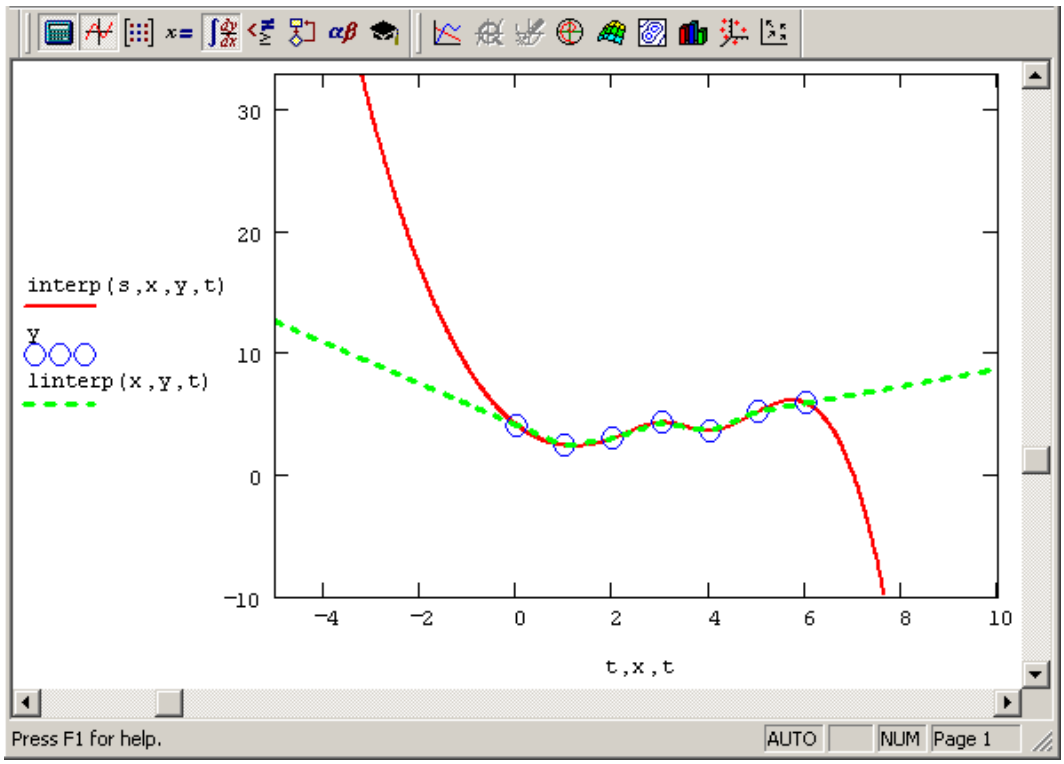


Рис. 7.23 — Квадратичная сплайн-экстраполяция

7.12 Многомерная интерполяция

Двумерная сплайн-интерполяция приводит к построению поверхности **z (x,y),** проходящей через массив точек, описывающий сетку на координатной плоскости **(x,y).** Поверхность создается участками двумерных кубических сплайнов, являющихся функциями **(x,y)** и имеющих непрерывные первые и вторые производные по обеим координатам.

Многомерная интерполяция строится с помощью тех же встроенных функций, что и одномерная (см. разд.), но имеет в качестве аргументов не векторы, а соответствующие матрицы. Существует одно важное ограничение, связанное с возможностью интерполяции только квадратных **NxN** массивов данных:

- **`interp(s,x,z,v)`** — скалярная функция, аппроксимирующая данные выборки двумерного поля по координатам **`x`** и **`y`** кубическими сплайнами:
- **`s`** — вектор вторых производных, созданный одной из сопутствующих функций **`cspline, pspline`** или **`lspline;`**
- **`x`** — матрица размерности **`Nx2`**, определяющая диагональ сетки значений аргумента (элементы обоих столбцов соответствуют меткам **`x`** и **`y`** и расположены в порядке возрастания);
- **`z`** — матрица действительных данных размерности **`NxN`**;
- **`v`** — вектор из двух элементов, содержащий значения аргументов **`x`** и **`y`**, для которых вычисляется интерполяция.

ПРИМЕЧАНИЕ.

Вспомогательные функции построения вторых производных имеют те же матричные аргументы, что и interp: ***lspline*** (***X, Y***)***, pspline*** (***X, Y***)***, cspline*** (***X, Y***).

Пример исходных данных приведен на рис. 7.24 в виде графика линий уровня, программная реализация двумерной интерполяции показана в листинге 6, а ее результат — на рис. 7.25.

Двумерная интерполяция

$$X := \begin{pmatrix} 0 & 0 \\ 1 & 10 \\ 2 & 20 \\ 3 & 30 \\ 4 & 40 \end{pmatrix} \qquad Y := \begin{pmatrix} 1 & 1 & 0 & 1.1 & 1.2 \\ 1 & 2 & 3 & 2.1 & 1.5 \\ 1.3 & 3.3 & 5 & 1.7 & 2 \\ 1.3 & 3 & 3.7 & 2.1 & 2.9 \\ 1.5 & 2 & 2.5 & 2.8 & 4 \end{pmatrix}$$

$S := \text{cspline}(X, Y)$

$$V := \begin{pmatrix} 3.7 \\ 2.2 \end{pmatrix}$$

$\text{interp}(S, X, Y, V) = 1.636$

$$A_{i,j} := \text{interp}\left[S, X, Y, \begin{pmatrix} \frac{i}{k} \cdot 4 \\ \frac{j}{k} \cdot 40 \end{pmatrix}\right]$$

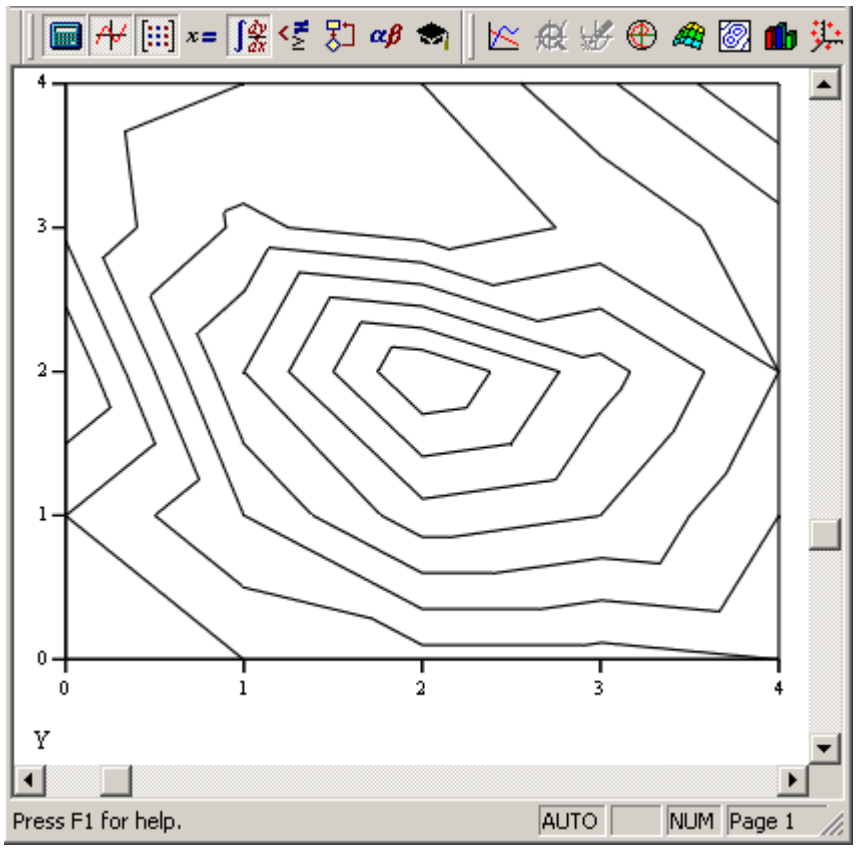


Рис. 7.24 — Исходное двумерное поле данных

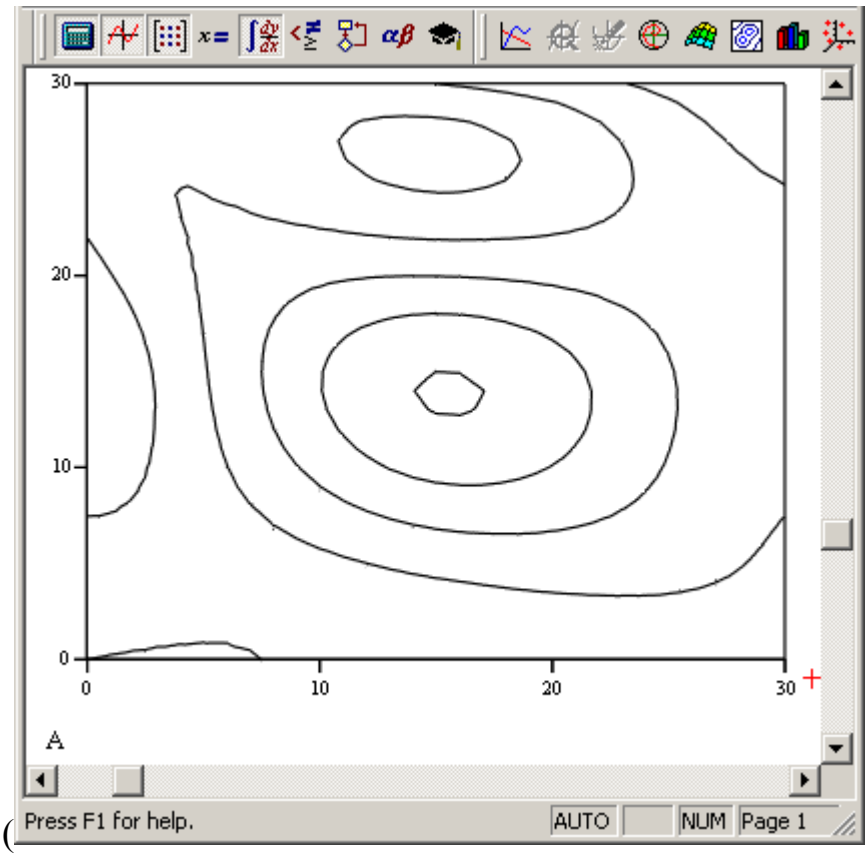


Рис. 7.25 — Результат двумерной интерполяции
(продолжение листинга 6)

7.13 Регрессия

Задачи математической регрессии имеют смысл приближения выборки данных (x_i,y_i) некоторой функцией f(x), определенным образом минимизирующей совокупность ошибок $|f(x_i)-y_i|$. Регрессия сводится к подбору неизвестных коэффициентов, определяющих аналитическую зависимость f (x). В силу производимого действия большинство задач регрессии являются частным случаем более общей проблемы сглаживания данных.

Как правило, регрессия очень эффективна, когда заранее известен (или, по крайней мере, хорошо угадывается) закон распределения данных $\mathbf{(x_i,y_i)}$.

7.13.1 Полиномиальная регрессия

В Mathcad реализована регрессия одним полиномом, отрезками нескольких полиномов, а также двумерная регрессия массива данных.

7.13.2 Регрессия одним полиномом

Полиномиальная регрессия означает приближение данных (x_i,y_i) полиномом **k**-й степени $A(x)=a+bx+cx^2+dx^3+...+hx^k$ (рис. 7.26). При **k=i** полином является прямой линией, при **k=2** — параболой, при **k=3** — кубической параболой и т.д. Как правило, на практике применяются **k<5**.

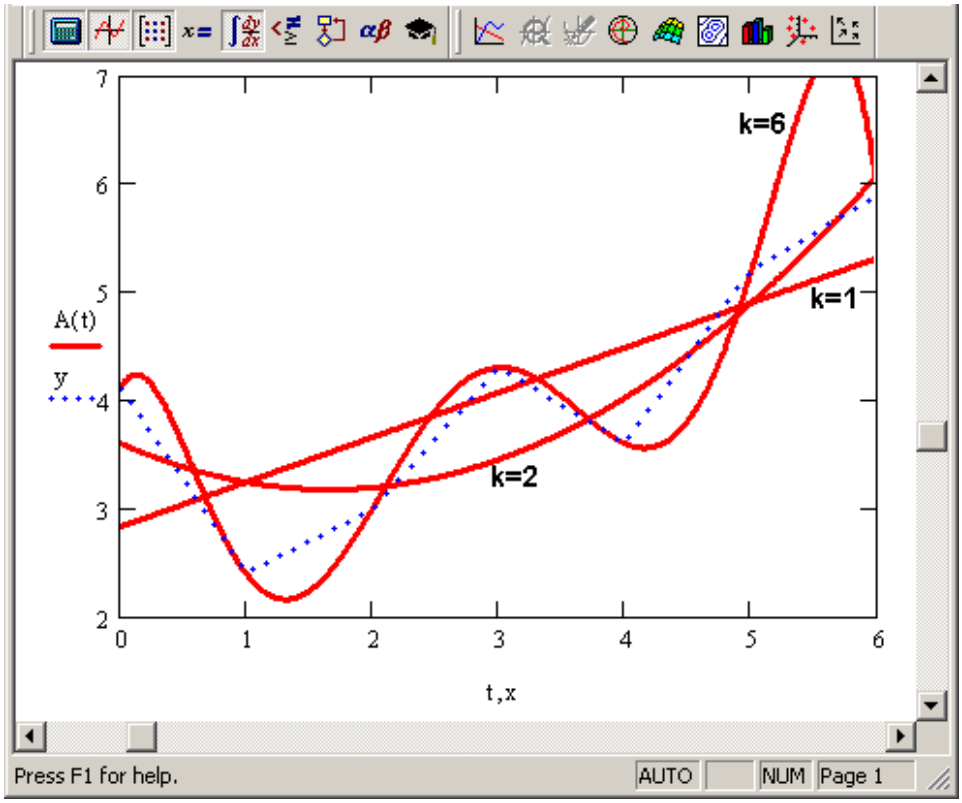


Рис. 7.26 — Регрессия полиномами разной степени

ПРИМЕЧАНИЕ

Для построения регрессии полиномом k-й степени необходимо наличие, по крайней мере, (***k***+1) *точек данных.*

В Mathcad полиномиальная регрессия осуществляется комбинацией встроенной функции **regress** и полиномиальной интерполяции:

- **regress (x, y, k)** — вектор коэффициентов для построения полиномиальной регрессии данных;
- **interp(s,x,y, t)** — результат полиномиальной регрессии:
- **s=regress(x,y,k);**
- **x** — вектор действительных данных аргумента, элементы которого расположены в порядке возрастания;
- **y** — вектор действительных данных значений того же размера;
- **k** — степень полинома регрессии (целое положительное число);
- **t** — значение аргумента полинома регрессии.

Пример полиномиальной регрессии квадратичной параболой приведен в листинге 10.

Листинг 10. Полиномиальная регрессия

```
x := (0  1  2  3  4  5  6)^T

y := (4.1  2.4  3  4.3  3.6  5.2  5.9)^T

   k := 2

 s := regress(x,y,k)

A(t) := interp(s,x,y,t)
```

7.13.3 Регрессия отрезками полиномов

Помимо приближения массива данных одним полиномом имеется возможность осуществить регрессию сшивкой отрезков (точнее говоря, участков, т.к. они имеют криволинейную форму) нескольких полиномов. Для этого имеется встроенная функция **loess**, применение которой аналогично функции **regress** (листинг 11 и рис. 7.27):

- **loess (x, y, span)** — вектор коэффициентов для построения регрессии данных отрезками полиномов;
- **interp(s,x,y,t)** — результат полиномиальной регрессии:
- **s=loess(x,y,span);**
- **x** — вектор действительных данных аргумента, элементы

которого расположены в порядке возрастания;

- **y** — вектор действительных данных значений того же размера;
- **span** — параметр, определяющий размер отрезков полиномов (положительное число, хорошие результаты дает значение порядка **span=0.75**).

Параметр **span** задает степень сглаженности данных. При больших значениях **span** регрессия практически не отличается от регрессии одним полиномом (например, **span=2** дает почти тот же результат, что и приближение точек параболой).

Листинг 11. Регрессия отрезками полиномов

```
x := (0 1 2 3 4 5 6)^T

y := (4.1 2.4 3 4.3 3.6 5.2 5.9)^T

s := loess(x,y,0.9)

A(t) := interp(s,x,y,t)
```

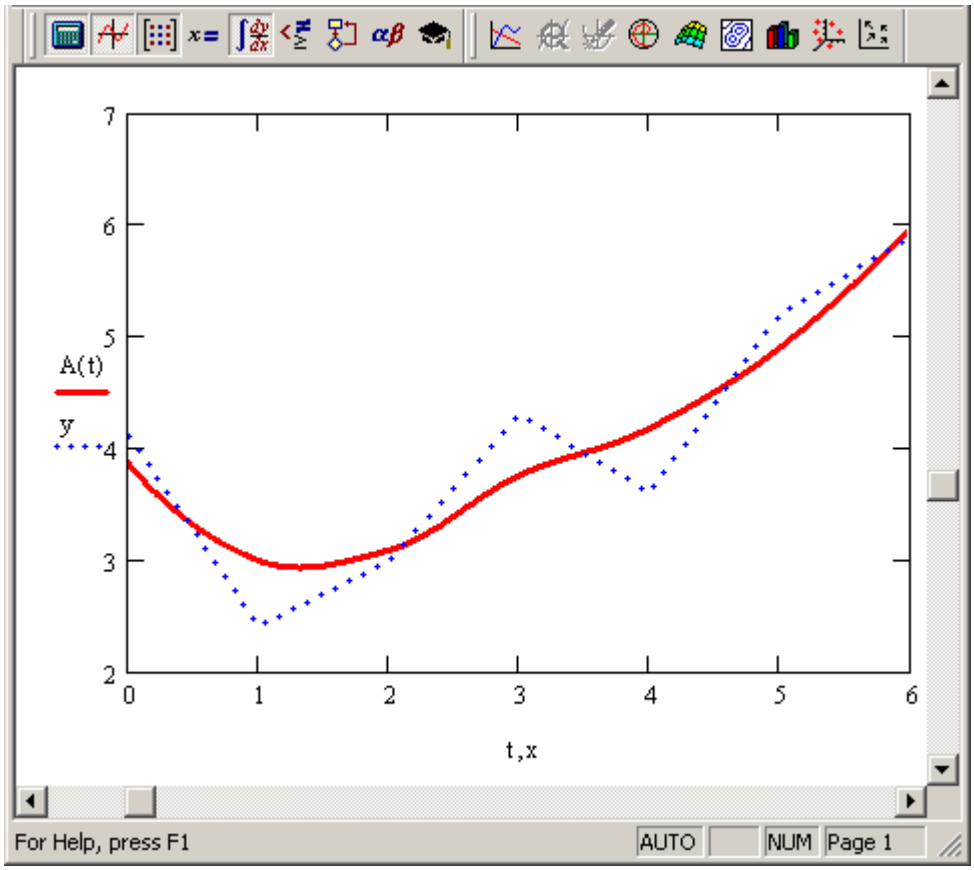


Рис. 7.27 — Регрессия отрезками полиномов

7.13.4 Двумерная полиномиальная регрессия

По аналогии с одномерной полиномиальной регрессией и двумерной интерполяцией, Mathcad позволяет приблизить множество точек $\mathbf{Z_{i,j}(x_i,y_j)}$ поверхностью, которая определяется многомерной полиномиальной зависимостью. В качестве аргументов встроенных функций для построения полиномиальной регрессии должны стоять в этом случае не векторы, а соответствующие матрицы.

- **regress (x,z,k)** — вектор коэффициентов для построения полиномиальной регрессии данных.
- **loess(x,z,span)** — вектор коэффициентов для построения регрессии данных отрезками полиномов.
- **interp(s,x,z,v)** — скалярная функция, аппроксимирующая данные выборки двумерного поля по координатам **x** и **y** кубическими сплайнами.
- **s** — вектор вторых производных, созданный одной из сопутствующих функций **loess** или **regress.**
- **x** — матрица размерности **Nx2**, определяющая пары значений аргумента (столбцы соответствуют меткам **x** и **y**).
- **z** — вектор действительных данных размерности **N.**
- **span** — параметр, определяющий размер отрезков полиномов.
- **k** — степень полинома регрессии (целое положительное число).
- **v** — вектор из двух элементов, содержащий значения аргументов x и y, для которых вычисляется интерполяция.

ВНИМАНИЕ! Для построения регрессии не предполагается никакого предварительного упорядочивания данных (*как, например, для двумерной интерполяции, которая требует их представления в виде матрицы **NxN**). В связи с этим данные представляются как вектор.*

Двумерная полиномиальная регрессия иллюстрируется листингом 12 и рис. 7.28. Сравните стиль представления данных для двумерной регрессии с представлением тех же данных для двумерной сплайн-интерполяции (см. листинг 6) и ее результаты с

исходными данными (см. рис. 7.22) и их сплайн-интерполяцией (см. рис. 7.23).

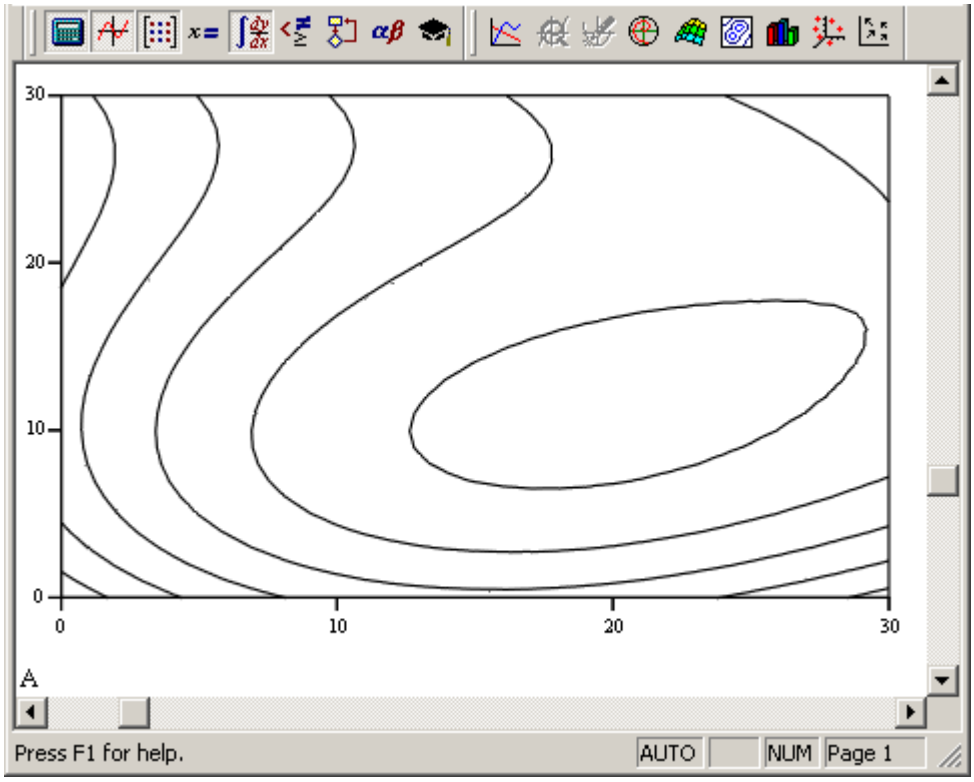


Рис. 7.28 — Двумерная полиномиальная регрессия (продолжение листинга 12)

Листинг 12. Двумерная полиномиальная регрессия

$$X := \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 2 & 2 & 2 & 2 & 2 & 3 & 3 & 3 & 3 & 3 & 4 & 4 & 4 & 4 & 4 \\ 0 & 10 & 20 & 30 & 40 & 0 & 10 & 20 & 30 & 40 & 0 & 10 & 20 & 30 & 40 & 0 & 10 & 20 & 30 & 40 & 0 & 10 & 20 & 30 & 40 \end{pmatrix}^T$$

$$Z := (1 \;\; 1 \;\; 0 \;\; 1.1 \;\; 1.2 \;\; 1 \;\; 2 \;\; 3 \;\; 2.1 \;\; 1.5 \;\; 1.3 \;\; 3.3 \;\; 5 \;\; 1.7 \;\; 2 \;\; 1.3 \;\; 3 \;\; 3.7 \;\; 2.1 \;\; 2.9 \;\; 1.5 \;\; 2 \;\; 2.5 \;\; 2.8 \;\; 4)^T$$

$$S := \text{regress}(X, Z, 3)$$

$$k := 30 \qquad i := 0..k \qquad j := 0..k$$

$$A_{i,j} := \text{interp}\left[S, X, Z, \begin{pmatrix} \frac{i}{k} \cdot 4 \\ \frac{j}{k} \cdot 40 \end{pmatrix}\right]$$

ПРИМЕЧАНИЕ. Обратите внимание на знаки транспонирования в листинге. Они применены для корректного представления аргументов (*например, z, в качестве вектора, а не строки*).

7.13.5 Другие типы регрессии

Кроме рассмотренных, в Mathcad встроено еще несколько видов трехпараметрической регрессии. Их реализация несколько отличается от приведенных выше вариантов регрессии тем, что для них, помимо массива данных, требуется задать некоторые начальные значения коэффициентов **a, b, c**. Используйте соответствующий вид регрессии, если хорошо представляете себе, какой зависимостью описывается ваш массив данных. Когда тип регрессии плохо отражает последовательность данных, то ее результат часто бывает неудовлетворительным и даже сильно различающимся в зависимости от выбора начальных значений. Каждая из функций выдает вектор уточненных параметров **a, b, c**.

- **expfit (x, y, g)** — регрессия экспонентой $\mathbf{f(x) = ae^{bx}+c}$.
- **igsfit (x,y,g)** — регрессия логистической функцией $\mathbf{f(x)=a/(1+be^{-cx})}$.
- **sinfit (x,y,g)** — регрессия синусоидой **f (x) = = a-sin (x+b) +c**.
- **pwfit(x,y,g)** — регрессия степенной функцией **f (x)=a-xb+c**.
- **logfit (x, y, g)** — регрессия логарифмической функцией **f (x)=aln(x+b) +c**.
- **infit (x,y)** — регрессия двухпараметрической логарифмической функцией **f (x)=aln(x)+b**.
- **x** — вектор действительных данных аргумента.

8 ЭЛЕМЕНТЫ ПРОГРАММИРОВАНИЯ НА С++

Язык Си был разработан в 70-е годы как язык системного программирования. При этом ставилась задача получить язык, обеспечивающий реализацию идей процедурного и структурного программирования и возможность реализации специфических приемов системного программирования. Такой язык позволил бы разрабатывать сложные программы на уровне, сравнимом с программированием на Ассемблере, но существенно быстрее. Эти цели, в основном, были достигнуты. Большинство компиляторов для Си сами написаны на этом языке, почти полностью написана на Си операционная система UNIX.

Недостатком Си оказалась низкая надежность разрабатываемых программ из-за отсутствия контроля типов. Попытка поправить дело включением в систему программирования Си отдельной программы, контролирующей неявные преобразования типов, решила эту проблему лишь частично.

На основе Си в 80-е годы был разработан язык Си++, вначале названный «Си с классами». Си++ практически включает язык Си и дополнен средствами объектно-ориентированного программирования. Рабочая версия Си++ появилась в 1983 г. С тех пор язык продолжает развиваться и опубликовано несколько версий проекта стандартов Си и Си++.

С++ — универсальный язык программирования, задуманный так, чтобы сделать программирование удобным для опытного программиста. За исключением второстепенных деталей С++ является надмножеством языка программирования Си.

Помимо возможностей, которые дает Си, С++ предоставляет гибкие и эффективные средства определения новых типов. Используя определения новых типов, точно отвечающих концепциям приложения, программист может разделять разрабатываемую программу на легко поддающиеся контролю части. Такой метод построения программ часто называют абстракцией данных. Информация о типах содержится в некоторых объектах типов, определенных пользователем. Такие объекты просты и надежны в использовании в тех ситуациях, когда их тип нельзя установить на стадии компиляции. Программирование с применением таких объектов часто называют объектно-ориентированным. При пра-

вильном использовании этот метод дает более короткие, проще понимаемые и легче контролируемые программы.

Ключевым понятием C++ является класс. Класс — это тип, определяемый пользователем. Классы обеспечивают скрытие данных, гарантированную инициализацию данных, неявное преобразование типов для типов, определенных пользователем, динамическое задание типа, контролируемое пользователем управление памятью и механизмы перегрузки операций. C++ предоставляет гораздо лучшие, чем в Си, средства выражения модульности программы и проверки типов. В языке C++ есть также усовершенствования, не связанные непосредственно с классами, включающие в себя символические константы, **`inline`** — подстановку функций, параметры функции по умолчанию, перегруженные имена функций, операции управления свободной памятью и ссылочный тип.

В C++ сохранены возможности языка Си по работе с основными объектами аппаратного обеспечения (биты, байты, слова, адреса и т.п.). Это позволяет весьма эффективно реализовывать типы, определяемые пользователем.

C++ и его стандартные библиотеки спроектированы так, чтобы обеспечивать переносимость. Имеющаяся на текущий момент реализация языка будет идти в большинстве систем, поддерживающих Си. Из C++ программ можно использовать Си библиотеки, и с C++ можно использовать большую часть инструментальных средств, поддерживающих программирование на Си.

8.1 Алфавит и лексемы языка C++

Алфавит языка C++ образован подмножеством символов ASCII. Это заглавные **A...Z** и строчные **a...z** буквы латинского алфавита, символ подчеркивания _, воспринимаемый как дополнительная буква латинского алфавита, арабские цифры 0...9, специальные символы:

!	%	^	&	*	(	)	-	+	=	{	}	|	~	[	]	\	;	'	:	«	<	>	?	,	.	/

В литеральных строках (см. ниже) возможно использование символов второй части кодовой таблицы ASCII (символов с кодом

128 и выше), где обычно располагаются буквы национальных алфавитов, в частности, кириллицы, символы псевдографики и т.п.

Компилятор C++ ***регистрочувствительный***, т.е. способен различать в лексемах заглавные и строчные буквы. Отметим, что компилятор PASCAL таким свойством не обладает.

Из символов алфавита формируются смысловые единицы языка — ***лексемы***.

Существуют лексемы пяти видов:

- идентификаторы,
- служебные слова,
- литералы,
- операции,
- разделители.

В тексте программы лексемы отделяются друг от друга т.н. «обобщенными» пробелами, к которым относят собственно символы пробелов, символы табуляции и перевода строки. Сюда же относятся и комментарии в тексте программы. Сколько бы ни было подряд идущих обобщенных пробелов, они игнорируются компилятором и воспринимаются им лишь как единый разделитель лексем.

8.2 Идентификаторы

Идентификатор — это *имя типа* или *имя объекта* в программе. Идентификатором служит последовательность латинских букв и цифр произвольной длины. К буквам относится и символ подчеркивания _. Обязательное условие: *первый символ идентификатора должен быть буквой или символом подчеркивания*. Напомним, что заглавные и строчные буквы различаются. Хотя длина символьной последовательности идентификатора не ограничена, но, например, компилятор Borland C различает только первые 32 символа идентификатора.

Являясь именем вводимого в программу объекта или определяемого пользователем типа, идентификаторы могут строиться в достаточной мере произвольно, однако, необходимо учитывать, что идентификатор не должен совпадать со служебным словом (см. ниже). Следует избегать применения идентификаторов с двумя лидирующими знаками подчеркивания, например **`__TIME`**,

т.к. в этом случае есть риск попасть на имя т.н. *переменной окружения*, используемой в среде программирования.

Практический совет. Вводя идентификатор в качестве имени объекта или типа, желательно отразить в нем некоторое смысловое содержание, это поможет пользователю (не компилятору!) легче ориентироваться в тексте программы.

Примеры идентификаторов:

`May_ident`	— правильный идентификатор
`AbraKadabra`	— правильный идентификатор
`Это_моеN`	— ***ошибка!***, *применение русских букв в идентификаторах недопустимо*
`5Ris`	***ошибка!***, *первым символом должна быть буква*
`_alfa`	— правильный идентификатор

8.3 Литералы

К литералам относят константы, значение которых задается в программе. В составе языка C++ имеется следующий набор литералов:

- символьные константы,
- целые арифметические константы,
- арифметические константы с плавающей точкой,
- строковые константы или строковые литералы.

Иногда в литературе символьные и арифметические константы выделяют как самостоятельный вид лексем.

8.3.1 Символьные константы

Символьная константа задается символом ASCII, заключенным в одинарные кавычки. Например: **`'A', 'g', '9','Ф', 'я'`**. Значение символьной константы — код ASCII символа, указанного в апострофах. Символьная константа занимает 1 байт памяти и имеет тип **`char`** (см. ниже).

В ряде случаев возникает необходимость задать символьной константой символ ASCII не имеющий графического представления. Такие символы, как правило, являются управляющими, их вывод на экран монитора, принтер или в текстовый файл вызывает изменение обычного режима вывода (осуществляется переход

на следующую строку, переход к началу строки, сдвиг точки вывода на позицию табуляции и т.п.). Символьные константы, соответствующие управляющим символам изображаются с использованием лидирующей обратной косой черты:

Константа	Обозначение	Действие
\n	NL (LF)	перевод на следующую строку
\t	HT	горизонтальная табуляция
\v	VT	вертикальная табуляция
\b	BS	шаг назад («забой» предыдущего символа)
\r	CR	возврат каретки (переход к началу строки)
\f	FF	перевод формата (авторегистр)
\a	BEL	звуковой сигнал

Кроме управляющих символов по аналогичной схеме строятся константы соответствующие некоторым символам, хотя и имеющим графическое представление, но уже задействованным в синтаксисе представления констант:

Символ	Изображение	Константа
обратная дробная черта	\	\\
одиночная кавычка	'	\'
двойная кавычка	"	\"

Задать символьную константу с любым кодом ASCII можно таким образом:

'**\xHH**' или '**\ooo**',

где **HH** — шестнадцатеричное число в диапазоне 0—FF,

ooo — восьмеричное число в диапазоне 0—377.

Числа, следующие за знаком обратной черты — код ASCII соответствующего символа.

8.3.2 Целочисленные константы

Целочисленные константы могут быть заданы в виде целых чисел со знаком в десятичном, восьмеричном и шестнадцатеричном представлении.

Примеры:

28953, -451 – десятичные константы;

0347, -0176 — восьмеричные константы;

0x1AF3, -0xAF89 – шестнадцатеричные константы.

Таким образом, признаком восьмеричной константы служит лидирующий нуль, а шестнадцатеричной — префикс 0x или 0X. Целочисленные константы могут иметь тип **int, long, unsigned int, unsigned long**.

8.3.3 Константы с плавающей точкой

Константы с плавающей точкой представляются в десятичной системе в двух формах:

- в виде последовательности десятичных цифр с точкой, оделяющей целую часть числа от дробной:

314.156 –0.2718 0.0654;

- в экспоненциальной форме с мантиссой и порядком:

1.62E-10, что эквивалентно 1.62×10^{-10}

-3.173e4, что эквивалентно -3.173×10^{4}

В экспоненциальном представлении разделитель мантиссы и порядка может быть представлен как заглавной буквой **E**, так и строчной **e**.

8.3.4 Литеральные константы

Литеральные константы или строки представляют собой последовательность символов ASCII, заключенных в двойные кавычки, например:

```
"Это литеральная строка" "C:\\Dir1\\dir2"
```

В составе литеральных констант могут содержаться любые символы ASCII, включая управляющие. Правила формирования символов аналогичны правилам построения символьных констант, рассмотренных выше.

Для обеспечения работы некоторых функций, оперирующих со строками, используется соглашение о завершении строки не

отображаемым «нуль-терминатором» — байтом со значением 0 (*Внимание: не кодом символа '0'!*).

При записи в тексте программы длинных строк удобно их фрагменты размещать на разных строках при этом нужно уведомить компилятор о том, что строка продолжается символом обратной косой черты \, например, запись

```
"Такая длин\
ная строка"
```

Будет воспринято компилятором как литерал

```
 "Такая длинная строка"
```

8.4 Служебные слова

Служебные слова — это зарезервированные лексемы. Они служат в качестве имен предопределенных типов и объектов, являются компонентами операторов и т.п. В языке C++ имеются следующие служебные слова:

_asm	_ds	int	_seg
asm	else	interrupt	short
auto	enum	_interrupt	signed
break	_es	_loadds	sizeof
case	_export	long	_ss
catch	extern	_near	static
_cdecl	_far	near	struct
cdecl	far	new	switch
char	fastcall	operator	template
class	float	_pascal	this
const	for	pascal	typedef
continue	friend	private	union
_cs	goto	protected	unsigned
default	_huge	public	virtual
delete	huge	register	void
do	if	return	volatile
double	inline	_saverages	while

Здесь нет смысла пояснять значения каждого из перечисленных служебных слов, разъяснения будут приводиться далее по мере необходимости. Этот список зарезервированных слов C++ приведен для того, чтобы случайно не использовать эти слова в качестве имен переменных, поскольку, напомним еще раз, *вводимые пользователем идентификаторы не должны совпадать со служебными словами*. Здесь идет речь о полном совпадении слов с учетом регистра. Например, пользовательский идентификатор **`static`** недопустим из-за его совпадения со служебным словом, а идентификаторы **`Static`** и **`stAtic`** вполне приемлемы.

В редакторах исходного кода, используемых в интегрированных средах разработки (IDE), таких как Borland C++, Visual C и т.д. служебные слова в тексте программы автоматически выделяются либо цветом, либо насыщенностью шрифта.

8.5 Операции

Все преобразования значений объектов — переменных осуществляются с помощью операций. В зависимости от количества участвующих в операции объектов аргументов (***операндов***) операции подразделяют на ***унарные***, применяемые к одному операнду, ***бинарные***, использующие два операнда и ***тернарные***, работающие с тремя операндами. На C++ используются в основном унарные и бинарные операции. Тернарная операция на C++ всего одна — условное арифметическое выражение. Синтаксис записи основных операции с кратким пояснением их смысла приведен в следующей таблице.

Тернарная операция

Оператор	Описание	Пример
a?b:c	Условное выражение: если ***a*** не нуль, то ***b***, иначе ***c***	x = a?b:c

Бинарные операции

Арифметические операции

Знак орерации	Описание	Пример
+	Сложение	x = x + z;
-	Вычитание	x = y - z;
*	Умножение	x = y * x;
/	Деление	x = y / z;
%	Остаток от деления целых чисел	r = i % j;

Поразрядные операции (применяются только к целым типам)

Знак орерации	Описание	Пример
&	Поразрядное И	c = a & b
|	Поразрядное ИЛИ	c = a | b
^	Поразрядное ИСКЛЮЧАЮЩЕЕ ИЛИ	c = a ^ b
<<	Поразрядный сдвиг кода влево	c = a << 3
>>	Поразрядный сдвиг кода вправо	c = a >> 4

Операции присваивания

Знак орерации	Описание	Пример
=	Присваивание	x = 10;

@=	Составное присваивание (здесь символом **@** обозначен любой из операторов: **+, -, *, /,&, |.**)	x @= a;
		то же, что и
		x = x @ a;

Логические операции

Знак орерации	Описание	Пример
&&	Логическое И	if (x && OxFF) {...}
||	Логическое ИЛИ	if (x || OxFF) (...)

Операции отношения

Знак орерации	Описание	Пример
==	Равно	if (x == 10) {...}
!=	Не равно	if (x != 10) {...}
<	Меньше	if (x < 10) {...}

Знак орерации	Описание	Пример
>	Больше	if (x > 10) {...}
<=	Меньше или равно	if (x <= 10) {...}
>=	Больше или равно	if (x >= 10) {...}

Унарные операции

Знак орерации	Описание	Пример
*	Косвенная адресация (разыменование)	int x = *y;
&	Взятие адреса	int* x = &y;
~	Поразрядное НЕ	x &= ~0x02;
!	Логическое НЕ	if (!v) {...}
++	Инкремент	x++; или ++x;
--	Декремент	x--; или --x;

Операции доступа к элементам объектов

Знак орерации	Описание	Пример
::	Разрешение области видимости	MC::F();
->	Доступ к элементам объекта через указатель (косвенный доступ)	MC->F();
.	Доступ к элементам объекта через его имя(прямой доступ)	MC.F();

Из приведенной таблицы видно, что один и тот же символ может использоваться для обозначения различных операций. В этом случае смысл операции определяется ее контекстом. Для обозначения операций на C++ используются как одиночные символы, так и группы из двух символов, причем группа из двух одинаковых символов, обозначает совершенно иную операцию, нежели одиночный символ.

Признаком того, что лексема определяет операцию является наличие знака или группы знаков операции. Если операция определена двухсимвольной группой знаков, то ее знаки нельзя разрывать обобщенным пробелом. Например, вместо **`a && b`** нельзя записать **`a & & b`**, это будет воспринято как две последовательно идущие операции.

Существуют три операции, обозначаемых служебными словами:

sizeof — определение размера области памяти в байтах, занятой объектом (не функцией!) и определение фактического размера памяти, отводимого под тип;

new — запрос на выделение памяти на этапе выполнения программы, достаточной для размещения объекта указанного типа;

delete — освобождение памяти, выделенной операцией **new**.

Примеры записи операций **sizeof**, **new** и **delete** приведены ниже:

s=sizeof A; — в **s** будет размер памяти в байтах, занимаемый объектом с именем A;

s=sizeof(int); — в **s** будет размер памяти в байтах, отводимой под тип **int**;

a=new double; — в **a** будет помещен начальный адрес динамически выделенной памяти, достаточной для размещения объекта типа **double**;

a=new double[10]; — в **a** будет помещен начальный адрес динамически выделенной памяти, достаточной для размещения массива из 10 объектов типа **double**;

delete a; — произойдет освобождение выделенной динамически памяти под объект с начальным адресом, записанным в **a**.

На C++ есть унарная операция явного преобразования типа, синтаксис которой имеет вид:

(type_new) T

Здесь **type_new** — имя любого типа, известного в программе к моменту выполнения операции, **T** — выражение типа, отличного от **type_new**. Эта операция предназначена для временной смены типа операнда. Так, например, если объявлена переменная **int A**, а по логике программы требуется представить ее значение как **double**, то значение выражения **(double)A** и даст желаемый результат.

8.6 Разделители

К символам разделителям языка C++ относятся:

Символ	Название	Назначение
{ }	Фигурные скобки	Выделение границ блоков (составных операторов)
[ ]	Квадратные скобки	Выделение индексов в массивах
()	Круглые скобки	Изменение порядка операций в арифметических выражениях. Выделение списка формальных и фактических параметров в функциях. Выделение параметров в заголовках условных операторов и операторов циклов
;	Точка с запятой	Обозначение конца оператора
,	Запятая	Разделение элементов в различных списках, например в списках параметров функций, инициализаторов и т.п.
:	Двоеточие	Отделяет метку от оператора. В заголовке производного класса обозначает начало списка базовых классов (только на C++). В заголовке функций — конструкторов обозначает начало списка инициализаторов (только на C++). В описаниях битовых полей отделяет имя поля от константы, задающей размер битового поля

8.7 Комментарии в программе

Комментарии в программе предназначены для пользователя, компилятором текст комментариев полностью игнорируется. На C++ существует два способа выделения комментариев:

```
/* Текст комментария произвольной длины*/

// Текст комментария до конца строки.
```

Первый способ выделения комментариев унаследован от языка Си, второй используется только в C++. В первом случае

текст комментария может занимать как часть строки, так и произвольное количество строк. Например:

```
int a=1, /*объект а инициализировали 1*/ b=100;
...
float d=3.7; /* Далее следует длинный
многострочный
комментарий, после которого
программа продолжается */ d *= 16.01;
...
```

Во втором случае текст комментария простирается от // до конца текущей строки, например:

```
f = A*cos(w*t); // далее до конца строки следует текст комментария
```

Кроме функций пояснения текста программы, комментарии удобно использовать при отладке программ. Закомментировав фрагмент текста программы, мы исключаем его из входного потока компилятора, без фактического удаления из файла.

Практический совет. Не пренебрегайте комментариями в тексте программы, они существенно помогают впоследствии разобраться в собственной программе, упрощают процесс сдачи программы преподавателю.

8.8 Типы и их описание

В C++ существуют *предопределенные* типы и типы, *определяемые пользователем*. Предопределенные типы «понимаются» компилятором без специального описания, имена предопределенных типов — это зарезервированные слова. Пользовательские типы нуждаются в специальном *определении* — их описании через предопределенные типы, либо через ранее созданные пользовательские типы. Слово «пользовательский» в данном контексте отнюдь не означает, что тип введен пользователем в данный момент составляющим программу. Пользовательский тип может быть создан кем-то и когда-то, но его описание должно быть обязательно включено в текст программы. Любой объект программы

должен быть описан до его первого использования в операторах, т.е. должен быть определен его тип. При описании переменных возможно проведение инициализации — присвоения значения непосредственно при создании объекта.

Типы в C++ подразделяют на основные и производные. Разница между ними проявляется при создании пользовательских типов: *в описании нуждается только основной тип, производные типы порождаются автоматически*.

По элементному составу типы делят на *скалярные*, *векторные* и *объектные*. Самыми элементарными являются скалярные предопределенные типы. Векторные типы (или массивы) образованы индексированным набором однотипных элементов. Объектные типы в общем случае представляют собой агрегат из элементов разного типа. Каждый элемент объектного типа имеет свое имя. На C++ элементами объекта могут быть функции.

В этом разделе мы рассмотрим предопределенные скалярные и векторные типы, рассмотрение объектных типов отложим до раздела, посвященного объектно-ориентированному программированию.

8.9 Основные скалярные предопределенные типы

Имена основных скалярных предопределенных типов C++ таковы:

Имя типа	Название
char	символьный тип
short	короткий целый
int	целый
long	длинный целый
float	плавающий обычной точности
double	плавающий двойной точности
long double	длинный плавающий двойной точности
void	«пустой» тип

Первые четыре типа используются для представления целых чисел, следующие три — для представления чисел с плавающей точкой (чисел с дробной частью).

8.9.1 Тип `void`

Тип **`void`** — специфический тип C++, он не применяется в качестве основного при создании объектов, используются лишь производные от него типы. Синтаксически **`void`** эквивалентен основным типам, но использовать его можно только в производном типе. Объектов типа **`void`** не существует.

С его помощью задаются указатели на объекты неизвестного типа или функции, не возвращающие значение.

```
void f();       // функция f не возвращает значения
void* pv;       // указатель на объект неизвестного типа
```

Указатель произвольного типа можно присваивать переменной типа **`void*`**. На первый взгляд этому трудно найти применение, поскольку для **`void*`** недопустимо косвенное обращение (разыменование). Однако, именно на этом ограничении основывается использование типа **`void*`**. Он приписывается параметрам функций, которые не должны знать истинного типа этих параметров. Тип **`void*`** имеют также бестиповые объекты, возвращаемые функциями.

Для использования таких объектов нужно выполнить явную операцию преобразования типа. Такие функции обычно находятся на самых нижних уровнях системы, которые управляют аппаратными ресурсами. Приведем пример:

```
void* malloc(unsigned size);// функция библиотеки Си,
                            //предназначенная для выделения
                            // динамической памяти на этапе
                            // выполнения программы
void free(void*););// функция библиотеки Си,
                   //предназначенная для освобождения
                  //ранее выделенной динамической памяти

void f()              // распределение памяти в стиле Си
    {
      int* pi = (int*)malloc(10*sizeof(int));
              // выделеляется память под массив из 10 int
      char* pc = (char*)malloc(10);
              // выделеляется память под массив из 10 char
```

```
    //...
    free(pi);
    free(pc);
}
```

На C++ все скалярные предопределенные типы, исключая **void**, являются арифметическими, т.е. соответствующие им значения могут интерпретироваться как числа и к ним применимы арифметические операции.

Обозначение: (*тип*) *выражение* — используется для задания операции преобразования выражения к типу (см. ниже), поэтому перед присваиванием **pi** тип **void***, возвращаемый в первом вызове **malloc()**, преобразуется в тип **int**. Пример записан в архаичном стиле; лучший стиль управления размещением в свободной памяти реализован в C++ операцией new (см. ниже).

8.9.2 Объявление скалярных предопределенных типов

Директива объявления переменной состоит из имени типа и имени — идентификатора объявляемой переменной. Например:

```
int A, i; // объявлены переменные знакового целого
 // типа с именами A и i;
float ff1; // объявлена переменная вещественного типа
   // обычной точности с именем ff1;
char hh0,cd;     // объявлены переменные символьного типа
// с именами hh0 и cd;
long VV      // объявлена переменная типа знакового длинного
// целого с именем VV;
float ff1; // объявлена переменная вещественного типа
   // обычной точности с именем ff1;
```

На C++ все скалярные предопределенные типы, исключая void, являются арифметическими, т.е. соответствующие им зна-

чения могут интерпретироваться как числа и к ним применимы арифметические операции.

Отрицательные целые числа во внутреннем формате представляются в дополнительном коде, при этом их старший бит интерпретируется как знаковый: для отрицательных чисел он 1, для положительных 0. Такую интерпретацию состояния старшего бита можно включать и выключать, добавляя в объявлении перед именем типа прилагательное **signed** (знаковый) и **unsigned** (беззнаковый). При применении прилагательных к типу **int** имя типа можно опускать. Например:

```
unsigned UI, T; // объявлены переменные UI и T
    //беззнакового целого типа;
signed I, k; // объявлены переменные I и k  знакового целого типа;
unsigned long V // объявлена переменная V беззнакового
    // длинного целого типа;
unsigned char ch; // объявлена переменная ch беззнакового
// символьного типа.
```

Прилагательное **signed** употребляют редко, т.к. этот режим включен по умолчанию. В переменных объявленных как **unsigned** старший бит интерпретируется обычным образом, представляемое число положительное, двоичный код — прямой.

В памяти компьютера переменные типов **int** и **long** располагаются так, что в первом байте, адрес которого совпадает с адресом переменной, располагаются младшие разряды двоичного кода числа, во втором и последующих байтах — старшие.

Можно запретить изменение значения объекта — данных в программе, употребив при его описании модификатор **const:**

```
const int c = 17;
const float e = 2.71828;
```

После такого объявления компилятор сам будет блокировать любую попытку изменения значений объекта, например, путем использования его в левой части оператора присваивания.

Очевидно, что значение константный объект может получить только путем инициализации при объявлении, далее это значение остается неизменным. Попытка создать константный объект без его инициализации вызывает ошибку на этапе компиляции.

8.9.3 Размеры памяти, занимаемые предопределенными типами

Переменная типа **`char`** имеет размер, естественный для хранения символа на данной машине, обычно, это — байт. Переменная типа **`int`** имеет размер, соответствующий целой арифметике на данной машине: 2 байта для 16-разрядных компьютеров и 4 байта для 32-разрядных.

Все перечисленные типы являются арифметическими, к ним применимы арифметические операции и операции сравнения, значения переменных этих типов в этом случае интерпретируются как числа. Диапазон целых чисел, которые могут быть представлены соответствующим типом, зависит от его размера:

Тип	Размер в байтах	Диапазон
char	1	$[-2^7, 2^7-1]$ или [–128, 127]
short	2	$[-2^{15}, 2^{15}-1]$ или [–32768, 32767]
int	2 или 4	см. **`short`** или см. **`long`**
long	4	$[-2^{31}, 2^{31}-1]$ или [–2147483648, 2147483647]
unsigned char	1	$[0, 2^8-1]$ или [0, 255]
unsigned short	2	$[0, 2^{16}-1]$ или [0, 65535]
unsigned int	2 или 4	см. unsigned short или unsigned long
unsigned long	4	$[0, 2^{32}-1]$ или [0, 4294967295]

Числа с плавающей точкой во внутреннем формате представляются в экспоненциальном формате: $\pm M \times 2^P$, где M — мантисса, при этом, если число $\neq 0$, то $1 \le M < 2$, P — порядок, целое число. Если число равно 0, то и мантисса и порядок равны нулю. Число F в плавающей форме выражается через биты своего представления следующим образом:

$$F = (-1)^s \cdot 2^p \left(1 + \sum_{k=0}^{n-1} 2^{k-n} \cdot q_k\right), \text{ где } p = \sum_{k=0}^{m-1} 2^k \cdot t_k - 2^{m-1} - 1.$$

Здесь **p** — двоичный порядок,

n — число разрядов мантиссы,

m — число разрядов порядка,

s — значение знакового разряда,

$\mathbf{q_k}$ и $\mathbf{t_k}$ — значения двоичных разрядов мантиссы и порядка соответственно.

Параметры **n** и **m** для разных типов имеют следующие значения:

float:	n = 23	m=8
double:	n = 52	m = 11
long double:	n = 63*	m = 15

*В представлении значений **`long double`** самый старший 64-й разряд мантиссы занят разрядом с весом 2^0, всегда установленным в 1. Для **`float`** и **`double`** этот разряд виртуальный, его наличие подразумевается, но физически он не существует.

Диапазоны представления чисел с плавающей точкой:

Тип	Размер в байтах	Диапазон
float	4	$[-2^{127}, 2^{127}]$ или $[-1.70\times10^{38}, 1.7\times10^{38}]$
double	8	$[-2^{1024}, 2^{1024}]$ или $[-1.8\times10^{308}, 1.8\times10^{308}]$
long double	10	$[-2^{16384}, 2^{16384}]$ или $[-1.19\times10^{4932}, 1.19\times10^{4932}]$

8.10 Производные типы

Если в программе введен тип, например, с именем **`type`**, то компилятор C++ сразу же начинает «понимать» производные от него типы:

type[]	вектор (массив) элементов типа **`type`**
type*	указатель на тип с именем **`type`**
type&	ссылка на тип с именем **`type`**
type()	функция, возвращающая значение типа **`type`**

Примеры объявлений объектов производных типов:

```
char v[10]      //объявлена переменная v — вектор (массив)
                //из 10 элементов типа char;
char* p         //объявлена переменная p — указатель на объект типа char;
int& u          //объявлена переменная u — ссылка на объект типа int;
int Fn()        //объявлена функция с именем Fn, возвращающая
                //значение типа int, не принимающая никаких аргументов
```

Техника работы с объектами основных и производных типов будет обсуждаться ниже в соответствующих разделах.

8.10.1 Векторные типы (массивы)

Вектором или массивом называется совокупность данных одинакового типа, объединенных одним именем с возможностью обращения к отдельному элементу по его номеру (индексу). Так, например, при объявлении объекта

```
type M[n];
```

где **type** – имя типа,

M – имя объекта,

n — положительная целочисленная константа.

Компилятор создает в памяти непрерывный сегмент, в котором последовательно один за другим расположены **n** элементов типа **type**. Доступ к элементам массива производится по индексу — целочисленной переменной или константе, указывающей номер элемента, например, **M[0], M[1], M[10].** *В C++ индексация массивов начинается с* 0*,* так что в нашем примере имеет смысл изменять индекс от 0 до n–1.

Внимание! На C++ нет автоматического контроля выхода индекса за границы массива, указанные при объявлении. Такой контроль возложен на программиста.

С именем массива M связывается адрес его первого элемента **M[0],** адрес i-го элемента определяется путем прибавления к этому адресу величины **i×sizeof (type).** Массивы могут быть образованы из элементов любого типа, как предопределенного типа, так и типов, определяемых пользователем. Элементом массива может служить другой массив. По такому принципу на

C++ строятся многомерные массивы. Синтаксис объявления двумерного массива:

```
int IM [12][10]
```

Даже по структуре объявления видно, что вводится 12 массивов по 10 элементов в каждом (или, что эквивалентно, 10 массивов по 12 элементов).

При объявлении массива обязательно в квадратных скобках указывается количество элементов в нем. Исключение из этого правила — ситуация, когда размер массива определяется автоматически из инициализирующего выражения. В этом случае в квадратных скобках можно ничего не писать, но наличие самих скобок обязательно. Например:

```
int C[] = {1,2,4,7};
char S[] = "Всем привет!";
```

Первой директивой создается массив из 4-х элементов типа **int**, которые сразу же после создания инициализируются значениями 1, 2, 4 и 7. Во втором случае создается массив элементов типа **char**, инициализированный литеральной константой. Размер массива, созданного второй директивой равен числу символов литеральной константы + 1 с учетом того, что последний символ литеральной константы — «нуль-терминатор».

8.10.2 Указатели C++

Указателем на C++ называют объекты значение, которых интерпретируется как адрес другого объекта. Можно выделить две категории указателей: ***указатели данных*** *и* ***указатели функций***. В пределах этого раздела рассматриваются только указатели на данные, указатели функций рассмотрены ниже.

Хотя адреса и представляют собой числа тождественные по формату числам типа **unsigned int** или **unsigned long**, указатели имеют свои собственные правила и ограничения на присваивания, преобразования и выполнение с ними арифметических действий.

8.10.2.1 Объявления указателей

На C++ практически не употребляется просто термин «указатель», а используется термин «указатель на тип». Тем самым подчеркивается, что указатель каким-то образом «типизирован», т.е. некоторые важные свойства указателя зависят от типа объекта, на который он указывает. Синтаксис объявления указателей следующий:

имя_типа *имя_указателя;

где ***имя_типа*** — имя типа объектов, адреса которых может хранить указатель;

имя_указателя — имя самого указателя, его идентификатор.

Важно, чтобы символ * располагался между именем типа и именем указателя. При этом неважно «прижат» ли этот символ к имени типа или к имени указателя и, вообще, не имеет значения сколько пробелов отделяют * от имени типа и от имени указателя.

Примеры объявления указателей на данные:

```
char *ucc;        // объявлен указатель на тип char с именем ucc
int* ui;          // объявлен указатель на тип int с именем ui
```

Поскольку указатель сам по себе является объектом, можно создать указатель на указатель, например:

```
int ** uui; // в uui может содержаться адрес указателя на тип int
```

Процесс «наращивания» числа звездочек можно продолжить, создавая указатель указателя на указатель и т.д.

Являясь производным типом, указатель может быть объявлен на любой ранее определенный тип. Более того, в самом определении пользовательского типа на основе структуры допустимо использовать в качестве поля указатель на тип, определяемый этой структурой, например:

```
struct List // определение пользовательского типа List
{
char IT[100];
List *next;       // указатель на определяемый тип в
//составе полей этого типа
};
```

Размер памяти, отводимой под указатель, зависит от используемой модели памяти, размера адресного пространства компьютера. В 16-разрядных компьютерах указатели могут быть «ближними» — **near** и «дальними» — **far**, например:

```
char near* uc;
int far* U;
```

Указатель **near** имеет размер 2 байта и может хранить адрес в пределах 64 килобайтного сегмента, указатель **far** занимает 4 байта и содержит полный 32-х разрядный адрес. В 32-разрядных системах используются только 4-х байтовые указатели. Модификаторы **near** или **far** не обязательно указывать в объявлениях — один из них устанавливается по умолчанию, какой именно — указывается в опциях настройки компилятора.

8.10.2.2 Операции с указателями данных и адресная арифметика

Указатели на разные типы не совместимы. Это значит, что автоматического преобразования типов не предусмотрено и попытка простого присвоения значения указателя на один тип указателю на другой тип будет блокирована компилятором. Исключением из этого правила является указатель на тип **void.** Указателю на **void** можно присваивать значение указателя на любой тип, поэтому этот указатель называют родовым. Обратное — неверно: попытка присвоить значение указателя на **void** указателю на любой другой тип без явного преобразования типа невозможна. Ввиду того, что автоматическое приведение типов для указателей отсутствует, в случае необходимости присвоения значения

указателя на тип **type1** указателю на тип **type2** следует использовать явное преобразование типов.

Пример:

```
type1 *T1; // объявлен указатель T1 на тип type1
type2 *T2; // объявлен указатель  T2 на тип type2
void *V;  // объявлен указатель V на тип void
T1 = T2; // ошибка! указатели на различные типы несовместимы
T1 = (type1*) T2; // присвоение значения указателя T2 указателю
   // T1 с использованием операции явного
  // преобразования типа
V = T1;// указателю на void можно присваивать
V = T2; //  значения указателей на любой тип
T1 = V; // ошибка!  если type1 не void
T1 =(type1*)V; // возможно с явным преобразованием типа
```

С указателями связаны две унарные операции: операция взятия адреса именованного объекта **&** и операция разыменования указателя *. Операция взятия адреса применяется для занесения значения адреса ранее введенного объекта в указатель, например:

```
int A; // введен объект с именем A типа int;
...
int *ua = &A; // введен  ua - указатель на int
                  //и инициализирован адресом объекта A;
```

Разыменование указателя — это ссылка на значение переменной, адрес которой содержится в указателе. По сути, это операция косвенной адресации. Например, смысловое содержание директивы `*u = 34;` таково: «объекту, адрес которого находится в **u** присвоить значение 34».

Из арифметических операций над указателями на один и тот же тип допустима только операция вычитания. Над указателями можно выполнять операции инкремента и декремента, к ним

можно также прибавлять (вычитать) значения переменных и констант целого типа. Операции инкремента / декремента значения указателя и их сложение / вычитание с целыми числами ***выполняются специфически***. Пусть объявлен указатель **Ut** на тип с именем **type**: **type* Ut**, тогда операция **Ut++** вызывает увеличение значения указателя на величину **sizeof(type)**, а операция **Ut+=n**, где **n** — целое число, увеличивает значение указателя на **n*sizeof(type)**.Обратим внимание на сходство арифметических операций с указателями с операциями над индексами массивов: добавление арифметической константы к указателю вызывает такое же изменение адреса, как увеличение на ту же величину индекса массива.

Внимание! Созданные и неинициализированные указатели опасны. В неинициализированном указателе содержится случайный код, воспринимаемый как адрес. Модификация значения объекта по такому адресу может привести к непредсказуемым последствиям.

В ряде случаев значение указателя нужно установить на такой адрес, которого заведомо не существует в адресном пространстве. В качестве такого адреса «в никуда» используют нулевое значение кода в указателе. Для этого специфического адреса введено мнемоническое обозначение **NULL**. Значение **NULL** эквивалентно значению арифметической константы 0.

8.10.2.3 Модификатор const и указатели

Модификатор **const** с указателями может использоваться в следующих контекстах.

```
const type * u1;            // объявлен указатель на константу
type *const u2 = &A;        // объявлен указатель - константа
const type *const u3=&A;  // объявлен указатель - константа
                                  // на константу
```

В первом случае будет блокироваться попытка изменения значение объекта, на который указывает **u1**. Во втором случае за-

прещено изменение значения указателя **u2** (изменение адреса находящегося в нем). Поскольку изменение значения **u2** невозможно, он обязательно должен быть инициализирован при объявлении, что и сделано в примере. Отсутствие инициализирующего выражения при объявлении указателя-константы вызывает ошибку компиляции. Третий случай объединяет предыдущие два: нельзя изменять ни адрес, ни значение объекта, на который указывает указатель. В последнем случае фактически запрещено изменять значение объекта **A,** ссылаясь на него через указатель **u3**, хотя изменение значения **A** при ссылке на него по имени вполне допустимо.

8.10.3 Ссылки

На C++ существует производный тип ***ссылка на тип***. Объявление ссылки на тип с именем **type** выглядит следующим образом:

```
type & RT = AA;
```

где **type** – имя типа,
RT – имя ссылки,
AA — имя ранее определенной переменной типа **type**.

В отличие от указателя ссылку невозможно объявить без инициализации. После объявления ссылка становится еще одним именем инициализирующей переменной.

Например:

```
int V1;          // описание объекта V1 типа int
int &r = V1;     // описание объекта r типа ссылка на
                 // int с инициализацией V1
...
r = 12;          // действия этих операторов
V1=12;           // эквивалентны
```

Рассматривая ссылки вне связи с функциями, трудно понять (и объяснить), зачем одному и тому же объекту может понадобиться еще одно имя. Далее, рассматривая передачу параметров

при вызове функций, покажем, сколь велики различия в механизме передачи параметров объявленных как ссылка.

8.11 Объектные типы

Наряду с массивами, являющимися агрегатами элементов одного и того же типа, на C++ возможно построение агрегатных типов с элементами разного типа, включая элементы — функции. Такие типы называют объектными. Существуют четыре вида объектных типов C++:

- класс;
- структура;
- объединение;
- битовые поля.

Объектные типы создаются пользователем и нуждаются в обязательном описании, т.к. необходимо указать из каких элементов они состоят. Такое описание типа, или, как говорят, его *определение* должно быть сделано до первого их использования в программе.

Классы C++ требуют отдельного рассмотрения, что будет сделано в разделах, посвященных объектно-ориентированному программированию. Сейчас же начнем рассмотрение только со структуры, причем в том их представлении, которое используется в Си. Дело в том, что классы и структуры C++ являются развитием структур Си. Класс и структура C++ — одно и то же, и различаются регламентом доступа к элементам по умолчанию.

8.11.1 Структуры

Структура на Си предназначена для описания составных объектных типов, имеющих в своем составе *поля* различного типа. В описание объектного типа, представленного структурой на C++, могут включаться и функции, но эту ситуацию мы рассмотрим позже, здесь же ограничимся рассмотрением структур, содержащих только поля данных. Описание структуры с полями имеет следующий вид:

struct ***тег_структуры***

{

тип1 поле1;

тип2 поле2;

...

типN полеN;

};

Это т.н. *протокол описания структуры*. За ключевым словом **struct** следует *тег_структуры* — идентификатор, который впоследствии будет являться именем определяемого типа. Далее в блоке из фигурных скобок следуют операторы описаний полей, входящих в структуру. Протокол описания структуры должен обязательно заканчиваться точкой с запятой, следующей за завершающей закрывающей фигурной скобкой.

Поля могут быть образованы как основными, так и производными типами. Они могут быть скалярными, векторными (массивами), другими определенными ранее структурами, указателями и ссылками на различные типы.

Например:

```
struct Mst
{
int n;
double *D;
char Na [20];
};
```

После такого описания в программу вводится новый основной тип с именем **Mst,** вместе с ним возникают и все производные от него типы. Теперь это имя типа можно точно так же как использовались имена предопределенных типов **int, float, char** и т.п.

Например, объявлением **Mst A, B;** создаются две переменных типа **Mst** с именами **A** и **B**. Объявление **Mst *U;** создает

переменную с именем **U** — указатель на тип **Mst**. У каждого из созданных объектов основного типа будут свои собственные поля с именами **n, D** и **Na**, обратиться к которым можно по составному имени, например:

```
A.n = 3;  // полю n объекта A присваивается значение 3
B.n = 156;  // полю n объекта B присваивается значение 156
A.Na[10] ='a';  // 10-му  элементу поля Na объекта A
                // присваивается значение 'a'.
```

Таким образом, синтаксис обращения к полю объекта типа структуры по имени имеет вид:

имя_объекта.имя_поля

Если объект имеет поля объектного типа, то количество элементов, записываемых через точку, увеличивается. Например, пусть имеется описание типа с именем **FW**:

```
struct FW
{
struct {int IA; int JB} ST; // поле с именем ST,
                            // созданное на основе структуры
int fib;
};
```

и в дальнейшем создан объект с именем **OB** типа **FW**:

```
FW OB;
```

тогда добраться до составляющих **IA** и **JB** поля **ST** можно следующим образом:

```
OB.ST.IA = 67;   OB.ST.JB = 53;
```

Обратим внимание на описание типа поля **ST** в протоколе описания структуры **FW**: там использована структура без тега. Использование такой безымянной структуры не приводит к созданию нового типа, структура лишь описывает тип конкретного объекта с именем **ST**.

К полям переменной объектного типа можно обращаться через указатель. Пусть определен тип с именем **DF:**

```
struct DF
{
 char str[30];
 float F;
};
```

и созданы переменная объектного типа **DF** с именем **T** и указатель на тип **DF** с именем **UU**:

```
DF T ;
DF *UU= &T;
```

при этом указатель инициализирован адресом объекта **T**.

Теперь в **UU** содержится адрес объекта **T**, и к его полям можно обратиться через указатель, например:

```
UU -> F = 3.1415; UU -> str[14] = 'g';
```

Таким образом, синтаксис доступа к полям переменной объектного типа, адрес которой помещен в указатель, имеет следующий вид:

имя_указателя -> имя_поля;

Знак операции доступа здесь состоит из двух символов — (минус) и > (больше). В случае, когда поле объекта тоже является указателем на объект, определенный структурой, стрелок в выражении доступа может быть более одной.

Инициализация объектов типа структуры может быть произведена аналогично инициализации массива (см. выше), но с учетом типов и длины полей. Например, пусть определен тип **FF**:

```
struct FF
{
char N[6];
int A;
```

```
float ff;
char c;
};
```

Создание переменной с именем **TY** типа **FF** с одновременной инициализацией ее полей заданными значениями может выглядеть так:

```
FF TY={'П','о','л','е',' ','N',321,7.895,'y'};
```

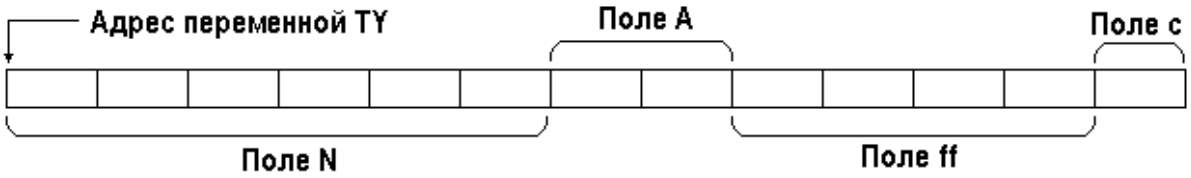
Список констант, перечисленных через запятую в фигурных скобках, представляет значения полей и их компонентов, которые будут установлены при создании переменной **TY**. Поле, образованное символьным массивом можно инициализировать одной литеральной константой, например:

```
FF TY={"Поле N", 321, 7.895, 'y'};
```

Результат будет тот же, что и в первом примере инициализации. В данном случае литеральная константа поместится в поле **TY.N** без завершающего нулевого байта. Потеря завершающего нуля может привести к проблемам, если в дальнейшем поле **TY.N** будет использовано как строка. Если же размер поля будет недостаточен для занесения информационных символов литеральной константы, то на этапе компиляции будет выдана ошибка:

```
FF TY={"Поле N23",321,7.895,'y'};// ошибка!
                    // строка не помещается в поле TY.N
```

Поля переменной типа структуры располагаются в памяти последовательно. В нашем примере компоненты переменной **TY** будут расположены в памяти следующим образом:



Таким образом, поля структуры располагаются в байтах памяти последовательно друг за другом в соответствии порядком их записи в описании структуры и в соответствии с типом и длиной каждого поля. Начальный адрес первого по порядку записи поля совпадает с адресом самой объектной переменной. Размер памяти, занимаемой объектной переменной равен суммарному размеру всех ее полей.

В нашем примере

```
sizeof TY=sizeof TY.N+sizeof TY.A+sizeof TY.ff+sizeof
TY.c =13
```

8.11.2 Объединения

Объединения или, как их еще называют *смеси*, имеют синтаксис определения сходный со структурой, различие лишь в том, что вместо ключевого слова **`struct`** используется ключевое слово **`union`**:

`union` ***тег_объединения***

{

тип1 поле1;

тип2 поле2;

...

типN полеN;

};

Здесь также тег объединения становится именем типа, синтаксис доступа к элементам полностью аналогичен доступу к элементам структуры. Различие структуры и объединения в том, что *все поля объединения начинаются с одного и того же адреса*. Этот адрес совпадает с адресом объектной переменной. Поля в объединении перекрываются, их значения как бы «смешиваются», отсюда и происхождение второго названия объединений.

Пример описания объектного типа **`union`**:

```
union UN
{
char BY[4];
int I[2];
unsigned long L;
float F;
};
```

В этом примере произведено описание объектного типа с именем **UN**. Поля имеют различный тип, но одинаковую длину. Начинаясь с одного и того же адреса и имея одинаковую длину, все четыре поля расположены в одном и том же участке памяти и адресуются к одной и той же битовой последовательности. Интерпретация же этой последовательности разная: если обращаться к ней через поле **BY**, последовательность будет интерпретирована как массив из 4-х символов, если обращение произведено через поле **I**, то содержимое последовательности будет воспринято как массив из 2-х целых чисел. Соответственно обращение к полю **L** — даст интерпретацию как значения переменной типа **unsigned long**, к полю **F** — как значения переменной типа **float**. Такая различная интерпретация одной и той же битовой последовательности может быть полезна в некоторых приложениях.

В приведенном примере все поля типа **UN** имели одинаковую длину, в общем случае, разумеется, длина полей может быть различной.

Размер памяти, занимаемый объектом типа **union,** равен максимальному размеру входящему в него поля.

8.11.3 Битовые поля

Минимальной единицей адресации в программах является байт. Непосредственная адресация к отдельному биту или произвольной группе битов невозможна. В некоторых важных приложениях потребность в таком доступе возникает. На C++ имеются

средства, позволяющие организовать доступ к битовым группам — это битовые поля.

Объектный тип с битовыми полями имеет синтаксис определения почти тот же, что и тип, определяемый структурой, даже ключевое слово то же самое — **`struct`**:

```
struct тег_структуры
{
целый_тип1 поле1 : целая_константа1;
целый_тип2 поле2  : целая_константа2;
...
целый_типN полеN    : целая_константаN;
};
```

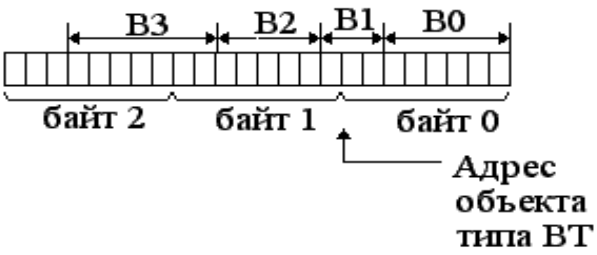
В отличие от описания объектного типа «структура», рассмотренного ранее здесь имеются следующие особенности:

- допускаются только следующие скалярные типы полей: **`char,unsigned char, int, unsigned int`**;
- после имени поля ставится разделитель: (двоеточие) и далее в форме целочисленной константы указывается ***длина битового поля в битах***.

Размер битового поля не может превышать числа битов в типе, имя которого указано слева. Это 8 для **`char`**, для **`int`** — либо 16, либо 32, в зависимости от реализации компилятора. Пусть определен тип **`BT`**, представляющий собой битовое поле:

```
struct BT
{
unsigned B0  : 6;
unsigned B1  : 3;
unsigned B2  : 5;
unsigned B3  : 7;
};
```

Расположение битовых полей типа BT в байтах адресного пространства показано на рисунке:



Первое по порядку следования в протоколе описания битовое поле (в нашем примере **B0**) располагается в *младших* битах первого байта адресного пространства, занимаемого объектом типа битового поля.

При размещении в памяти объекта типа битовое поле его размер выравнивается до целого числа байт. В нашем примере под объект типа **BT** отведено 3 байта.

8.12 Объявления и определения C++

На C++ существуют описания двух типов:

- объявления;
- определения.

Объявление содержит краткую характеристику объекта. При объявлении переменной приводится имя соответствующего ей типа и ее идентификатор. При объявлении функций дается только описание ее интерфейса (см. ниже). Определение дает более развернутую информацию. В определении переменной может присутствовать описание типа, если он определяется пользователем, и инициализатор, задающий начальные значения элементам объекта. Определение функции содержит текст ее программы (тело функции).

Примеры объявлений:

`int A;`	// объявлена переменная с именем **A**, // имеющая тип **int** (знаковый целый)
`char C[10];`	// объявлен вектор (массив) с именем **C**, состоящий из 10 элементов типа **char** (знаковый)
`float d, d1;`	// объявлены две переменные (списком), имеющие тип **float** (вещественный с плавающей точкой)
`struct SW;`	// объявлена структура с именем **SW**

`int Fn(float);`	// объявлена функция с именем **Fn**, возвращающая значение типа **int** и принимающая один параметр типа **float**.
`typedef unsigned WORD`	// для типа **unsigned int** объявлено новое имя типа **WORD**

Примеры определений:

`int A = 32;`	// определена переменная с именем **A**,
	// имеющая тип **int** и инициализирована
	// значением 32 при создании.

`float d=2.33,d1=1.75;`	// определены и инициализированы две // переменные типа **float**

`struct SW`
`{`
`  char C;`
`  float fl;`
`};`
	// определен пользовательский тип
	// — структура с именем **SW** с двумя
	// элементами: **C** – типа **char** и
	// **fl** — типа **float**

`int Fn(float x)`
`{`
`return x/7;`
`}`
	// определена функция **Fn**, возвращающая
	// целую часть от деления значения
	// параметра **x** типа **float** на 7.

Как следует из приведенных примеров, объявления переменных предопределенных типов **int** и **float** отличаются от соответствующих определений только отсутствием инициализаторов. Это различие не столь значимое, поскольку инициализацию можно заменить операцией присваивания. Для типов, создаваемых пользователем (в примере это структура **SW**) и функций (в примере это функция **Fn**), наличие определения обязательно и это определение должно быть только одно.

Необходимость введения объявлений вызвана тем, что они, как правило, короче определений, а информации в них достаточно, чтобы встроить объект в программу. Лаконичные объявления размещают впереди программы, обеспечивая предварительный уровень описания участвующих в ней объектов.

8.12.1 Объявление `typedef`

Директива **`typedef`** используется для создания ***нового имени типа***.

Например:

```
typedef unsigned long UINT;

typedef float VECT[3];
```

После отработки этой директивы в программу будут введены новые имена типов **`UINT`** и **`VECT`**. Теперь описание вида **`UINT a, a1;`** будет эквивалентно описанию **`unsigned long a, a1,`** а описание **`VECT C1,C2;`** будет эквивалентно директиве **`float C1[3],C2[3].`** Директива **`typedef`** *не создает новый тип*, типы объектов в примерах как были **`unsigned long`** и массив **`float`**, так и остались, сохранилась и интерпретация операций.

8.13 Выражения

Выражением называется последовательность операций, операндов и разделителей, задающих определенное вычисление. В выражениях образуются новые значения и изменяются значения переменных.

Выражения C++, подобно объектам, *имеют тип и значение* интерпретируемое в соответствии с этим типом. Примеры выражений:

```
(a + d)/c - dd*k      — арифметическое выражение;
!(fn && Rd) || tp     — логическое выражение;
GG > T                ⎫
H <= y                ⎬ — выражения отношения;
a == b                ⎪
c != d                ⎭

F = d                 — выражение присваивания;
a & b | c<<3          — выражение с поразрядными операциями;
cos(x)                — выражение вызова функции;
a                     — выражение — значение объекта.
```

Тип выражения зависит от типов операндов и смысла операций с этими типами.

8.13.1 Арифметические выражения

Арифметические выражения внешне похожи на обычные алгебраические формулы. Приоритет и порядок выполнения арифметических операций тот же, что и в алгебре. Высшим приоритетом пользуются *мультипликативные* операции: умножение * и деление /, сюда же относится и специальная операция нахождения остатка от деления целых чисел %. Низшим приоритетом обладают *аддитивные* операции: сложение + и вычитание -. Выполнение операций одинакового приоритета осуществляется слева направо в порядке их записи в выражении. При необходимости изменить порядок выполнения операций в выражениях применяют круглые скобки.

Следует сделать замечание об особенности выполнения операции деления операндов целого типа. *Если в выражении A/B оба операнда целого типа, то значение выражения будет целой частью от деления A на B.* Отметим, что на PASCAL такая операция недопустима.

Арифметические выражения с операндами различных преопределенных типов вычисляются с приведением всех операндов к «старшему» типу. Один арифметический тип считается *старшим* по отношению к другому, если его диапазон представляемых им значений полностью включает в себя диапазон младшего типа. В этом случае преобразование к старшему типу выполняется без потери информации и производится компилятором автоматически. Например, при **`double D; int z; char cc`**; выражение **`D+z-cc`** будет иметь тип **`double`**.

8.13.2 Выражения отношения и логические выражения

Аналогичное приведение к старшему типу производиться в выражениях отношения: сначала операнды приводятся к старшему типу и лишь затем производится сравнение. По логике вещей, тип выражений отношения и логических выражений должен быть

логическим с двумя значениями «истина»/»ложь». Но на C++ нет специального предопределенного типа для логических данных.

Логическое значение объекта любого типа интерпретируется как «ложь», если все биты в его битовом представлении уставлены в 0 *и, соответственно, «истина», если хотя бы один бит представления находится в состоянии* 1.

Это правило действует во всех случаях, когда по контексту требуется логическое «значение» объекта или выражения. Логические выражения и выражения отношения имеют тип **`int`**, при этом состояние «истина» соответствует значению единица, «ложь» — нуль.

8.13.3 Выражения с поразрядными операциями

Поразрядные операции характерны для программирования на низком уровне, их присутствие в языках Си и C++ дает возможность работать непосредственно с битовыми последовательностями. Это особенно важно при современной тенденции готовить программы для микроконтроллеров на языке высокого уровня.

Рассмотрим примеры выражений с поразрядными операциями. Пусть имеются два объекта беззнакового целого типа, инициализированные некоторым кодом:

```
unsigned a = 0xA7F2, b = 0x70D3;
```

Формирование значений выражений с поразрядными операциями с участием a и b легко понять из приводимых ниже таблиц.

Поразрядное «И»: `a&b`

Выражение	Значение		
	двоичное	шестнадцатеричное	десятичное
`a`	**`1010011111110010`**	**`A7F2`**	**`42994`**
`b`	**`0111000011010011`**	**`70D3`**	**`28883`**
`a&b`	**`0010000011010010`**	**`20D2`**	**`8402`**

Поразрядное «ИЛИ»: `a|b`

Выражение	Значение		
	двоичное	шестнадцатеричное	десятичное
`a`	`1010011111110010`	`A7F2`	`42994`
`b`	`0111000011010011`	`70D3`	`28883`
`a|b`	`1111011111110011`	`F7F3`	`63475`

Поразрядное «ИСКЛЮЧАЮЩЕЕ ИЛИ»: `a^b`

Выражение	Значение		
	двоичное	шестнадцатеричное	десятичное
`a`	`1010011111110010`	`A7F2`	`42994`
`b`	`0111000011010011`	`70D3`	`28883`
`a^b`	`1101011100100001`	`D721`	`55073`

Поразрядное «НЕ»: `~a` (инверсия кода)

Выражение	Значение		
	двоичное	шестнадцатеричное	десятичное
`a`	`1010011111110010`	`A7F2`	`42994`
`~a`	`0101100000001101`	`580D`	`22541`

Как следует из приведенных примеров, действие поразрядных операций сводится к применению логических операций к `i`-му разряду операндов и записи результата в `i`-й разряд результата.

Рассмотрим пример поразрядного сдвига кода беззнакового целого, представленный следующими таблицами:

Поразрядной сдвиг кода вправо на k разрядов `a>>k`

Выражение	Значение		
	двоичное	шестнадцатеричное	десятичное
`a>>0`	`1010011111110010`	`A7F2`	`42994`
`a>>1`	`0101001111111001`	`53F9`	`21497`
`a>>2`	`0010100111111100`	`29FC`	`10748`
`a>>3`	`0001010011111110`	`14FE`	`5374`
`a>>13`	`0000000000000101`	`5`	`5`
`a>>14`	`0000000000000010`	`2`	`2`
`a>>15`	`0000000000000001`	`1`	`1`

Поразрядной сдвиг кода влево на k разрядов a<<k

Выражение	Значение		
	двоичное	шестнадцатеричное	десятичное
a<<0	**1010011111110010**	**A7F2**	**42994**
a<<1	**0100111111100100**	**4FE4**	**20452**
a<<2	**100111111100100**	**9FC8**	**40904**
a<<3	**0011111110010000**	**3F90**	**16272**
a<<13	**0100000000000000**	**4000**	**16384**
a<<14	**1000000000000000**	**8000**	**32768**
a<<15	**000000000000000**	**0**	**0**

Сдвиг кода аргумента знакового типа при его отрицательном значении происходит с восстановлением единицы в знаковом разряде. Пусть объявлено:

```
int a = A7F2;
```

Тогда поразрядной сдвиг кода отрицательного числа вправо на k разрядов (значение выражения **a>>k**) будет иметь вид:

Выражение	Значение		
	двоичное	шестнадцатеричное	десятичное
a>>0	**1010011111110010**	**A7F2**	**-22542**
a>>1	**1101001111111001**	**D3F9**	**-11271**
a>>2	**1110100111111100**	**E9FC**	**-5636**
a>>3	**1111010011111110**	**F4FE**	**-2818**
a>>13	**1111111111111101**	**FFFD**	**-3**
a>>14	**1111111111111110**	**FFFE**	**-2**
a>>15	**1111111111111111**	**FFFF**	**-1**

Применение выражений с поразрядными операциями на C++ позволяет моделировать работу цифровых устройств на уровне регистров.

8.13.4 Выражения присваивания

В Си и C++ есть выражение присваивания, а не только оператор присваивания, как, например в PASCAL. Это «полноправное» выражение, имеющее тип и значение. Тип и значение выражения присваивания совпадает с типом и значением объекта, расположенным в левой части выражения. Например, если **int**

I и **float f =3.78**, то выражение **I=f**, будет иметь тип **int** и значение **3**.

Наличие у выражения присваивания значения дает возможность употреблять его в несколько неожиданном контексте; например, **x=1.5*b+(a=3*x)**. Возможна организация цепочек: **A=b=f**, что означает присвоение значения объекта **f** сначала объекту **b**, а затем значение объекта **b** объекту **A**.

Еще одной особенностью операции присваивания C++ является то, что она может быть составной, т.е. совмещаться с большинством бинарных операций. Например, **x[i+3]*=4** означает **x[i+3]=x[i+3]*4**, за исключением того факта, что выражение **x[i+3]** вычисляется только один раз.

8.13.5 Выражения инкремента и декремента

На C++ существуют унарные операции инкремента (увеличения) и декремента (уменьшения) операндов. Выражения инкремента и декремента могут иметь ***постфиксную*** и ***префиксную*** формы. Эти операции могут применятся как операндам арифметического типа, так указателям. Пример действия операций для арифметических типов приведен в следующих таблицах.

ИНКРЕМЕНТ

Постфиксная форма: **I++**			Префиксная форма: **++I**		
значение операнда до операции	значение выражения	значение операнда после операции	значение операнда до операции	значение выражения	значение операнда после операции
Тип операнда целый					
10	10	11	10	11	11
Тип операнда плавающий					
0.95	0.95	1.95	0.95	1.95	1.95

ДЕКРЕМЕНТ

Постфиксная форма: **I--**			Префиксная форма: **--I**		
значение операнда до операции	значение выражения	значение операнда после операции	значение операнда до операции	значение выражения	значение операнда после операции
Тип операнда целый					
10	10	9	10	9	9
Тип операнда плавающий					
0.95	0.95	–0.05	0.95	–0.05	–0.05

Из приведенных примеров следует, что в применении к арифметическим типам операции инкремента и декремента вызывают соответственно увеличение и уменьшение значения операнда на единицу.

Постфиксная и префиксная формы операций отличаются значением самого выражения: в постфиксной форме значение выражения совпадает со значением операнда *до* операции, в префиксной — со значением операнда *после* операции.

С указателями операции инкремента и декремента выполняются в соответствии с правилами адресной арифметики, рассмотренной выше.

8.14 Операторы

Исходный текст любой программы, в том числе и программы на C++, состоит из набора директив — *операторов*, предписывающих компьютеру выполнить какое то ни было действие.

Оператор программы — законченная инструкция на исходном языке. Оператор в программе по логической завершенности аналогичен предложению на естественном языке. После компиляции оператор преобразуется в поток команд, переводящих вычислительную систему из одного состояния в другое. По завершении выполнения оператора внутреннее состояния вычислительной среды (состояние регистров процессора, регистров общего назначения, состояние стека и пр.) возвращается некоторому «стандартному» состоянию. Любой оператор C++ должен заканчиваться разделителем; — точка с запятой.

Операторы C++ можно разделить на три группы, это

- операторы описания;
- операторы выражения;
- операторы управления.

К группе операторов описания относятся рассмотренные выше директивы объявления и определения объектов программы с инициализацией данных или без таковой.

Операторы — выражения представляют наиболее многочисленную группу операторов. Именно операторы — выражения вызывают преобразование значений переменных, осуществляют вызов функций и осуществляют доступ к объектам.

Операторы управления позволяют разветвлять вычислительный процесс в зависимости от условий, выполнять итерации (циклы) в программе и т.п.

8.14.1 Пустой оператор

На языке C++ возможна запись пустого оператора, т.е. такого оператора, с которым не связаны никакие действия. Синтаксис записи пустого оператора — это просто точка с запятой:

```
;   // пустой оператор
```

Пустой оператор — это синтаксически полноправный оператор, он может использоваться в качестве своего рода заглушки: там, где по соображениям синтаксиса должен находиться оператор, но никаких действий от программы не требуется.

Замечание. Неуместное использование знака; воспринимаемого компилятором как пустой оператор, может приводить к семантическим ошибкам в программе.

8.14.2 Операторы описания

Оператор описания вводит в программу объект, связывая имя типа с идентификатором объекта. Таким образом, с помощью операторов описания реализуются объявления объектов. На C++ операторы описания имеют не только дескриптивный, но и конструктивный аспект: их действие вызывает создание объектов. Примеры операторов описания даны в п. 2.9.

8.14.3 Операторы-выражения

Самый обычный вид оператора — оператор-выражение. Он состоит из выражения, за которым следует точка с запятой. Например:

```
a = b*3+c; // оператор присваивания с арифметическим  выражением
lseek(fd,0,2); // оператор вызова функции
```

8.14.4 Составные операторы (блоки)

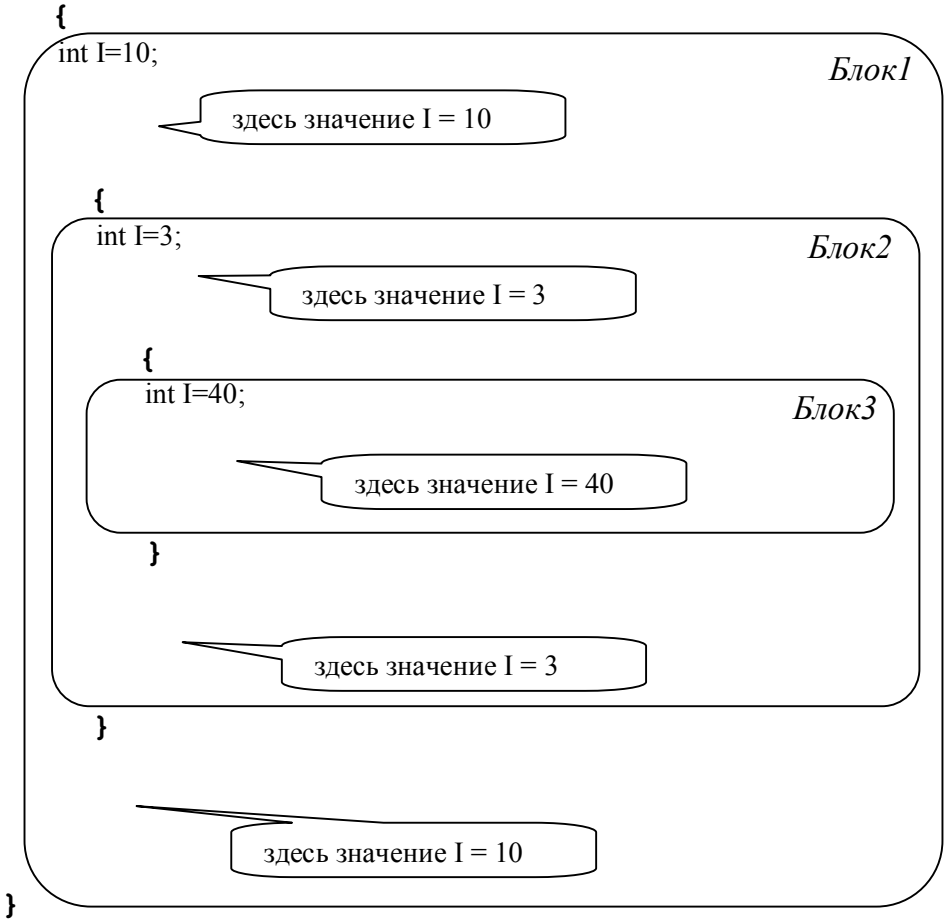
Составной оператор или блок — это последовательность операторов, заключенных в фигурные скобки:

```
{           // начало блока
  a=b+2;
  . . .     // еще операторы
  b++;
 }          // конец блока
```

Синтаксически блок рассматривается как один оператор.

Если в составе операторов блока встречаются операторы описания, то вводимые ими объекты локальны и имеют *область видимости* (см. ниже) только в пределах данного блока. В случае, если имя локальной переменной совпадает с именем переменной, описанной в охватывающем блоке, то происходит т.н. *маскировка имени* — внутри блока имя будет ссылаться *только* на локальную переменную. Рассмотрим пример.

Пусть имеется три вложенных друг в друга блока и в каждом из них объявлена переменная **`I`**:



Таким образом, операторы описания, содержащиеся внутри блока, создают локальный объект внутри этого блока. В случае, если имя локального объекта совпадает с именем объекта во внешнем охватывающем блоке, то имя локального объекта маскирует имя внешнего объекта. Образно говоря, на имя внутри блока «откликается» ближайший объект. Более подробно явления маскировки будут обсуждаться в разделе «Области видимости и время жизни объектов».

8.15 Операторы ветвления

8.15.1 Операторы `if` и `if/else`

Оператор **`if`** предназначен для реализации в программе следующей структуры:

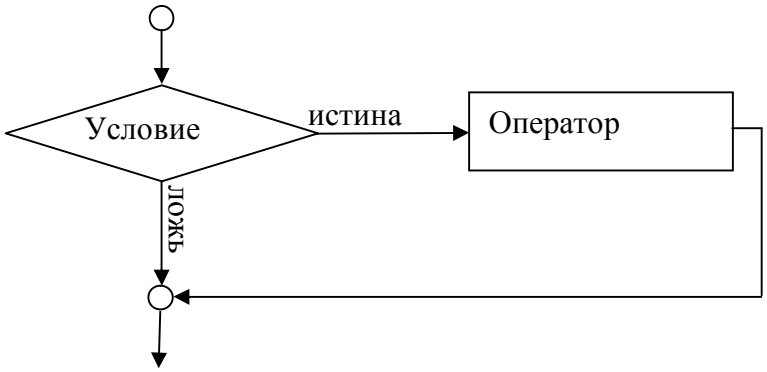


Рис 8.1 — Логическая структура оператора **`if`**

Оператор исполняется, если условие истинно, иначе он пропускается. Синтаксис записи условного оператора:

`if (`*логическое выражение***`)`** *исполняемый оператор***`;`** …

Например:

```
if(a>5) a=5;
```

```
if(t) t=1/t;  // если t ≠ 0 значение t меняется на обратное
```

В последнем примере использована логическая интерпретация значения переменной арифметического типа.

Условный оператор состоит из заголовка, образованного зарезервированным словом **if** и последующим условным выражением в круглых скобках. За заголовком следует исполняемый оператор.

Исполняемый оператор может быть блоком, содержащем сколь угодно много операторов. При истинности условия будет выполняться весь этот блок.

Внимание! Не ставьте после заголовка оператора точку с запятой, это приведет к семантической ошибке: при истинности условия будет «выполняться» пустой оператор.

Условный оператор имеет альтернативную форму **if/else**, реализующую следующую алгоритмическую структуру:

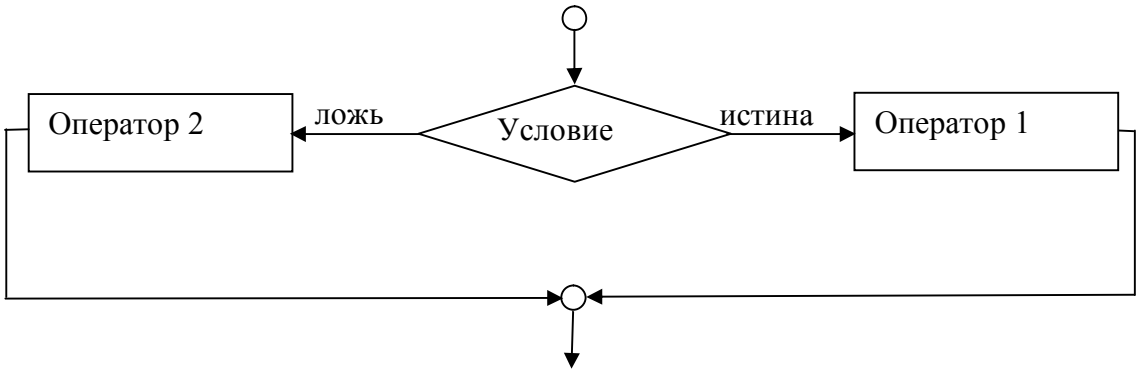


Рис. 8.3 — Логическая структура оператора **if\else**

В отличие от оператора **if**, рассмотренного выше, оператор **if\else** производит альтернативный выбор: если условие истинно, выполняются действия оператора 1, иначе — действия оператора 2.

Синтаксис записи оператора на языке C++:

if(*логическое выражение*) *оператор1*; **else** *оператор2*;

Пример использования:

```
if(f<0) c=-1;  else c=1;
if(g) h='1'; else h='0';
```

Здесь также, как и в операторе **if** исполняемые операторы 1 и 2 могут быть составными и представлять собой блоки.

Условные операторы могут быть вложенными друг в друга и глубина вложения теоретически не ограничена. Это обстоятельство создает определенные неудобства при чтении текста программы с многократным вложением альтернативных операторов: последовательность перемежающихся **`if`** и **`else`** затеняет структурность текста, так что становиться трудно определить к какому же **`if`** относиться тот или иной **`else`**. Во избежание подобной путаницы не следует употреблять глубоко вложенные конструкции альтернативных операторов.

8.15.2 Операторы `switch`

Оператор многоальтернативного выбора **`switch`** предназначен для реализации в программе структуры ветвления следующего вида:

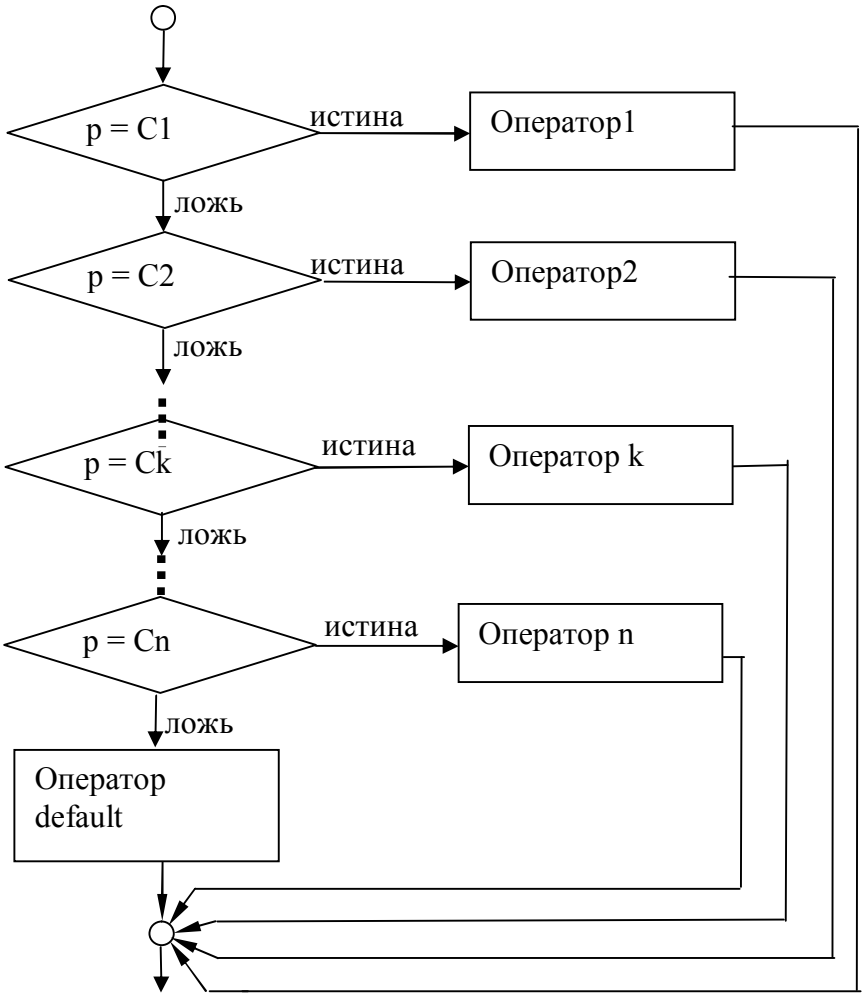


Рис. 8.4 — Логическая структура оператора **`switsh`**

Здесь **p** — параметр оператора, значение которого сравнивается множеством констант **C1, C2,..Cn**. При равенстве значения **p** какой-либо из констант вычислительный процесс направляется на выполнение соответствующего оператора (блока операторов). В случае, когда значение p оказывается отличным от всего набора констант исполняется оператор по умолчанию (оператор default)

```
switch (выражение для значения параметра)
{
case C1: {оператор 1; break;}
case C2: {оператор 2; break;}
    ...
case Ck: {оператор k; break;}
    ...
case Cn: {оператор n; break;}
default :оператор default;

}
```

Тело оператора **switch** представляет собой блок, охватывающий все альтернативные ветви вычислительного процесса. Каждая из альтернативных ветвей тоже представляет собой блок, заканчивающийся оператором **break.**

Операторы **break** применяются для выхода из оператора **switch**. Дело в том, **case Ck** синтаксически представляют собой метки. При отсутствии оператора **break** вычислительный процесс, попав, например, на метку **case Ck**, пойдет далее по порядку следования операторов, невзирая на встречающиеся метки, и после выполнения оператора **k** произойдет выполнение всех операторов, расположенных ниже.

Константы в вариантах **case** должны быть различными, и если проверяемое значение не совпадает ни с одной из констант, выбирается вариант **default**. Можно не предусматривать вариант **default**, в этом случае вычислительный процесс покинет пределы оператора **switch**, ничего не исполнив.

Пример использования оператора **switch:**

```
switch (i)
{
case 0: {cout<<"тишина"; break;}
case 1: {cout<<"соло"; break;}
case 2: {cout<<"дуэт"; break;}
case 3: {cout<<"трио"; break;}
case 4: {cout<<"квартет"; break;}
default :cout<< "ни то, ни се";
}
```

Обратим внимание, что в ветви последней альтернативы оператор **break** можно не ставить т.к. после ее выполнения вычислительный процесс сам выйдет за пределы оператора **switch.**

8.16 Операторы итерации

8.16.1 Операторы `while` и `do while`

Операторы итерации **while** и **do while** предназначены для создания в программе циклических структур следующего вида:

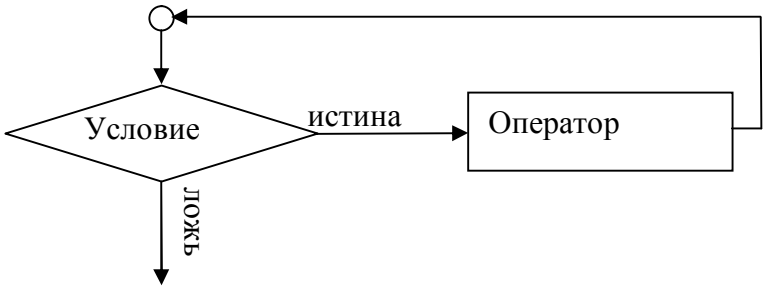


Рис. 8.5 — Структура **while**

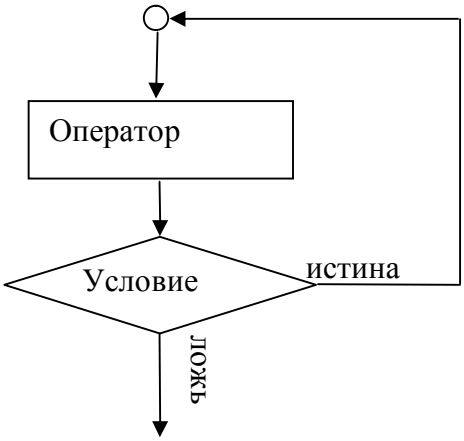


Рис. 8.6 — Структура **do while**

Циклические структуры **while** и **do while** обе реализуют циклы типа «*пока*». Вычислительный процесс циклит, пока условие истинно. Различие между ними лишь в точке проверки условия продолжения цикла: в структуре **while** это условие проверяется *до* исполнения оператора в теле цикла, в структуре **do while** — *после* исполнения этого оператора. Поэтому структуру **while** называют циклом *с предусловием*, а структуру **do while** — циклом с *постусловием*.

Синтаксис записи оператора **while**:

while(*условное выражение*) *оператор*;

Синтаксис записи оператора **do while:**

do *оператор*; **while (***условное выражение***);**

В операторе **do while** зарезервированное слово **do** ставиться перед оператором, образующим тело цикла. Между **do** и **while** может располагаться только один оператор, если требуется поместить в тело цикла более одного оператора, то их следует объединить в блок.

Примеры записи операторов:

```
w while(c=fgetch(F)!=EOF)C[i++]=c; // чтение в массив С
                                   // символов из файлового потока,
                                   // пока не будет достигнут конец файла

do {c='A'; cout<<c++;} while(c<256);// вывод
                                   // последовательности символов ASCII,
                                   // начиная с кода 'A' и выше.
```

Оператор выборки символов из файла из первого примера можно записать короче, исключив оператор тела цикла и заменив его пустым оператором:

```
while((C[i++]=fgetch(F))!=EOF);
```

При этом отпадает необходимость в дополнительной буферной переменной **c,** но при такой форме записи в последний элемент массива попадет символ конца файла **EOF**.

8.16.2 Оператор `for`

Оператор **`for`** предназначен для создания в программе следующей циклической структуры:

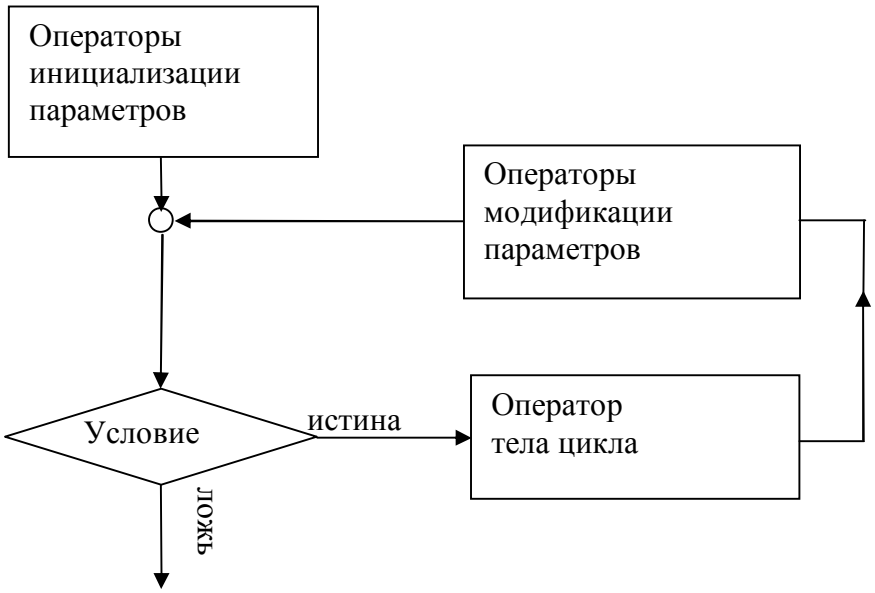


Рис. 8.7 —Логическая.структура оператора **`for`**

Вычислительный процесс, соответствующий этой структуре в информационном плане может быть более насыщен, чем просто цикл со счетчиком. В его составе могут быть не только одиночные операторы инициализации счетчика, но и группы операторов инициализации для набора параметров. Такие же группы операторов могут изменять значения параметров после каждого выполнения тела цикла.

Синтаксис оператора **`for`**:

`for(`*список операторов инициализации* **`;`** *условие продолжения цикла*; *список операторов модификации***`)`** *оператор тела цикла*;

Списки операторов инициализации и модификации — это перечисление их выражений через запятую.

Примеры:

```
for(i=0; i<10; i++) D[i]=i; // простейший цикл
                            // со счетчиком итераций
```

```
for(i=0,a=0.17;  i<10;  i++,a+=0.5)  D[i]=a;// цикл с
                                    //двумя выражениями в инициализирующей
                                    // и модифицирующей частях. Переменные i
                                    // и a должны быть объявлены ранее.

for(int  i=0,k=-17;  i<10;  i++,k+=5)  D[i]=a;// цикл с
                                      //объявлением и инициализацией двух
                                      // переменных в инициализирующей
                                      // части оператора for.
```

В последнем примере в заголовке оператора **for** введены две переменные **i** и **k**. В версии Borland C++ 3.1 область видимости этих переменных распространяется до конца внешнего блока, охватывающего оператор **for**. В более старших версиях C++ область видимости переменных, введенных в инициализирующей части оператора **for** *ограничена заголовком и телом цикла*.

На основе структуры **for** можно получать многообразные программные реализации. В литературе даже высказывается мнение, что любую программу можно представить в виде логической структуры цикла **for**. На самом деле, эта логическая конструкция имеет все необходимые компоненты типичной моделирующей программы — блок формирования начальных значений, блок проверки факта достижения цели и блок смены условий моделирования перед повторным просчетом модели.

В ряде ситуаций для реализации поставленной цели может оказаться достаточным лишь тех операций, что содержатся в заголовке цикла, тогда отпадает необходимость в операторе тела цикла, и на его место можно поместить пустой оператор. Например, заполнение 10 элементов массива **S** начальным фрагментом геометрической прогрессии s0, s0·q, s0·q^2, … можно выполнить с помощью следующего оператора **for**:

```
for (s[0]=s0, i = 1; i<10; s[i]=s[i-1]*=q,i++);
```

В этом примере оператор тела цикла — пустой, т.к. операторы в заголовке цикла решают в полной мере решают поставленную задачу, так что в теле цикла уже ничего делать не нужно.

8.17 Операторы управления

8.17.1 Оператор `break`

Этот оператор предназначен для использования в структурах повторения (циклах), и в структуре многоальтернативного выбора, реализуемой оператором **`switch.`** В том и другом случае назначение оператора **`break`** одинаковое — немедленное завершение выполнения оператора, в теле которого он расположен.

Действие **`break`** в операторе **`switch`** было рассмотрено выше. В циклах оператор **`break`** обычно используется совместно с условным оператором if и служит для прерывания цикла при истинности некоторого условного выражения. Оператор **`break`** удобен для прерывания по определенному условию т.н. «вечных» циклов специально зацикливающих программу в ожидании наступления какого-то внешнего события. Например:

```
while(1)
{ int c = getch();
 if(c==27) break;
cprintf("%c",c);
}
```

Этот фрагмент программы будет постоянно запрашивать ввода символа с клавиатуры и выводить полученный символ на экран, до тех пор, пока не будет введен символ с кодом 27 (код клавиши Esc), после чего цикл прервется.

8.17.2 Оператор `continue`

Оператор **`continue`** используется в циклах для возврата к началу цикла, минуя оставшиеся в теле цикла операторы. Пример:

```
for(int i=0; i<n-1; i++)
{ if(M[i]<M[i+1]) continue;
int T=M[i];
M[i]=M[i+1];
```

```
M[i+1]=T;
}
```

Этот фрагмент программы просматривает массив из n элементов и если предыдущий элемент окажется больше последующего меняет их местами. Исключение трех операторов, осуществляющих обмен, выполнено с использованием оператора **continue**.

Оператор **continue** применяется редко, т.к. можно легко заменить другими операторами, причем программа будет более структурной. Так, в приведенном примере вполне можно было обойтись и без оператора **continue**:

```
for(int i=0; i<n-1; i++)
{ if(M[i]>M[i+1])
  { int T=M[i];
   M[i]=M[i+1];
   M[i+1]=T;}
  }
}
```

Этот фрагмент кода делает то же самое, но выглядит более структурировано.

8.17.3 Оператор `return`

Оператор **return** предназначен для завершения выполнения функции и возврата в вызывающую программу. Оператор является обязательным, если функция объявлена с возвращаемым типом, отличным от **void.** В этом случае оператор **return** записывается так:

return <*выражение*>;

где < *выражение* > должно иметь тот же тип, что и тип возвращаемого значения в объявлении функции. Значение этого выражения и будет возвращено в вызывающую программу в качестве результата работы функции.

Функция может содержать несколько операторов **return**, находящихся на различных ветвях ее вычислительного процесса, но в любом случае это последний исполняемый оператор в теле функции. Не будет ошибкой, если оператор **return** поместить в функцию с объявленным типом возвращаемого значения **void**. И в этом случае оператор **return** выполнит свою задачу по завершению работы функции, но при этом следует записывать этот оператор без возвращаемого значения:

```
return;
```

Поскольку на Си и C++ головная программа — тоже функция, то использовав в ней оператор **return** в соответствующей форме, можно завершить выполнение всей программы.

8.17.4 Оператор `goto`

Этот оператор выполняет безусловную передачу управления на метку. Синтаксис его записи:

```
goto AAA;
```

где AAA — имя метки.

Оператор **goto** относится к виду неструктурных операторов, он достался в наследство от прежних технологий программирования и в современной практике практически не применяется.

Дело в том, что с помощью этого оператора легко нарушается структурность программы: можно запросто войти внутрь цикла, минуя его заголовок, можно передать управление внутрь блока условного оператора без анализа истинности соответствующего ему условия и т.п. Даже правильно написанная программа с операторами **goto** чрезвычайно трудно читается и анализируется.

Итак, существует достаточно веских причин не использовать этот оператор, но поскольку он присутствует в синтаксисе языка, рассмотрим его работу на примере. Пусть имеется фрагмент прораммы:

```
    int i;
beg : i = getch();
      if(i==27) goto fin;
      cprintf("%c",i);
    goto beg;
    fin : <продолжение программы>
```

Этот фрагмент программы выполняет тот же бесконечный цикл ввода символов с клавиатуры, прерываемый нажатием клавиши Esc, что и в примере с бесконечным циклом **while**, но реализованный другим способом. Здесь использованы две метки с именами **beg** и **fin**. Первая метка отмечает точку начала цикла, вторая — первый исполняемый оператор за циклом, ту точку, куда передается управление при прерывании цикла.

Метки на C++ это особые лексемы, состоящие, подобно идентификаторам, из последовательности букв и цифр. Значение метки — это ее имя. Метка отделяется от помечаемого оператора двоеточием. Наличие метки у оператора никак не сказывается на его выполнении.

Несмотря на кажущееся удобство в приведенном простом примере, использование операторов **goto** и меток в сколько-нибудь сложных программах нежелательно, лучше воспользоваться структурными операторами.

9 ФУНКЦИИ

Языки Си и C++ иногда называют языками функций — столь велика в них их роль. Функции позволяют полной мере реализовать модульную технологию программирования. Модули — подпрограммы обычно выделяют по принципу функциональной декомпозиции, и каждый из них решает свою логически законченную задачу. Инструментом создания обособленных модулей на Си и C++ служат только функции, в отличие от PASCAL процедур здесь нет. В программах составленных на этих языках отсутствует и синтаксически выделенная головная программа, ее роль также играет функция, единственной особенностью которой является зарезервированное за ней имя **`main`**.

Ни на Си, ни на C++ нет специальных операторов ввода \ вывода, подобных паскалевским WRITE и READ, операции обмена осуществляются с помощью функций.

Более или менее сложную программу целесообразно разбить на модули, которые можно редактировать и отлаживать независимо друг от друга, так что лучший способ разработки и сопровождения большой программы — разделение ее на несколько модулей, каждый из которых более управляем, чем исходная программа. Модули пишутся на C++ в виде функций и классов. Классы мы рассмотрим позже, а сейчас займемся функциями.

9.1 Структура заголовка функции

Функция — это именованная подпрограмма, к которой можно обращаться из других частей программы столько раз, сколько потребуется. Синтаксически функция C++ представляет собой обособленный именованный блок с *интерфейсом*, задающим формат обмена информацией с вызывающей программой. Передача информации, минуя интерфейс, например, посредством «глобальных» переменных, хотя и допустима, но крайне нежелательна т.к. нарушает автономность подпрограммы.

Цель скрытия информации в функциях заключается в том, чтобы дать доступ только к той информации, которая нужна для выполнения их задач. Это средство реализации принципа наи-

меньших привилегий, одного из наиболее важных принципов разработки хорошего программного обеспечения.

Описание интерфейса содержится в заголовке функции, который имеет следующую структуру:

тип_возвращаемого_значения Имя_функции (список_параметров)

Результат работы функции обычно возвращается в вызывающую ее программу в виде значения, которое ассоциируется с именем функции точно так же, как значение переменной ассоциируется с ее именем — идентификатором. Какой тип значения будет возвращать функция, указывает первый по порядку элемент заголовка — *тип_возвращаемого_значения.*

Имя функции — обычный идентификатор **C**++, и назначение имени функции схоже с назначением идентификатора переменной — с ним связывается адрес объекта, в данном случае это адрес точки входа в подпрограмму.

Список параметров — это описание информационного входа функции. В списке через запятую указываются типы аргументов и их имена, задается количество аргументов и порядок их следования при вызове функции.

9.2 Логический механизм вызова функций

Код программы, вызывающей функцию, и код ее реализующий обычно разнесены в адресном пространстве. Поэтому при вызове функции производится скачок по адресному пространству — передача управления на точку входа в подпрограмму. Тот адрес, куда надо «прыгнуть» связывается с именем функции. До совершения скачка требуется куда-то временно поместить значения входных параметров с тем, чтобы их использовать в теле подпрограммы.

Компиляторы C++ строят исполняемую программу так, что в качестве временного хранилища используется программный стек. Перед передачей управления стек загружается значениями параметров в строгом соответствии с той последовательностью и типами, что указаны с списке параметров в заголовке функции.

После завершения работы подпрограммы в освободившийся к тому времени стек загружается возвращаемое значение тоже в

соответствии с типом, указанным в заголовке функции, и совершается обратный скачок в вызывающую программу.

В вызывающей программе происходит выгрузка из стека возвращенного значения (в соответствии объявленным типом!) и вычислительный процесс продолжается. Цепочка действий при вызове функции показана на рисунке 8.8.

Рис. 8.8 — Схема вызова функции

Если функция выполняет набор каких-либо простых действий, ее исполняемый код имеет небольшой размер, не содержит структур ветвления и итераций, то «накладные расходы», связанные с проходом вычислительного процесса по петле, становятся соизмеримыми с вычислительными затратами на исполнение самого тела функции, а, зачастую, даже и превосходить их. В этом случае целесообразно не вызывать функцию по обычной схеме, а встроить ее исполняемый код в вызывающую программу прямо

за точкой вызова. Функции, вызов которых организован по такому принципу, называют встраиваемыми или инлайновыми (*inline*) функциями.

Использование *inline* — функций экономит время вызова, жертвуя при этом объемом памяти, т.к. в каждой точке вызова требуется разместить исполняемый код вызываемой функции.

Компилятор предупреждается о том, что функцию следует вызывать как инлайновую зарезервированным словом **inline,** записываемым перед заголовком функции, например:

```
inline int GetX();
inline double SQR(double x){return x*x ;}
```

Не всякую функцию компилятор может реализовать как inline. Если в теле функции есть операторы ветвления и циклы, то у компилятора «не хватает интеллекта» построить ее встраиваемый код. Поэтому директива **inline** перед заголовком функции носит рекомендательный характер, в случае, если построение функции, помеченной программистом как **inline** по встраиваемой схеме невозможно, компилятор строит ее вызов по обычной схеме, выдавая пользователю соответствующее предупреждение.

9.3 Объявление, определение и вызов функции

С функцией на языке C++ связано три понятия:

- объявление функции;
- определение функции;
- вызов функции.

9.3.1 Объявление функции

Функции могут быть самостоятельными единицами трансляции, т.е. способны компилироваться независимо друг от друга. Такая независимость потребовала дополнительных мер по информированию компилятора о структуре интерфейсов вызываемых функций. Роль такого «информатора» отведена *объявлению (прототипу)* функции. Объявление функции должно предшествовать ее первому вызову.

Объявление (или прототип) функции на C++ выглядит следующим образом:

тип_возвращаемого_значения Имя_функции
(список_типов_параметров);

Таким образом, прототип функции — это ее заголовок, заканчивающийся точкой с запятой. Отметим, что в прототипе можно не указывать имена параметров, достаточно указать лишь их типы.

Объявления, представляющего собой описание интерфейса функции, достаточно для согласования формата обмена информацией с вызывающей программой. В самом деле, список типов входных параметров указывает, чем должен загружаться стек при вызове данной функции, и в какой последовательности это следует делать. Тип возвращаемого значения предписывает, как следует интерпретировать содержимое стека после возврата из функции.

Примеры объявлений функций:

```
int IJmaxR(int,int,double*,int*,int*);

int G_MWel(int,double*,double&,double*);

void FF (int);

int RR().
```

В приведенных примерах функции с именами **IJmaxR** и **G_MWel,** возвращают целочисленные значения. Функция **IJmaxR** принимает при вызове пять аргументов в соответствии с указанной последовательностью типов, функция **G_MWel** — четыре.

Указание специального типа **void** в качестве типа возвращаемого значения — это предупреждение компилятору о том, что по завершении работы функция не будет возвращать результат через свое имя, и выгружать что-либо из стека по ее окончании не следует. Функции, объявленные с возвращаемым типом **void** – это аналоги паскалевских процедур.

Таким образом, функция **FF** не возвращает ничего, принимает один параметр типа **int**. Функция **RR** объявлена с пустым

списком параметров, это означает, что она не принимает никаких параметров.

На C++ имеется возможность задавать значения аргументов функции по умолчанию, для этого в списке параметров нужно указать какие из параметров могут быть заданы по умолчанию и какое при этом им присвоится значение. Синтаксис записи списка параметров при этом таков:

(тип_1 имя1, тип2 имя2, тип3 имя3= значение_по_умолчанию)

Здесь показан вид списка параметров функции, принимающей три параметра. Последний по порядку списка параметр может быть инициализирован значением по умолчанию, указанным справа от знака = .

Механизм присвоения по умолчанию запускается, если аргумент по умолчанию пропускается при вызове функции. Только в этом случае используется его значение по умолчанию.

Аргументы по умолчанию должны быть крайними правыми (последними) в списке параметров функции. Аргументы по умолчанию должны быть указаны при первом же упоминании имени функции. Значения по умолчанию могут быть константами, глобальными переменными или вызовами функций.

9.3.2 Определение функций

В общем виде определение функции на C++ выглядит следующим образом:

тип_возвращаемого_значения имя_функции (список_ параметров)

{

тело функции: директивы программы

}

Примеры определений функции:

```
int power(int a, int b)
{ int s=1;
  for(int i=0; i<b; i++)
  s*=a;
  return s;
```

```
}

double SUMMA(int n, double* D)
{ double s=0;
for(int i;i<n; i++)s+=D[i];
return s;
}
```

Функция первого примера возводит значение целочисленной переменной, задаваемой в качестве первого аргумента, в целочисленную степень, определяемую значением второго аргумента.

Функция второго примера вычисляет сумму **n** значений элементов массива **D**, состоящего из компонентов типа **double**. При вызове функции величина **n**, передается первым параметром типа **int**, массив передается вторым параметром как указатель на **double***.

В определении наряду с заголовком функции, определяющим ее интерфейс, содержится и текст ее программы. Как уже упоминалось выше, функция — это именованный блок. Это блок — составной оператор, со всеми присущими ему свойствами.

Локальные переменные, введенные в описании данной функции, известны только ей, прочие функции, включая функцию **main**, этих переменных не видят и, следовательно, не могут их использовать.

Блок, содержащий тело функции не может быть помещен в какой-либо иной охватывающий блок, т.е. нельзя определить одну функцию «внутри» другой, *функции не могут быть вложенными друг в друга*.

Роль охватывающего блока для функций играет область файла, где помещены их описания. В пределах этой области, вне всяких блоков, включая функции, могут находиться операторы описания объектов. Эти т.н. *глобальные* объекты могут быть видимы всеми функциями. В принципе, обмен информацией между функциями можно организовать в помощью глобальных переменных, но как уже отмечалось этот путь нарушает принцип инкапсуляции данных и чреват негативными последствиями.

В функциях, точно так же как в блоках, может происходить маскировка имен, когда имя локальной переменной совпадает с именем глобальной переменной. В этом случае до глобальной переменной можно добраться, используя унарную операцию разрешения области действия (::), которая позволяет обеспечить доступ к глобальной переменной в случае, когда локальная переменная имеет в области действия такое же имя.

9.3.3 Вызов функций

Активизация программы функции происходит после исполнения оператора ее вызова. Оператор вызова функции — это оператор выражение. Собственно выражение вызова состоит из имени функции и следующего за ним списка аргументов в круглых скобках. Пример операторов вызова функций:

```
sqrt(x);
clrscr();
fopen("c:\\mayfile.txt","wb");
```

При вызове функции на место формальных параметров становятся фактические. Передача фактических параметров идет в строгом соответствии с порядком их следования в списке формальных параметров. При этом компилятор проверяет соответствие типов данных и их количества, тому, что было указано в списке при объявлении функции.

Пример фрагмента программы с операторами вызова функций:

```
int N=6,nn=3,II[6],JJ[6], rI, rG;// объявления данных
double A[42], detA, Xr[6];      // в вызывающей программе
//. . .
rI = IJmaxR( N, nn, A,II,JJ); // вызов функции IJmaxR
rG = G_MWel(N, nn, detA,Xr); // вызов функции G_MWel
//. . .
```

В приведенном фрагменте вызывающей программы наряду с операторами вызова функций показаны директивы объявления данных, используемых в качестве фактических параметров.

Замечание: имена формальных параметров в заголовке определения функции никак не связаны с именами фактических параметров в операторе ее вызова.

9.3.4 Передача параметров функциям

При вызове функции в стеке выделяется память для ее формальных параметров, и каждый формальный параметр инициализируется значением соответствующего фактического параметра. Фактически, в любом случае значение фактического параметра копируется в стек, и далее функция работает только с этой копией.

Существуют три способа передачи информации внутрь функции:

1) передача параметра *по значению*;

2) передача параметра *по указателю*;

3) передача параметра *по ссылке*;

Семантика передачи параметров тождественна семантике инициализации. В частности, сверяются типы формального и соответствующего ему фактического параметра, и выполняются, если необходимо, стандартные и пользовательские преобразования типа.

Рассмотрим функцию:

```
void f(int val, int* poin, int& ref)
{val++;
ref++;
*poin = 7;
}
```

При вызове f() в выражении val++ увеличивается локальная копия первого фактического параметра, тогда как в ref++ увеличивается сам третий фактический параметр. В результате операции разыменования указателя poin произойдет изменение значения того объекта, адрес которого находится в poin. Поэтому в функции

```
void g()
{int i = 1;
 int j = 1;
 int k = 1;
f(i,&j,k);
}
```

увеличатся значения j и k, а i останется неизменным. Первый параметр i передается по значению, второй параметр j передается через указатель и третий параметр k — по ссылке. В качестве второго фактического параметра при вызове f() указан адрес переменной k, определенной в вызывающей функции g().

Функции, которые изменяют свой передаваемый по ссылке параметр, труднее понять, и что поэтому лучше их избегать. Но большие объекты, очевидно, гораздо эффективнее передавать по ссылке, чем по значению. Правда можно описать параметр со спецификацией **const**, чтобы гарантировать, что передача по ссылке используется только для эффективности, и вызываемая функция не может изменить значение объекта:

```
void f(const large& arg)
{
// значение "arg" нельзя изменить без явных операций
// преобразования типа
  }
```

Если в описании параметра ссылки **const** не указано, то это рассматривается как намерение изменять передаваемый объект:

```
void g(large& arg); // считается, что в g() arg будет меняться
```

Отсюда вывод: необходимо использовать **const** всюду, где необходимо гарантированно защитить фактический параметр от воздействия вызываемой функции.

Точно так же, описание параметра, являющегося указателем, со спецификацией **const** говорит о том, что указываемый объект не будет изменяться в вызываемой функции. Например:

```
int strlen(const char*);
```

Значение такого приема растет вместе с ростом программы.

Отметим, что семантика передачи параметров отличается от семантики присваивания. Это различие существенно для параметров, являющихся const или ссылкой, а также для параметров с типом, определенным пользователем.

Литерал, константу и параметр, требующий преобразования типа, можно передавать как параметр типа const&, но без спецификации const передавать нельзя. Допуская преобразования для параметра типа const T&, мы гарантируем, что он может принимать значения из того же множества, что и параметр типа T, значение которого передается при необходимости с помощью временной переменной.

```
float fsqrt(const float&);

  void g(double d)
  {
    float r;
    r = fsqrt(2.0);  // передача ссылки на временную
                     // переменную, содержащую 2.0
    r = fsqrt(r);     // передача ссылки на r
    r = fsqrt(d);     // передача ссылки на временную
                     // переменную, содержащую float(d)
  }
```

Запрет на преобразования типа для параметров-ссылок без спецификации **const** введен для того, чтобы избежать нелепых ошибок, связанных с использованием при передаче параметров временных переменных:

```
void update(float& i);

void g(double d)
{
  float r;

  update(2.0);    // ошибка: параметр-константа
  update(r);      // нормально: передается ссылка на r
  update(d);      // ошибка: здесь нужно преобразовывать тип

}
```

Возвращаемое значение

Если функция не описана как void, она должна возвращать значение. Возвращаемое значение указывается в операторе return в теле функции.

Например:

```
int fac(int n) { return (n>1) ? n*fac(n-1) : 1; }
```

В теле функции может быть несколько операторов return:

```
int fac(int n)
{
if (n > 1) return n*fac(n-1);
else  return 1;
}
```

Подобно передаче параметров, операция возвращения значения функции эквивалентна инициализации. Считается, что оператор **`return`** инициализирует переменную, имеющую тип возвращаемого значения.

Тип выражения в операторе **`return`** сверяется с типом функции, и производятся все стандартные и пользовательские преобразования типа. Например:

```
double f()
{
 // ...
 return 1;   // неявно преобразуется в double(1)
}
```

При каждом вызове функции создается новая копия ее формальных параметров и внутренних локальных переменных. Занятая ими память после выхода из функции будет снова использоваться, поэтому неразумно возвращать указатель на локальную переменную. Содержимое памяти, на которую настроен такой указатель, может измениться непредсказуемым образом:

```
int* f()
{    int local = 1;
  return &local;    // ошибка
}
```

К счастью, компилятор предупреждает о том, что возвращается ссылка на локальную переменную.

Вот другой пример:

```
int& f() { return 1; }  // ошибка
```

Передача параметров — массивов

Если в качестве параметра функции указан массив, то передается указатель на его первый элемент. Например:

```
int strlen(const char*);
void f()
{
char v[] = "массив";
strlen(v);
strlen("Николай");
}
```

Это означает, что фактический параметр типа T[] сначала преобразуется к типу T*, и затем передается. Поэтому присваивание элементу формального параметра-массива какого-либо значения внутри тела функции изменяет этот элемент. Иными словами, *массивы отличаются от других типов тем, что они не передаются и не могут передаваться по значению*.

В вызываемой функции размер передаваемого массива неизвестен. Это неприятно, но есть несколько способов обойти данную трудность. Прежде всего, все строки оканчиваются нулевым символом, и, значит, их размер легко вычислить. Можно передавать еще один параметр, задающий размер массива. Другой способ: определить структуру, содержащую указатель на массив и размер массива, и передавать ее как параметр. Например:

```
void compute1(int* vec_ptr, int vec_size);  // 1-ый способ

struct vec {              // 2-ой способ
  int* ptr;
  int size;
};

void compute2(vec v);
```

Сложнее с многомерными массивами, но часто вместо них можно использовать массив указателей, сведя эти случаи к одномерным массивам. Например:

```
char* day[] = {
"понедельник",
"вторник",
"среда",
"четверг",
 "пятница",
"суббота",
"воскресение" };
```

Теперь рассмотрим функцию, работающую с двумерным массивом — матрицей. Если размеры обоих индексов известны на этапе трансляции, то проблем нет:

```
void print_m34(int m[3][4])
{
for (int i = 0; i<3; i++) {
  for (int j = 0; j<4; J++)
   cout << ' ' << m[i][j];
cout << '\n';
        }
       }
```

Конечно, матрица по-прежнему передается как указатель, а размерности приведены просто для полноты описания. Первая размерность для вычисления адреса элемента неважна, поэтому ее можно передавать как параметр:

```
void print_mi4(int m[][4], int dim1)
{
   for ( int i = 0; i<dim1; i++) {
      for ( int j = 0; j<4; j++)
         cout << ' ' << m[i][j];
      cout << '\n';
   }
}
```

Самый сложный случай — когда надо передавать обе размерности. Здесь «очевидное» решение просто непригодно:

```
void print_mij(int m[][], int dim1, int dim2)   // ошибка
{
 for ( int i = 0; i<dim1; i++) {
    for ( int j = 0; j<dim2; j++)
       cout << ' ' << m[i][j];
    cout << '\n';
 }
}
```

Во-первых, описание параметра m[][] недопустимо, поскольку для вычисления адреса элемента многомерного массива нужно знать вторую размерность. Во-вторых, выражение m[i][j] вычисляется как *(*(m+i)+j), а это, по всей видимости, не то, что имел в виду программист. Приведем правильное решение:

```
void print_mij(int** m, int dim1, int dim2)
{
for (int i = 0; i< dim1; i++) {
  for (int j = 0; j<dim2; j++)
 cout << ' ' << ((int*)m)[i*dim2+j];
  cout << '\n';
   }
}
```

Выражение, используемое для выбора элемента матрицы, эквивалентно тому, которое создает для этой же цели компилятор, когда известна последняя размерность. Можно ввести дополнительную переменную, чтобы это выражение стало понятнее:

```
int* v = (int*)m;
.....
 v[i*dim2+j]
```

Лучше такие достаточно запутанные места в программе упрятывать. Можно определить тип многомерного массива с соответствующей операцией индексирования. Тогда пользователь может и не знать, как размещаются данные в массиве

9.4 Перегрузка функций

Разные функции обычно имеют разные имена, но функциям, выполняющим сходные действия над объектами различных типов, иногда лучше дать возможность иметь одинаковые имена.

На C++ возможно определение нескольких функций с одинаковыми именами, но разными типами параметров. Компилятор C++ способен различать функции не только по ее именам, но и по составу списков параметров. Комплекс *имя функции + список ее формальных параметров* иногда называют *сигнатурой функции*. Если имена функций одинаковы, но сигнатуры различны, то компилятор всегда может различить их и выбрать для вызова нужную функцию. Функции с одинаковыми именами, но различными сигнатурами называются *перегруженными*.

При вызове перегруженной функции компилятор выбирает соответствующую функцию, анализируя количество и тип аргументов в вызове. Для этого сравниваются типы фактических параметров, указанные в вызове, с типами формальных параметров всех описаний функций с данным именем. В результате вызывается та функция, у которой формальные параметры наилучшим образом сопоставились с параметрами вызова, или выдается ошибка, если такой функции не нашлось. Например:

```
void print(double);
void print(long);

void f()
{
print(1L);   // будет вызвана print(long)
print(1.0);  // будет вызвана print(double)
print(1);    // ошибка, неоднозначность: что вызывать
          // print(long(1)) или print(double(1)) ?
}
```

Правила сопоставления типов фактических параметров в выражении вызова функций с типами формальных параметров, указанными при ее объявлении, применяются в следующем порядке по убыванию их приоритета:

1. Точное сопоставление: сопоставление произошло без всяких преобразований типа или только с неизбежными преобразованиями (например, имени массива в указатель, имени функции в указатель на функцию и типа T в const T).

2. Сопоставление с использованием стандартных целочисленных преобразований, определенных в (т.е. char в int, short в int и их беззнаковых двойников в int), а также преобразований float в double.

3. Сопоставление с использованием стандартных преобразований, определенных в C++ (например, int в double, unsigned в int).

4. Сопоставление с использованием пользовательских преобразований.

5. Сопоставление с использованием эллипсиса (многоточия ...) в описании функции.

Если найдены два сопоставления по самому приоритетному правилу, то вызов считается неоднозначным, процесс компиляции останавливается и выдается соответствующее сообщение об ошибке. Эти правила сопоставления параметров работают с учетом правил преобразований арифметических типов для C и C++.

Пусть имеются такие описания функции print:

```
void print(int);
void print(const char*);
void print(double);
void print(long);
void print(char);
```

Тогда результаты следующих вызовов print() будут такими:

```
void h(char c, int i, short s, float f)
{
 print(c); // точное сопоставление: вызывается print(char)
 print(i); // точное сопоставление: вызывается print(int)
 print(s); // стандартное целочисленное преобразование:
        // вызывается print(int)
 print(f); // стандартное преобразование:
           // вызывается print(double)
```

```
print('a'); // точное сопоставление: вызывается print(char)
print(49);  // точное сопоставление: вызывается print(int)
print(0);   // точное сопоставление: вызывается print(int)
print("a"); // точное сопоставление:
            // вызывается print(const char*)
  }
```

Обращение `print(0)` приводит к вызову `print(int)`, ведь `0` имеет тип **`int`**. Обращение print('a') приводит к вызову print(char), т.к. 'a' — типа char .

Отметим, что на разрешение неопределенности при перегрузке не влияет порядок описаний рассматриваемых функций, а типы возвращаемых функциями значений вообще не учитываются.

Исходя из этих правил, можно гарантировать, что если эффективность или точность вычислений значительно различаются для рассматриваемых типов, то вызывается функция, реализующая самый простой алгоритм. Например:

```
int pow(int, int);
double pow(double, double);    // из <math.h>
complex pow(double, complex);     // из <complex.h>
complex pow(complex, int);
complex pow(complex, double);
complex pow(complex, complex);


void k(complex z)
{
int i = pow(2,2);     // вызывается pow(int,int)
double d = pow(2.0,2); // вызывается pow(double,double)
complex z2 = pow(2,z); // вызывается pow(double,complex)
complex z3 = pow(z,2); // вызывается pow(complex,int)
complex z4 = pow(z,z); // вызывается pow(complex,complex)
}
```

Таким образом, возможно определение нескольких функций с одинаковыми именами, но разными типами параметров. Эти функции называются перегруженными. При вызове перегруженной функции компилятор выбирает соответствующую функцию, анализируя количество и тип аргументов в вызове.

Перегруженные функции могут иметь разные или одинаковые типы возвращаемых значений и обязательно должны иметь разные списки параметров. Две функции, отличающиеся только типами возвращаемых значений, вызовут ошибку компиляции.

9.5 Резюме по теме функции

Функция активизируется посредством вызова функции. В вызове указывается имя функции и передается информация (в виде аргументов), которая нужна вызываемой функции для выполнения ее задачи. Выражение вызова функции состоит из имени функции, за которым в круглых скобках, следуют аргументы функции, список ее фактических параметров.

Каждый фактический параметр функции может быть константой, переменной или выражением.

Локальная переменная известна только в описании данной функции. Функции не знают детали реализации другой функции (включая локальные переменные). Цель скрытия информации в функциях заключается в том, чтобы дать доступ только к той информации, которая нужна для выполнения их задач. Это средство реализации принципа наименьших привилегий, одного из наиболее важных принципов разработки хорошего программного обеспечения.

Общий формат определения функции:

Тип-возвращаемого-значения имя-функции (список-параметров) { *Тело-функции* }

Тип-возвращаемого-значения устанавливает тип значения, возвращаемого в вызывающую функцию. Если функция не возвращает значение, *тип-возвращаемого-значения* объявляется как **`void`**.

Имя-функции — любой правильно написанный идентификатор.

Список-параметров — написанный через запятые список, содержащий объявления переменных, которые будут переданы

функции. Если функция не принимает никаких значений, *список-параметров* оставляется пустым, либо объявляется как **void**.

Тело-функции — набор объявлений и операторов, которые составляют текст программы, реализующей функцию.

Аргументы, передаваемые функции, должны быть согласованы по количеству, типу и порядку следования с параметрами в определении функции.

Когда программа доходит до вызова функции, управление передается из точки активации к вызываемой функции, функция выполняется и управление возвращается оператору вызова.

Вызываемая функция может вернуть управление оператору вызова одним из трех способов. Если функция не возвращает никакого значения (объявлен тип возврата **void**), управление возвращается при достижении закрывающей операторной скобки, охватывающей тело функции или при выполнении оператора

return;

Если функция возвращает значение, то возврат производится оператором

return *выражение;*

Где тип и значение *выражения* и определяют то, что возвращается функцией в вызывающую программу.

Прототип (или объявление) функции объявляет тип возвращаемого значения функции, количество, типы и порядок следования параметров, передаваемых в функцию.

Прототипы функций дают возможность компилятору проверить, правильно ли вызывается функция.

Компилятор игнорирует имена переменных, упомянутые в прототипе функции.

Если аргумент передается в функцию по значению, создается копия значения переменной и именно она передается вызываемой функции. Изменения копии в вызываемой функции не влияют на значение исходной переменной.

Локальные переменные, объявленные в начале функции, имеют областью действия блок подобно параметрам функции, которые считаются локальными переменными функции.

Единственными идентификаторами, имеющими областью действия прототип функции, являются те, которые использованы в списке параметров прототипа функции. Идентификаторы, использованные в прототипе функции, можно повторно использовать в других местах программы без опасений возникновения неопределенности.

Функция, не возвращающая значение, объявляется с типом void. Если предпринять попытку вернуть значение функции или использовать результат активизации функции в вызывающем выражении, компилятор сообщит об ошибке.

Пустой список параметров указывается пустыми круглыми скобками или ключевым словом **`void`** в круглых скобках.

Встраиваемые функции исключают накладные расходы, связанные с вызовом функции. Программист может использовать ключевое слово **`inline`**, чтобы рекомендовать компилятору сгенерировать машинные коды функции в нужных местах программы (если это возможно), чтобы минимизировать вызовы функции. Компилятор может проигнорировать **`inline`**.

C++ предусматривает возможность прямой формы вызова по ссылке с помощью ссылочных параметров. Чтобы указать, что параметр функции передается по ссылке, после типа параметра в прототипе функции пишется символ &. В вызове функции переменная указывается по имени, но она будет передана по ссылке. В вызываемой функции обозначение переменной ее локальным именем на самом деле отсылает к исходной переменной в вызывающей функции. Таким образом, исходная переменная может быть изменена с помощью вызываемой функции.

Спецификация **`const`** может создать именованную константу. Именованная константа должна получить в качестве начального значения постоянное выражение и после этого не может изменяться. Именованные константы часто называют постоянными переменными или переменными только для чтения. Именованные константы могут быть помещены всюду, где может быть помещено постоянное выражение. Другим распространенным применением спецификации **`const`** является создание ссылок на константы.

ЛИТЕРАТУРА

1. Симонович С.В. Информатика. Базовый курс. 2-е изд.: Учебник для вузов. — СПб.: Питер, 2006. — 640 с.

2. Конев Ф.Б. Информатика для инженеров: Учебное пособие. — М.: Высшая школа, 2004. — 272 с.

3. Острейковский В А. Информатика: Учебник для вузов. — М.: Высшая школа, 2001. — 512 с.

4. Акулов О.А., Медведев Н.В. Информатика: базовый курс. 4-е изд. Учебник для вузов. — М.: Омега-Л, 2007, 560 с.

5. Н. В. Макарова [и др.] Информатика: Практикум по технологии работы на компьютере: Учебное пособие для вузов. — 3-е изд., перераб. — М.: Финансы и статистика, 2005. — 255 с.

6. Павловская Т.А. С/С++. Программирование на языке высокого уровня: Учебник для вузов. — СПб.: Питер, 2006. — 461 с.

7. Воройский Ф.С. Информатика. Новый систематизированный толковый словарь-справочник. — М.: ФИЗМАТЛИТ, 2003. — 760 с.

8. Пасько В. Энциклопедия ПК. Аппаратура. Программы. Интернет. — СПб.: Питер, 2004. — 800 с.

9. Волкотруб Л.П., Егоров И.М. Компьютер и здоровье. — Томск: Томск. гос. ун-т систем упр и радиоэлектроники, 2006. — 158 с.

10. Дьяконов В. Mathcad 2001: Учебный курс. — СПб.: Питер, 2001.

11. Кирьянов Д.В. Самоучитель Mathcad 13. — СПб.: БХВ-Петербург, 2006. — 528 с.

12. Франка П. С++: Учебный курс: Пер. с англ. П. Бибикова. — СПб.: Питер, 2005. — 521 с

13. Павловская Т.А., Щупак Ю.А. С++. Объектно-ориентированное программирование: практикум: Учебное пособие для вузов. — СПб.: Питер, 2005. — 464 с

14. Боровский А.Н. Самоучитель С++ и Borland C++ Builder: самоучитель. — СПб.: Питер, 2005. — 255 с.