СРЕДНЕЕ
ПРОФЕССИОНАЛЬНОЕ
ОБРАЗОВАНИЕ

В.Ф. ЛЯХОВИЧ
В.А. МОЛОДЦОВ
Н.Б. РЫЖИКОВА

ОСНОВЫ ИНФОРМАТИКИ

Рекомендовано ФГАУ «ФИРО»
в качестве **учебника** для использования
в учебном процессе образовательных учреждений,
реализующих программы СПО

Регистрационный номер рецензии № 356 от 17.07.2013 ФГАУ «ФИРО»

КНОРУС • МОСКВА • 2016

УДК 004(075.32)
ББК 32.973я723
Л98

Рецензенты:
П.В. Панкратов, начальник учебно-методического управления Южно-Российского государственного технического университета, канд. техн. наук, доц.,
С.Ф. Скачек, преподаватель информационных технологий ГБОУ СПО «Строительный колледж № 12»

Ляхович В.Ф.

Л98 Основы информатики : учебник / В.Ф. Ляхович, В.А. Молодцов, Н.Б. Рыжикова. — М. : КНОРУС, 2016. — 348 с. — (Среднее профессиональное образование).

ISBN 978-5-406-04695-1

Изложен процесс разработки основных видов алгоритмов и программ на базе оригинальной общей методики построения алгоритмов, не имеющей аналогов в России. Методика позволяет быстро осваивать различные языки программирования высокого уровня. Отражены принципы построения и развития компьютерных сетей. Главы книги соответствуют разделам учебной программы и содержат необходимые теоретические сведения, особенно подробно излагаются темы, вызывающие трудности при самостоятельном изучении.

Соответствует действующему Федеральному государственному образовательному стандарту среднего профессионального образования нового поколения.

Для учебных заведений среднего профессионального образования. Может быть использован в старших классах общеобразовательной школы и для самостоятельной подготовки к ЕГЭ по информатике.

УДК 004(075.32)
ББК 32.973я723

Ляхович Владислав Федорович
Молодцов Валерий Алексеевич
Рыжикова Наталья Борисовна

ОСНОВЫ ИНФОРМАТИКИ

Сертификат соответствия № РОСС RU. АЕ51. Н 16604 от 07.07.2014.

Изд. № 9432. Формат 60×90/16. Гарнитура «NewtonC».
Усл. печ. л. 22,0. Уч.-изд. л. 13,41.

ООО «Издательство «КноРус».
117218, г. Москва, ул. Кедрова, д. 14, корп. 2.
Тел.: 8-495-741-46-28.
E-mail: office@knorus.ru http://www.knorus.ru

Отпечатано в ООО «Контакт».
107150, г. Москва, проезд Подбельского 4-й, дом 3.

ISBN 978-5-406-04695-1

ОГЛАВЛЕНИЕ

ПРЕДИСЛОВИЕ

Данное пособие призвано помочь студенту в изучении курса информатики и ИКТ. Книга написана по ныне действующей программе по информатике и отвечает требованиям стандарта. Главы книги соответствуют разделам программы и содержат теоретические сведения, особенно подробно излагаются темы, вызывающие трудности при самостоятельном изучении.

Учебник можно разбить на два раздела: первый — главы 1—9, второй 10—14 главы.

Первый раздел посвящен изучению теоретической информатики, средств информатизации и информационных технологий. В них вводятся представления об основных информационных процессах в живой природе, обществе, технике, раскрывается роль информации в процессах управления. Формируются способности обучаемых использовать различные языки (естественные и искусственные) для представления информации, построения информационных моделей из области естественных и гуманитарных дисциплин, дается представление о функциональном устройстве, структуре и функционировании средств информационных и коммуникационных технологий, основных компонентах программного обеспечения. Информационные технологии изучаются в 4—9 главах пособия. Содержание этих глав направлено на формирование готовности обучаемых использовать типовые информационные и коммуникационные технологии для решения познавательных и практических задач.

Основная тема второго раздела — алгоритмизация и программирование, основная его цель — научить программированию достаточно сложных задач (за время, отведенное на уроки информатики).

Составление программ для компьютера весьма сложный процесс, включающий в себя значительное число качественно разнообразных этапов. Наиболее сложный из них — постановка задачи и ее алгоритмизация. Именно этим этапам уделяется основное внимание в пособии.

Вышло много книг по информатике и программированию для ЭВМ, и почти в каждой из них рассматриваются указанные вопросы. Однако данное пособие может претендовать на оригинальность в изложении ряда вопросов. В учебнике собран комплекс методических разработок, педагогических находок и приемов, явившихся результатом многолетнего опыта преподавания различных курсов по ЭВМ и программированию и позволяющих резко повысить эффективность

преподавания соответствующих дисциплин, особенно для слабоподготовленных учащихся.

Принципиальное отличие настоящего пособия от большинства аналогичных состоит в том, что в разделе алгоритмизации излагается унифицированная и формализованная процедура перехода от словесно-формульного описания метода решения задачи к схеме алгоритма этой задачи, причем такой схеме, которая может быть чисто формально перекодирована в программу на языке программирования. Эта процедура изложена в виде довольно четкой методики и позволяет строго логически выводить формулы и условия, составляющие «начинку» алгоритма (программы), причем *правильные* формулы и условия, что может быть доказано в каждом конкретном случае.

Особенность такого подхода еще и в том, что он требует от учащегося подробного описания процесса вывода алгоритма, поэтому процесс обучения алгоритмизации становится хорошо наблюдаемым для преподавателя, а значит и хорошо управляемым.

Упомянутая методика излагается без излишней формализации и математизации и вполне доступна любой категории учащихся на начальной стадии обучения информатике и программированию. Она базируется на известных подходах к построению программы — структурном программировании и проектировании «сверху вниз» и является их развитием. Именно эта методика используется при решении всех видов задач, рассматриваемых в пособии, даже при решении простейших из них — составлении линейных алгоритмов; тем самым достигается единый методический подход в пределах пособия.

Методика достаточно универсальна. Как показано в пособии, она применима при решении задач обработки не только числовой, но и текстовой и графической информации.

Отметим, что в качестве основного средства описания алгоритмов в настоящем учебнике выбраны схемы алгоритмов — наиболее наглядный и понятный, а кроме того, и наиболее естественный для человека способ, так как человек мыслит образами и ему легче воспринимать образы, нежели тексты.

Другой важный вопрос, рассматриваемый в учебнике, — постановка задач, причем, что существенно, он изучается значительно глубже, чем это принято в учебной литературе. В частности, подробно рассматривается процесс перехода от содержательной постановки задачи к математической и выявляется суть этого процесса. В самостоятельный этап выделен процесс перехода математической задачи к такой, которая может быть описана и решена средствами используемого языка программирования. Этот этап назван *формализацией задачи*. Он

тщательно проработан и проиллюстрирован значительным число примеров.

В учебнике использован ряд методических приемов, которые можно было бы отнести к разделу «маленьких хитростей». В частности, как обязательный элемент обучения алгоритмизации — отладка алгоритмов каждого вида. Этот элемент позволяет учащимся не только «прочувствовать» виды алгоритмов, но и предлагает им правила самоконтроля — правила нахождения ответов к задачам алгоритмизации, позволяет понять и закрепить принципы отладки программ.

Изложение основ программирования в пособии ведется на языках Бейсик и Паскаль как наиболее доступных в пределах России и наиболее перспективных с точки зрения профессионального применения навыков программирования в будущем.

Все вместе взятые особенности в содержании и изложении второго раздела пособия позволяют существенно ускорить процесс обучения составлению алгоритмов и программ для компьютера.

Как показала практика, применение изложенного в этом разделе подхода в курсе «Информатики и ИКТ» позволяет ставить в числе прочих перед преподавателем цель — научить основную массу учащихся составлять алгоритмы и программы достаточно сложных задач.

При традиционном подходе можно говорить лишь о знакомстве с понятиями «алгоритм» и «программа» и прочими в результате изучения этого курса (опять-таки для основной массы учащихся).

Обобщая сказанное, отметим, что настоящее пособие охватывает весь комплекс понятий, связанный с изучением курса «Информатики и ИКТ», оно содержит все необходимые сведения, излагаемые конспективно, и может использоваться для самостоятельного изучения предмета.

Не замыкаясь в рамках указанного общеобразовательного курса, пособие перекидывает мостик к тем дисциплинам, которые обеспечивают профессиональную подготовку программистов, во всяком случае на уровне средних специальных учебных заведений как минимум.

Несмотря на то что учебник ориентирован в первую очередь на учебные заведения среднего профессионального образования, он с успехом может быть использован как в старших классах общеобразовательной школы, так и для самостоятельной подготовки к ЕГЭ по информатике.

ГЛАВА 1

ИНФОРМАЦИЯ. КОЛИЧЕСТВО ИНФОРМАЦИИ И ЕЕ КОДИРОВАНИЕ. ИНФОРМАТИКА КАК НАУКА

1.1. Информация и информационные процессы

Информация — это сведения, знания и сообщения, получаемые человеком из различных источников.

Но для человека важен не весь общий поток информации, а только определенная его часть. Какая — зависит от многих условий (профессия человека, его увлечения, уровень образования и т.д.). Для получения и приема информации могут быть использованы различные носители: СМИ, Интернет, книги, записи. Любой предмет, находящийся вокруг нас, может содержать в себе информацию.

В настоящее время разнообразная по своему значению информация, зафиксированная на специальных носителях, стала национальным богатством нового типа — информационным ресурсом государства. Являясь предметом купли-продажи во все времена, информация имеет свои специфические особенности: при обмене информацией ее количество увеличивается. «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, то у каждого их будет по две». Общение людей, информирование друг друга приводит к их сближению, повышению интеллектуального потенциала. У информационных ресурсов есть еще уникальное свойство — они не убывают от интенсивного использования. Более того, в процессе применения они постоянно развиваются и совершенствуются, избавляясь от ошибок и уточняя свои параметры.

Информация должна быть:

- достоверной — нельзя назвать информацией сообщение о том, что идет снег, а вы видите, что за окнами август и светит солнце;
- понятной — мы не сможем сказать, что владеем ценной информацией, если она была передана на языке, которого мы не знаем;
- актуальной — важна именно та информация, которая получена вовремя;

- полной — наличие неполной информации иногда даже хуже, чем полное отсутствие информации;
- полезной — хотя это свойство нельзя назвать абсолютным — полезной информация может быть для одного человека, а для другого — нет.

Информацию, передаваемую видимыми образами и символами, называют визуальной, звуками — аудиальной, ощущениями — тактильной, запахами и вкусами — органолептической, информацию, выдаваемую и воспринимаемую средствами вычислительной техники, — машинной.

Каким образом можно представить информацию? Она может быть представлена в виде текста, рисунка, набора определенных цифр, музыкального произведения или в комбинированном виде, например в виде кинофильма и т.д.

Разнообразие источников и потребителей информации привело к существованию различных форм ее представления: символьной, текстовой и графической. Символьная форма основана на использовании символов — букв, цифр, знаков, рисунков и т.д., является наиболее простой, но она практически применяется только для передачи несложных сигналов о различных событиях (например, дорожные знаки). Более сложной является текстовая форма представления информации. Наиболее емкой и сложной является графическая форма представления информации. К этой форме относятся фотографии, схемы, чертежи, рисунки, играющие большое значение в деятельности человека.

Можно также охарактеризовать информацию как отражение внешнего мира с помощью знаков и сигналов. Любой передаваемый сигнал переносится либо веществом (текст, наскальный рисунок, гены и т.д.), либо энергией (звук, свет, радиоволны и т.д.).

Способ передачи информации — сигнал (от англ. *sign* — знак, символ) — свет, код Морзе, текст, электрический импульс и т.д. Сигнал — это физический процесс, имеющий информационное значение. Сигнал может быть непрерывным (аналоговым) или дискретным (прерывистым).

Аналоговый сигнал — сигнал, непрерывно изменяющийся по амплитуде и во времени (напряжение, ток, температура и т.д.). Используют в проводной телефонной связи, кардиограмме и т.д.

Дискретный сигнал — сигнал, который может принимать лишь конечное число значений в конечное число моментов времени.

Информационные процессы. К информационными процессами относятся такие процессы, как:

1) хранение информации.

Способ хранения информации зависит от ее носителя (книга — библиотека, картина — музей, фотография — альбом и т.д.). Хранение больших объемов информации оправданно только при условии, если известен способ оперативного доступа к ней, а получаемые сведения представлены в доступной форме.

Компьютер — универсальное устройство для компактного хранения информации с возможностью быстрого (оперативного) доступа к ней;

2) передача информации.

Информацию могут передавать друг другу не только люди, но и животные и растения. В этом процессе участвуют источник и приемник информации с каналом передачи информации — каналом связи между ними.

Устройство, предназначенное для преобразования исходного сообщения источника информации к виду, удобному для передачи, называется кодирующим устройством.

Устройство, служащее для преобразования кодированного сообщения в исходное, называется декодирующим устройством.

Канал связи — совокупность технических устройств, обеспечивающих передачу сигнала от источника к получателю.

Каналы передачи сообщений характеризуются пропускной способностью и помехозащищенностью. Канала передачи данных разделяются:

- на симплексные (с передачей информации только в одну сторону);
- полудуплексные (с передачей информации по двум направлениям попеременно);
- дуплексные (с передачей информации по двум направлениям).

Пропускная способность канала определяется максимальным количеством символов, передаваемых по нему в отсутствие помех, и зависит от его физических свойств;

3) обработка информации.

Обработка информации — преобразование информации из одного вида в другой;

4) защита информации.

Защитой информации называется предотвращение несанкционированного доступа к информации с целью ее недозволенного использования, изменения или разрушения.

Информация и управление. Основной и постоянной функцией мозга и нервной системы человека или животного является преобразование информации о состоянии окружающей среды и выбор наиболее це-

лесообразного поведения. Процесс преобразования исходной информации в информацию, отражающую результат решения какой-либо задачи, — это и есть решение задачи, поставленной перед человеком в любом виде его деятельности. Процессом управления в любой области, основой выбора метода решения и является преобразование, анализ информации.

Рассмотрим, как осуществляется процесс управления, например велосипедом.

С помощью слуха и зрения человек получает информацию об окружающей среде (состояние дороги, дорожные знаки, сигналы светофора, наличие встречного транспорта, пешеходов и т.д.). Эта информация передается в мозг, где преобразуется в последовательность сигналов нервным окончаниям, управляющим движением ног и рук, которые воздействуют на руль и тормоза велосипеда. То есть без информации, ее передачи, преобразования и использования управление невозможно. В основе любого процесса управления лежат информационные процессы.

В процессе управления происходит взаимодействие двух систем — управляющей и управляемой. Если они соединены двумя каналами: первым — от управляющей системы к управляемой, второй — от управляемой системы к управляющей, то такую систему называют замкнутой или системой с обратной связью.

По каналу прямой связи передаются сигналы управления, которые вырабатывает управляющая система. Управляемая система выполняет свои функции, подчиняясь этим командам, и в свою очередь передает информацию о своем состоянии по каналам обратной связи управляющей системе. В управляющем органе эта информация используется для соответствующей корректировки сигналов управления.

Очень наглядно процесс управления с обратной связью иллюстрирует следующий пример: поддержание постоянно заданной температуры нагрева электрической печи. Без применения автоматических средств человек должен выполнять три задачи: 1) следить за показаниями термометра, 2) сравнивать их с заданной температурой и 3) если данные различаются, то изменить силу тока и температуру электрической печи, передвигая ползунок реостата.

Автоматическая система, решающая эту задачу, сводится к схеме, изображенной на рис. 1.1.

Измерительный орган (в нашем случае датчик) измеряет регулируемую величину (температуру) и преобразует ее в величину, более удобную для использования в управляющем органе. Управляющий орган сравнивает эту информацию с заданным значением и при наличии

расхождения передает соответствующую команду на исполнительный орган, который и изменяет значение регулируемой величины (температуры). В качестве исполнительных органов используются такие устройства, как двигатели и электромагниты.

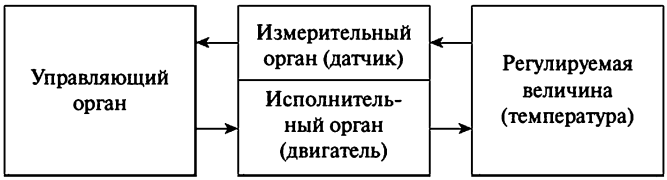


Рис. 1.1

Такая замкнутая система (система с обратной связью) представляет собой типичный пример систем автоматического регулирования.

1.2. Информатика. Ее роль и место в системе научного знания

Современная научно-техническая революция привела к гигантскому росту экономического и социального значения новой деятельности человека — информационной. Индустрия информатики находится на самом острие научно-технического прогресса и продолжает бурно развиваться. Это развитие обусловливает эффективное функционирование всех отраслей народного хозяйства, науки и культуры.

По данным ЮНЕСКО, более половины занятого населения всех развитых стран участвует в процессе производства и распространения информации, происходит процесс перераспределения трудовых ресурсов из сферы материального производства в информационную сферу. Например, в информационной сфере США уже сейчас работает более 60% занятого населения страны. Это свидетельствует о начале перехода развитых стран на качественно новый этап технического развития. Индустрия информатики играет для промышленности этих стран огромную роль, сопоставимую с той ролью, которую играла тяжелая промышленность на этапе индустриализации. Информационные ресурсы становятся основным национальным богатством, а экономическую мощь страны определяет эффективность их эксплуатации. Все это привело к росту роли науки информатики.

Информатика — наука, изучающая свойства информации, а также способы представления, накопления, обработки и передачи информации с помощью технических средств.

В информатике выделяют три основных направления: теоретическое, практическое и техническое. Они изучают:

1) теоретические вопросы информатики, связанные с теорией информации, теорией алгоритмов, математической логикой и комбинаторным анализом;
2) практические вопросы информатики, связанные с программированием и использованием прикладных программ;
3) проектирование, разработку и использование технических средств обработки информации.

В настоящее время человечество завершает переход к **информационному обществу**, в котором можно эффективно и оптимально строить любую деятельность на основе овладения информацией о самых различных процессах и явлениях. В информационном обществе повышается качество не только потребления, но и производства, труд человека становится творческим и интеллектуальным, если он использует новые информационные технологии, значительно улучшаются условия труда.

1.3. Измерение информации

Количество информации зависит от новизны сведений об интересном для получателя информации явлении, другими словами, получая информацию, мы уменьшаем неполноту знаний, т.е. неопределенность. Если в результате полученного сообщения неопределенность полностью исчезнет, т.е. будет достигнута полная ясность, то говорят, что полученная информация была исчерпывающей, полной.

Существует формула, связывающая между собой количество возможных событий (N) и количество информации (I):

$$N = 2^I.$$

Воспользовавшись этой формулой, можно определить и количество информации по заданному количеству событий.

Информацию можно понимать как результат интерпретации I сообщения r. Информационной системе обычно ставят в соответствие понятие отображения из математики. Если имеется точно описанное множество R представлений с интерпретацией I в множестве A элементов (информаций), то интерпретации соответствует отображение $I: R \to A$. С этой точки зрения интерпретация сообщению r ставит в соответствие абстрактное содержание $I(r)$. Такой подход позволяет описывать процессы преобразования информации с помощью математических формул.

Большой вклад в теорию информации внес Клод Шеннон. В частности, он ввел количественную оценку информации как меру снятой неопределенности. Для понимания формулы количества информации необходимо ознакомиться с понятием случайных процессов как математической модели сигналов, понятием энтропии и неопределенности.

Концепция К. Шеннона, отражая количественно-информационный подход, определяет информацию как меру неопределенности (энтропию) события. Количество информации в том или ином случае зависит от вероятности его получения: чем более вероятным является сообщение, тем меньше информации содержится в нем. Этот подход оказался весьма полезным в технике связи и вычислительной технике и послужил основой для измерения информации и оптимального кодирования сообщений. Кроме того, он представляется удобным для иллюстрации такого важного свойства информации, как новизна, неожиданность сообщений. При таком понимании информация — это снятая неопределенность, или результат выбора из набора возможных альтернатив.

Шеннон вывел формулу для вычисления количества информации в случае различных вероятностей событий:

$$I = -\sum_{i=1}^{N} p_I \log_2 p_I,$$

где I — количество информации, N — количество возможных событий, p_I — вероятность *i*-го события.

Как мы уже отмечали ранее, обмен информацией происходит при помощи сигналов. Сигналы, передаваемые по радио и телевидению, а также используемые в магнитной записи, имеют форму непрерывных быстро изменяющихся во времени кривых линий. Такие сигналы называются непрерывными, или аналоговыми сигналами. В противоположность этому в телеграфии и вычислительной технике сигналы имеют импульсную форму и именуются дискретными сигналами. Другими словами, информация передается в двух формах: дискретной и аналоговой.

Для определения количества любой информации как символьной, так и текстовой или графической нужно найти способ представить ее в едином, стандартном, виде. Таким видом стала двоичная форма представления информации — записи любой информации в виде последовательности только двух символов, например: цифрами 0 или 1, буквами А или Б; словами ДА, НЕТ. Однако ради простоты записи применяют цифры 1 и 0. В компьютере эти сигналы рассматриваются как наличие или отсутствие напряжения.

Давайте поясним принцип информации в двоичной форме, проведя следующую игру. Нам нужно получить интересующую нас информацию у собеседника, задавая любые вопросы, но получая в ответ ДА либо НЕТ.

Для получения двоичной формы информации необходимо перечисление всех возможных событий. Например, задаем один вопрос: «Вы сегодня обедали?» С одинаковой вероятностью следует ожидать ответ: «ДА» или «НЕТ», причем любой из этих ответов несет самое малое количество информации. Эта минимальная единица измерения информации называют БИТОМ. Благодаря введению понятия единицы информации появилась возможность определения размера любой информации в битах. Восемь последовательных битов составляют **байт**. В одном байте можно закодировать значение одного символа из 256 возможных ($256 = 2^8$). Байт записывается в памяти машины, читается и обрабатывается обычно как единое целое. Наряду с битами и байтами для измерения количества информации используются и более крупные единицы:

1 Килобайт (Кбайт, Кб) = 1024 или 2^{10} байт;

1 Мегабайт (Мбайт, Мб) = 1 048 576 или 2^{20} байт, или 1024 Кбайт;

1 Гигабайт (Гбайт, Гб) = 1 099 511 627 776 или 2^{30} байт, или 1024 Мбайт.

1 Терабайт (Тбайт, Тб) = 1 099 511 627 776 или 1024 Гбайт.

1.4. Системы счисления. Кодирование информации

Системой счисления называется совокупность символов (цифр) и правил их использования для представления чисел.

Позиционные и непозиционные системы счисления

Существует два вида систем счисления:

Непозиционные системы счисления. Примером этой системы счисления является римская система, в которой в качестве цифр используются некоторые буквы: I (1), V (5), X (10), L (50), C (100), D (500), M (1000). Значение цифры не зависит от ее положения в числе. Например, в числе **XXX** цифра **X** встречается трижды и в каждом случае обозначает одну и ту же величину 10, а в сумме **XXX** — 30.

Величина числа в римской системе счисления определяется как сумма или разность чисел. Если меньшая цифра стоит слева от большей, то она вычитается, если справа — прибавляется.

Например:

$$1998 = \text{MCMXCVIII} = 1000 + (1000 - 100) + (100 - 10) + 5 + 1 + 1 + 1,$$
$$2002 = \text{MMII} = 1000 + 1000 + 1 + 1.$$

Позиционные системы счисления. В позиционной системе счисления количественное значение цифры зависит от ее позиции в числе. Позиция цифры называется **разрядом**. Разряд числа возрастает справа налево.

В позиционной системе счисления **основание** системы равно количеству цифр, используемых ею, и определяет, во сколько раз различаются значения цифр соседних разрядов чисел.

Наиболее известна десятичная позиционная система счисления. В 595 году (уже нашей эры) в Индии впервые появилась знакомая всем нам сегодня десятичная система счисления. Знаменитый персидский математик Аль-Хорезми выпустил учебник, в котором изложил основы десятичной системы индусов. После перевода его с арабского языка на латынь и выпуска книги Леонардо Пизано (Фибоначчи) эта система счисления стала доступна европейцам, получив название арабской.

В написании десятичных цифр существует строгая система: количество углов в цифре соответствует числу, которое эта цифра изображает.

В настоящее время существуют и так называемые машинные системы счисления, применяемые в компьютерах, такие как двоичная, восьмеричная и шестнадцатеричная. Двоичная система счисления была впервые предложена Г.В. Лейбницем в 1703 г.

Любое число, записанное в позиционной системе счисления с произвольным основанием, можно записать в виде полинома (многочлена):

$$A_{(S)} = a_n S^n + a_{n-1} S^{n-1} + \ldots + a_1 S^1 + a_0 S^0 + a_{-1} S^{-1} + \ldots + a_{-m} S^m,$$

где S — основание системы счисления, а степень соответствует разряду цифры a в числе $A_{(S)}$.

Приведем пример записи числа в десятичной системе счисления:

$$34^5_{10} = 3 \cdot 10^2 + 4 \cdot 10^1 + 5 \cdot 10^0,$$
$$45^9_{10} = 400 + 50 + 9 = 4 \cdot 10^2 + 5 \cdot 10^1 + 9 \cdot 10^0.$$

Если число имеет дробную часть, то добавляется сумма оснований 10 с отрицательными степенями. Например:

$$321{,}409_{10} = 3 \cdot 10^2 + 2 \cdot 10^1 + 1 \cdot 10^0 + 4 \cdot 10^{-1} + 0 \cdot 10^{-2} + 9 \cdot 10^{-3}.$$

Двоичное кодирование информации

Мы уже знаем, что компьютер может обрабатывать информацию, представленную только в виде двоичных чисел, но из опыта мы знаем, что он может обрабатывать числовую, текстовую, графическую видео- и звуковую информацию. Так каким же образом компьютер обрабатывает столь различающиеся по восприятию человеком виды информации? Эта информация (звуки, изображения и т.п.) для обработки на компьютере должна быть сначала преобразована в числовую форму, т.е. информация кодируется в последовательности нулей и единиц. Такое кодирование информации в компьютере называется двоичным кодированием, а логические последовательности нулей и единиц — машинным языком.

Двоичное кодирование текстовой информации. В настоящее время большая часть персональных компьютеров в мире занято обработкой текстовой информации.

Для представления текстовой информации используется 256 различных символов. Эти символы включают в себя прописные и заглавные буквы русского и латинского алфавита, цифры, знаки, графические символы и т.д.

Для двоичного кодирования 1 символа необходим 1 байт информации или 8 двоичных разрядов, то есть каждому известному нам символу соответствует своя уникальная последовательность из восьми нулей и единиц, которая фиксируется в кодовой таблице. Символы кодируются следующим образом: при нажатии на определенную клавишу в компьютер передается соответствующая этому символу последовательность нулей и единиц на машинном языке. Специальная программа, которая называется драйвер клавиатуры и экрана, по кодовой таблице определяет символ и изображает его на экране. Тексты хранятся в памяти компьютера в двоичном коде и программным способом преобразуются в изображения символов на мониторе компьютера.

1.5. Перевод чисел из одной системы счисления в другую. Двоичная арифметика

Перевод чисел в десятичную систему из системы счисления с произвольным основанием

Для того чтобы перевести число в десятичную систему счисления, запишем его в виде известного нам полинома:

$$A_{(S)} = a_n S^n + a_{n-1} S^{n-1} + \ldots + a_1 S^1 + a_0 S^0 + a_{-1} S^{-1} + \ldots + a_{-m} S^m.$$

и вычислим его значение.

Например: переведем двоичное число 111101_2 в десятичную систему счисления:

$$111101_2 = 1 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = \\ = 32 + 16 + 8 + 4 + 1 = 61_{10}.$$

Аналогично можно осуществить перевод и троичного числа 221_3 в десятичную систему счисления:

$$221_3 = 2 \cdot 3^2 + 2 \cdot 3^1 + 1 \cdot 3^0 = 18 + 6 + 1 = 25_{10}.$$

Перевод из десятичной системы счисления в систему счисления с произвольным основанием

Существует несколько способов перевода чисел из десятичной системы счисления в систему счисления с произвольным основанием.

Рассмотрим первый способ.

Для перевода нужно представить исходное число в виде полинома

$$A_{(S)} = a_n S^n + a_{n-1} S^{n-1} + \ldots + a_1 S^1 + a_0 S^0 + a_{-1} S^{-1} + \ldots + a_{-m} S^m,$$

взяв в качестве S основание той системы счисления, в которую данное число нужно перевести. Затем выпишем коэффициенты $a_n - a_{-m}$, которые и составят нужную цифру.

Например: переведем число 13_{10} в систему счисления с основанием 2. Для этого представим 13 как сумму степеней числа 2:

$$13_{10} = 8 + 4 + 1.$$

Воспользуемся формулой

$$A_{(S)} = a_n S^n + a_{n-1} S^{n-1} + \ldots + a_1 S^1 + a_0 S^0$$

и запишем число 13 в виде полинома

$$13_{10} = \underline{1} \cdot 2^3 + \underline{1} \cdot 2^2 + \underline{0} \cdot 2^1 + \underline{1} \cdot 2^0.$$

Теперь выпишем все коэффициенты

$$a_n - a_0 : 1101.$$

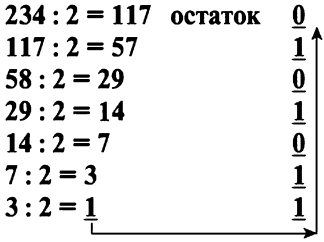
Таким образом, десятичное число 13_{10} в двоичной системе счисления будет записано как 1101_2.

Аналогично можно осуществить перевод этого числа и в другую, например в троичную систему счисления:

$$13_{10} = 9 + 3 + 1 = \underline{1} \cdot 3^2 + 1 \cdot 3^1 + \underline{1} \cdot 3^0 = 111_3.$$

Существует и другой, более простой способ перевода — перевод методом последовательного деления:

Например, перевести в двоичную систему счисления число 234_{10}.



Выписываем остатки, начиная с результата последнего деления:

$$234_{10} = 1101010_2.$$

Перевод правильных десятичных дробей в систему счисления с произвольным основанием выполняют по следующему правилу: дробь умножить на число, равное основанию системы счисления в которую переводим число, и отделить целую часть. Умножение нужно производить до тех пор, пока дробная часть не станет равной нулю.

Например: переведем число $0{,}25_{10}$ в двоичную систему счисления:

$$0{,}25 \cdot 2 = \underline{0},5 \text{ целая часть равна } \underline{0},$$
$$0{,}5 \cdot 2 = \underline{1},0 \text{ целая часть равна } \underline{1}$$

Таким образом, $0{,}25_{10} = 0{,}01_2$.

Арифметические операции в двоичной системе счисления

Сложение двоичных чисел, как и в любой позиционной системе, осуществляется вычислением суммы значений одноименных разрядов и единицы переноса из предыдущего разряда, если она есть. Перенос производится, если эта сумма не меньше, чем основание системы счисления, т.е. число 2.

Сложение двоичных чисел производится в соответствии со следующими правилами:

$0 + 0 = 0$
$0 + 1 = 1$
$1 + 0 = 1$
$1 + 1 = 10$ (0 и единица переноса в следующий, старший разряд).

Например:

Сложим самые простые числа 1_2 и 1_2

$$\begin{array}{r} 0001 \\ +\ 0001 \\ \hline 0010 \end{array}.$$

Теперь сложим 7_2 и 4_2

$$\begin{array}{r} 111 \\ +\ 100 \\ \hline 1011 \end{array}.$$

Сложим 9_2 и 3_2

$$\begin{array}{r} 1001 \\ +\ 0011 \\ \hline 1100 \end{array}.$$

Вычитать двоичные числа можно также поразрядно по следующим правилам:

$$\begin{array}{l} 0 - 0 = 0 \\ 10 - 1 = 1 \\ 1 - 0 = 1 \\ 1 - 1 = 0 \end{array}$$

Выполняя вычитание из ноля единицы, следует занять единицу из старшего значащего разряда.

$$\begin{array}{r} 100 \\ -\ 001 \\ \hline 011 \end{array}$$

Можно рассматривать вычитание как сложение положительного числа с отрицательным числом. В компьютере для представления отрицательных чисел используется **дополнительный** код. Дополнительный код двоичного числа получается путем инверсии (замены единицы нулем, а нуля — единицей) всех разрядов числа и последующего прибавления единицы к младшему разряду. Можно получить дополнительный код и так: от исходного десятичного кода отнять единицу, а затем перевести его в двоичный код и проинвертировать. В памяти компьютера числа записываются в ячейке памяти по 8 двоичных знаков, поэтому число, например 2, будет записано как 00000010. Чтобы получить из числа 2 число минус 2, нужно выполнить следующие преобразования

1) **00000010** — число 2;
2) **11111101** — заменив нули единицами, а единицы нулями, получим инвертированное число 2;

3) $+\begin{array}{r}11111101\\ \underline{00000001}\end{array}$ — прибавим к инверсному коду числа единицу;

4) **11111110** — получим дополнительный код двоичного числа минус 2.

Сложив полученный дополнительный код числа с уменьшаемым, можно получить разность двоичных чисел. Кстати, если найти дополнительный код отрицательного числа, то получим противоположное ему положительное число.

Восьмеричная и шестнадцатеричная системы счисления

Основной недостаток двоичной системы состоит в том, что для записи даже не очень больших чисел приходится использовать много знаков, поскольку основание системы счисления мало. Поэтому в современных компьютерах помимо двоичной системы счисления применяют и другие, более компактные по длине чисел системы, такие как восьмеричная и шестнадцатиричная.

В восьмеричной системе 8 цифр: **1**, **2**, **3**, **4**, **5**, **6**, **7**, число восемь обозначается 10 (один и ноль), 64_{10} — не что иное, как 100_8.

Перевод чисел из восьмеричной системы счисления в десятичную и обратно можно осуществить по уже известным правилам.

Например:
Переведем число 611_8 в десятичную систему:

$$611_8 = 6 \cdot 8^2 + 1 \cdot 8^1 + 1 \cdot 8^0 = 6 \cdot 64_{10} + 1 \cdot 8_{10} + 1 = 39^3_{10}.$$

Теперь переведем число 611_8 в двоичную систему. Для этого нужно заменить каждую цифру восьмеричного числа группой из трех двоичных цифр:

0	**000**
1	**001**
2	**010**
3	**011**
4	**100**
5	**101**
6	**110**
7	**111**

В нашем числе 611_8 заменим цифру **6** группой **110**, **1 – 001** и получим:

$$611_8 = 110\ 001\ 001_2.$$

Для того чтобы перевести в восьмеричную систему счисления многозначное двоичное число, его нужно разбить на группы по три цифры справа налево (если количество цифр в числе не кратно трем, то впереди надо дописать нужное количество нулей) и заменить каждую группу соответствующей восьмеричной цифрой.

Например:

$$1\ 111\ 101\ 001_2 = \overbrace{011}\ \overbrace{111}\ \overbrace{101}\ \overbrace{001}_2 = 3751_8.$$

Запись числа еще компактнее в 16-ричной системе. Так как цифр мы знаем всего десять, то для записи шестнадцатеричных цифр, больших 9 используют первые буквы латинского алфавита. Перевод из шестнадцатеричной системы счисления в двоичную и обратно аналогичен переводу из восьмеричную и обратно. Разница только в том, что шестнадцатеричные цифры заменяются группами по четыре двоичные цифры:

0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

Например:

$$A0F_{16} = 1010\ 0000\ 1111_2,$$
$$11\ 1110\ 1001_2 = \overbrace{0011}\ \overbrace{1110}\ \overbrace{1001}_2 = 3E9_{16}.$$

1.6. Примеры решения задач

Задача № 1

Определите информационный объем сообщения:

В одном байте восемь бит информации.

Решение

Посчитаем количество символов, включая пробелы и знаки препинания, в этом тексте 36 символов. В 8-ми битной кодировке информационный объем сообщения будет равен $36 \times 8 = 288$ бит, соответственно, в 16-битной этот объем будет вдвое больше.

Задача № 2

Автоматическое устройство осуществило перекодировку информационного сообщения на русском языке, первоначально записанного в 16-битном коде Unicode, в 8-битную кодировку КОИ-8. При этом информационное сообщение уменьшилось на 480 бит. Какова длина сообщения в символах?

Решение

При перекодировке из 16-битного кода Unicode в 8-битную кодировку КОИ-8 сообщение уменьшится вдвое. Следовательно, длина сообщения 480 бит или 60 символов.

Задача № 3

В велокроссе участвуют 119 спортсменов. Специальное устройство регистрирует прохождение каждым из участников промежуточного финиша, записывая его номер с использованием минимально возможного количества бит, одинакового для каждого спортсмена. Каков информационный объем сообщения, записанного устройством, после того как промежуточный финиш прошли 70 велосипедистов?

Решение

Для кодировки 119 номеров участников потребуется 7 бит информации:

$$119_{10} = 111011_{12}.$$

Устройство регистрации зафиксировало 70 участников, т.е. $7 \times 70 = 490$ бит информации.

Задача № 4

Перевести число $71{,}5_{(10)}$ в системы счисления с основаниями 2, 8 и 16.

Решение

Перевод смешанного числа, т.е. числа, у которого есть и целая и дробная часть, осуществляется поэтапно: сначала переводим целую, а затем дробную часть числа.

1. Перевод в двоичную систему счисления:

Сначала переведем целую часть нашего числа методом последовательного деления:

$$\begin{aligned} 71 : 2 &= 35 \text{ остаток } \underline{1} \\ 35 : 2 &= 17 \text{ остаток } \underline{1} \\ 17 : 2 &= 8 \text{ остаток } \underline{1} \\ 8 : 2 &= 4 \text{ остаток } \underline{0} \\ 4 : 2 &= 2 \text{ остаток } \underline{0} \\ 2 : 2 &= \underline{1} \text{ остаток } \underline{0} \end{aligned}$$

Получили, что число $71_{(10)} = 1000111_{(2)}$.

Теперь в двоичную систему счисления переведем дробную часть методом последовательного умножения:

$$0{,}5 \times 2 = 1{,}0.$$

Получили, что $0{,}5_{(10)} = 0{,}1_{(2)}$.

Ответ: $71{,}5_{(10)} = 1000111{,}1_{(2)}$.

2. Перевод в восьмеричную систему счисления.

Целая часть:

$$\begin{aligned} 71 : 8 &= 8 \text{ остаток } \underline{7} \\ 8 : 8 &= \underline{1} \text{ остаток } \underline{0} \end{aligned}$$

Получили, что число $71_{(10)} = 107_{(8)}$.

Дробная часть:

$$0{,}5 \times 8 = \underline{4}{,}0.$$

Получили, что $0{,}5_{(10)} = 0{,}4_{(8)}$.

Ответ: $71{,}5_{(10)} = 107{,}4_{(8)}$.

3. Перевод в шестнадцатеричную систему счисления.

Целая часть:

$$71 : 16 = \underline{4} \text{ остаток } \underline{7}$$

Значит, $71_{(10)} = 47_{(16)}$.

Дробная часть:

$$0{,}5 \times 16 = \underline{8}{,}0.$$

Получили, что $0{,}5_{(10)} = 0{,}8_{(16)}$.

Ответ: $71{,}5_{(10)} = 47{,}8_{(16)}$.

Задача № 5

Перевести десятичные числа 464, 380,1875 и 115,94 в двоичную систему счисления.

Ответ: $464_{(10)} = 111010000_{(2)}$;
$380{,}1875_{(10)} = 101111100{,}0011_{(2)}$;
$115{,}94_{(10)} = 1110011{,}11110_{(2)}$.

Задача № 6

Перевести числа $1000001_{(2)}$, $1001{,}01_{(2)}$, $1{,}01_{(2)}$, $1000011111{,}0101_{(2)}$. $1216{,}04_{(8)}$, $1234{,}56_{(8)}$, $29A{,}5_{(16)}$, $1A2{,}3C_{(16)}$ в десятичную систему счисления.

Решение

При переводе чисел из системы счисления с произвольным основанием в десятичную систему счисления необходимо пронумеровать разряды целой части справа налево, начиная с нулевого, и в дробной части, начиная с разряда сразу после запятой, слева направо (начальный номер — 1). Затем записать число в виде многочлена (полинома), представляющего собой сумму произведений соответствующих значений разрядов на основание системы счисления в степени, равной номеру разряда. Если в каком-либо разряде стоит ноль, то соответствующее слагаемое можно опускать. Вычислим сумму — это и есть представление исходного числа в десятичной системе счисления.

Осуществим перевод, представив исходные числа в виде полинома и вычислив его значение:

1) $1000001_{(2)} = 1 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 64 + 1 = 65_{(10)}$;
2) $1001{,}01_{(2)} = 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 + 0 \cdot 2^{-1} + 1 \cdot 2^{-1} = 8 + 1 + 0{,}25 = 9{,}25_{(10)}$;
3) $1{,}01_{(2)} = 1 \cdot 2^0 + 0 \cdot 2^{-1} + 0 \cdot 2^{-2} = 1 + 0{,}25 = 1{,}25_{(10)}$;
4) $1000011111{,}0101_{(2)} = 1 \cdot 2^9 + 1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 + 1 \cdot 2^{-2} + 1 \cdot 2^{-4} = 512 + 16 + 8 + 4 + 2 + 1 + 0{,}25 + 0{,}0625 = 543{,}3125_{(10)}$;
5) $1216{,}04_{(8)} = 1 \cdot 8^3 + 2 \cdot 8^2 + 1 \cdot 8^1 + 6 \cdot 8^0 + 4 \cdot 8^{-2} = 512 + 128 + 8 + 6 + 0{,}0625 = 654{,}0625_{(10)}$;
6) $1234{,}56_{(8)} = 1 \cdot 8^3 + 2 \cdot 8^2 + 3 \cdot 8^1 + 4 \cdot 8^0 + 5 \cdot 8^{-1} + 6 \cdot 8^{-2} = 668{,}71875_{(10)}$;
7) $29A{,}5_{(16)} = 2 \cdot 16^2 + 9 \cdot 16^1 + 10 \cdot 16^0 + 5 \cdot 16^{-1} = 512 + 144 + 10 + 0{,}3125 = 656{,}3125_{(10)}$;
8) $1A2{,}3C_{(16)} = 1 \cdot 16^2 + 10 \cdot 16^1 + 2 \cdot 16^0 + 3 \cdot 16^{-1} + 12 \cdot 16^{-2} = 418{,}234375_{(10)}$.

Задача № 7

Перевести число $1000111,1_{(2)}$ в восьмеричную и шестнадцатеричную системы счисления.

Решение

1) перевод в восьмеричную систему счисления двоичного числа осуществляется путем разбиения этого числа на группы по три цифры и замены каждой группы соответствующей ей восьмеричной цифрой:

$$\mathbf{1000111,1_{(2)} = 001000111,100 = 107,4_{(8)}};$$

2) перевод в шестнадцатеричную систему счисления двоичного числа осуществляется путем разбиения этого числа на группы по четыре цифры и замены каждой группы соответствующей ей шестнадцатеричной цифрой:

$$\mathbf{1000111,1_{(2)} = 01000111,1000 = 47,8_{(16)}}.$$

Задача № 8

Перевести из двоичной системы в шестнадцатеричную число $1111010101,11_{(2)}$.

Ответ: $0011\ 1101\ 0101,1100_{(2)} = 3D5,C_{(16)}$.

Задача № 9

Перевести числа $1234,56_{(8)}$ и $1A2,3C_{(16)}$ в двоичную систему счисления.

Решение

1) каждую цифру восьмеричного числа заменим соответствующей ей группой из трех двоичных цифр, затем перепишем это число, убрав незначащие нули:

$$\mathbf{1234,56_{(8)} = 0010\ 1001\ 1100,101110_{(2)} = 10\ 1001\ 1100,10111_{(2)}};$$

2) аналогично каждую цифру шестнадцатеричного числа заменим соответствующей ей группой из четырех двоичный цифр, затем перепишем это число, убрав незначащие нули:

$$\mathbf{1A2,3C_{(16)} = 0001\ 1010\ 0010,0011\ 1100_{(2)} = 110100010,001111_{(2)}}.$$

Задача № 10

Сложить два двоичных числа 101101_2 и 110011_2; $110100010,001111_{(2)}$ и $1010011100,10111_{(2)}$.

Решение

Сложение двоичных чисел производится по тем же правилам, что и в десятичной системе, т.е. вычисляется сумма значений одноимен-

ных разрядов и переноса из предыдущего разряда. Вычисления производятся в столбик.

1) $\mathbf{101101_{(2)} + 110011_{(2)} = 1100000_{(2)}}$

$$\begin{array}{r} \mathbf{101101} \\ +\mathbf{110011}; \\ \hline \mathbf{1100000} \end{array}$$

2) $\mathbf{110100010{,}001111_{(2)} + 1010011100{,}10111_{(2)} = 10000111110{,}111101_{(2)}}$

$$\begin{array}{r} \mathbf{110100010{,}001111} \\ +\mathbf{1010011100{,}101110}. \\ \hline \mathbf{10000111110{,}111101} \end{array}$$

Задача № 11

Вычесть поразрядно двоичные числа: $100110_{(2)} - 1010_{(2)}$; $100110{,}101_{(2)} - 110{,}11_{(2)}$.

Решение

При вычитании двоичных чисел в столбик необходимо помнить, что, выполняя вычитание из ноля единицы, следует занять единицу из старшего значащего разряда.

1) $\mathbf{100110_{(2)} - 1010_{(2)} = 11100_{(2)}}$

$$\begin{array}{r} \mathbf{100110} \\ -\mathbf{1010}; \\ \hline \mathbf{11100} \end{array}$$

2) $\mathbf{100110{,}101_{(2)} - 110{,}11_{(2)} = 11111{,}111_{(2)}}$

$$\begin{array}{r} \mathbf{100110{,}101} \\ -\mathbf{110{,}110}. \\ \hline \mathbf{11111{,}111} \end{array}$$

Задача № 12

Вычесть двоичные числа $100110_{(2)} - 1010_{(2)}$, рассматривая вычитание как сложение положительного числа с отрицательным числом.

Решение

1) $\mathbf{100110_{(2)} - 1010_{(2)} = 11100_{(2)}}$;
2) запишем числа в виде восьмиразрядных, т.е. так, как они записываются в ячейки памяти компьютера:

$$00100110_{(2)} - 00001010_{(2)};$$

3) получим дополнительный код вычитаемого 1010;
4) **00001010** — прямой код;

5) **11110101** — инверсный код;
6) $\begin{array}{r} \mathbf{11110101} \\ +\ \mathbf{00000001} \\ \hline \end{array}$ — прибавим к инверсному коду числа единицу;
7) **1110110** — дополнительный код;
8) найдем разность, сложив полученный дополнительный код числа с уменьшаемым и отбросив старший разряд полученного числа:

$$\begin{array}{r} \mathbf{00100110} \\ +\ \mathbf{11110110} \\ \hline \mathbf{100011100}. \end{array}$$

ГЛАВА 2

ОСНОВЫ АЛГЕБРЫ ЛОГИКИ

2.1. Формы мышления. Алгебра высказываний

Логика, как и теория алгоритмов — является теоретической основой современных ЭВМ и программирования. Слово «логика» в широком смысле означает науку о правилах рассуждений, а в узком смысле — совокупность правил, которым подчиняется процесс мышления.

Объектами изучения логики являются формы мышления: понятие, суждение и умозаключение.

Понятие — это мысль, в которой обобщаются отличительные свойства предметов.

Суждение (высказывание) — есть мысль (выраженная в форме повествовательного предложения), в которой нечто утверждается о предмете действительности, которая объективно является либо истинной, либо ложной. Суждение истинно, если оно соответствует действительности. Суждение, значение истинности которого неоднозначно, называется **гипотезой**. **Закон науки** — это суждение, истинность которого доказана.

Умозаключение — прием мышления, посредством которого из исходного знания получается новое знание; из одного или нескольких истинных суждений, называемых посылками, мы по определенным правилам вывода получаем заключение. Существуют умозаключения, осуществляемые по схемам аналогии, индукции и дедукции.

Формальная логика связана с анализом наших обычных содержательных умозаключений, выражаемых разговорным языком. Одной из частей формальной логики можно назвать математическую логику. Математическая логика изучает только умозаключения со строго определенными объектами и суждениями, для которых можно однозначно решить, истинны они или ложны, а также логические связи и отношения, лежащие в основе дедуктивного (логического) вывода.

Математическая логика является специальным математическим аппаратом, который лежит в основе работы логических схем

и устройств персонального компьютера. Математическая логика изучает вопросы применения математических методов для решения логических задач и построения логических схем. Знание логики необходимо при разработке алгоритмов и программ, так как в большинстве языков программирования есть логические операции.

Алгебру логики иначе называют алгеброй высказываний. Суждения и утверждения в математической логике называют высказываниями и предикатами. Высказывания — это конкретные частные утверждения. Предикаты — это утверждения о переменных, истинность предикатов зависит от значений входящих в них переменных. Пример высказываний: «5 + 7 = 12», «4 — четное число», пример предикатов: «x + y > 0», «N — число нечетное».

Алгебра высказываний позволяет определять истинность или ложность составных высказываний.

В алгебре высказываний простым высказываниям или суждениям соответствуют логические переменные. Истинному высказыванию соответствует значение логической переменной 1, а ложному — значение 0. Над высказываниями можно производить определенные логические операции, в результате которых получаются новые, составные высказывания.

Для образования новых высказываний наиболее часто используются базовые логические операции, выражаемые с помощью логических связок «и» (логическое умножение (конъюнкция)), «или» (логическое сложение (дизъюнкция)), «не» (логическое отрицание (инверсия)).

Конъюнкция. Операцию логического умножения (конъюнкцию) принято обозначать значком «&» либо «∧»:

$$F = X \& Y.$$

Функция логического умножения F может принимать лишь два значения «истина» (1) и «ложь» (0). Значение логической функции определяется с помощью таблицы истинности:

X	Y	$F = X \& Y$
0	0	0
0	1	0
1	0	0
1	1	1

Дизъюнкция. Операцию логического сложения обозначают «∨» либо «+».

$$F = X \vee Y.$$

Таблица истинности:

X	Y	$F = X \vee Y$
0	0	0
0	1	1
1	0	1
1	1	1

Инверсия. Операцию логического отрицания обозначают $F = \bar{A}$ или $F = \neg A$.

Таблица истинности логического отрицания:

A	$F = \bar{A}$
0	1
1	0

2.2. Логические выражения и функции

Алгебра логики явилась математической основой теории электрических и электронных переключателей схем, используемых в ЭВМ, поэтому ее предпочитают называть не алгеброй логики, а Булевой алгеброй — по имени ее создателя.

Суждения, истинность которых постоянна и не зависит от истинности входящих в них простых суждений, а определяется только их структурой, называются **тождественными** или **тавтологиями**.

Овладев этими основными свойствами суждений, можно упрощать формулы логики суждений уже формально, подобно тому, как в алгебре выполняются тождественные преобразования.

Применение алгебры суждений:

1) для упрощения сложных логических формул и доказательства тождеств;
2) при решении логических задач;
3) в контактных схемах;
4) при доказательстве теорем;
5) в базах данных при составлении запросов.

Логические выражения. Составные высказывания можно представить в виде логического выражения или формулы, состоящей из логических переменных, которые обозначают высказывания, и знаков логических операций.

Логические операции выполняются в следующем порядке: инверсия, конъюнкция, дизъюнкция. Скобки позволяют этот порядок изменить:

$$\mathbf{F = (A \vee B) \& (\neg A \vee \neg B)}.$$

Таблицы истинности можно построить для каждого логического выражения. Она определяет его значение при всех возможных комбинациях значений логических переменных.

Построение таблицы истинности:

1) количество строк N в таблице истинности равно количеству возможных комбинаций значений логических переменных n, и определяется по формуле: $N = 2$;
2) количество столбцов в таблице истинности равно количеству логических переменных плюс количество логических операций;
3) построить таблицу истинности с необходимым количеством строк и столбцов и записать значения исходных логических переменных;
4) заполнить таблицу истинности по столбцам в соответствии с таблицами истинности.

Равносильными логическими выражениями называются логические выражения, у которых совпадают последние столбцы таблиц истинности.

Составное высказывание можно рассматривать как некую **логическую функцию**. Логическая функция двух аргументов имеет четыре возможных набора исходных значений этих аргументов, т.е. существует 16 различных логических функций двух аргументов:

Аргументы		Логические функции															
A	B	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}	F_{16}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Логическое следование (импликация) — это логическая функция, которую можно описать помощью оборота «если..., то...» и которая обозначается $A \to B$.

Таблица истинности:

A	B	$A \to B$
0	0	1
0	1	1
1	0	0
1	1	1

Логическое равенство (эквивалентность) — это логическая функция, которую можно описать помощью оборота «тогда и только тогда, когда...» и обозначается $A - B$. Таблица истинности:

A	B	$A - B$
0	0	1
0	1	0
1	0	0
1	1	1

2.3. Логические законы

Закон тождества. Всякое высказывание тождественно самому себе:

$$\mathbf{A = A.}$$

Закон непротиворечия. Высказывание не может быть одновременно истинным и ложным:

$$\mathbf{A \,\&\, \overline{A} = 0.}$$

Закон исключенного третьего. Высказывание может быть либо истинным, либо ложным:

$$\mathbf{A \vee A = 1.}$$

Закон двойного отрицания. Двойное отрицание дает в итоге исходное высказывание:

$$\mathbf{\overline{\overline{A}} = A.}$$

Законы де Моргана:

$$\mathbf{\overline{A \vee B} = \overline{A} \,\&\, \overline{B}}$$
$$\mathbf{\overline{A \,\&\, B} = \overline{A} \vee \overline{B}}.$$

Закон коммутативности. Как и в алгебре: от перемены мест ...

$$\mathbf{A \,\&\, B = B \,\&\, A \;|\; A \vee B = B \vee A.}$$

Закон ассоциативности:

$$\mathbf{(A \,\&\, B) \,\&\, C = A \,\&\, (B \,\&\, C) \;|\; (A \vee B) \vee C = A \vee (B \vee C).}$$

Закон дистрибутивности. Отличается от подобного закона в алгебре — за скобки можно выносить не только общие множители, но и общие слагаемые:

$$\mathbf{(A \,\&\, B) \vee (A \,\&\, C) = A \,\&\, (B \vee C),}$$
$$\mathbf{(A \vee B) \,\&\, (A \vee C) = A \vee (B \,\&\, C).}$$

2.4. Примеры решения задач

Задача № 1

Упростить формулу $(A \vee B) \wedge (A \vee C)$.

Решение

$$(A \vee B) \wedge (A \vee C) = A \wedge A \vee A \wedge C \vee B \wedge A \vee B \wedge C,$$

так как $A \wedge A = A$, следовательно

$$A \wedge A \vee A \wedge C \vee B \wedge A \vee B \wedge C = A \vee A \wedge C \vee B \wedge A \vee B \wedge C,$$

у первых двух слагаемых вынесем за скобки общий множитель:

$$A \vee A \wedge C \vee B \wedge A \vee B \wedge C = A \wedge (1 \vee C) \vee B \wedge A \vee B \wedge C,$$

так как $(1 \vee C) = 1$, получим

$$A \vee B \wedge A \vee B \wedge C,$$

еще раз вынесем за скобки A

$$A \wedge (1 \vee B) \vee B \wedge C,$$

так как $(1 \vee B) = 1$, получим

$$(A \vee B) \wedge (A \vee C) = A \vee (B \wedge C).$$

Задача № 2

Упростить логическое выражение:

$$(A \wedge B) \vee (\neg A \wedge B).$$

Решение

По закону дистрибутивности вынесем за скобки B:

$$(A \wedge B) \vee (\neg A \wedge B) = B \wedge (A \vee \neg A).$$

Так как

$$(A \vee \neg A) = 1,$$

тогда получим:

$$(A \wedge B) \vee (\neg A \wedge B) = B \wedge 1 = B.$$

Задача № 3

Преобразуйте логическое выражение: $A \wedge \neg(\neg B \vee C)$.

Решение

$$A \wedge \neg(\neg B \vee C) = A \wedge \neg\neg B \wedge \neg C,$$

так как $\neg\neg B = B$, то получим

$$A \wedge \neg\neg B \wedge \neg C = A \wedge B \wedge \neg C.$$

Задача № 4

Для какого из указанных значений X (1, 2, 3, 4) истинно высказывание

$$\neg((X > 2) \rightarrow (X > 3))?$$

Решение

Составим таблицу, в первом столбце которой укажем все возможные варианты X, во втором столбце — значение логического выражения $X > 2$ (1 — истина, 0 — ложь), в третьем $X > 3$, в четвертом выполняем импликацию 2-го и 3-го столбцов, а в последнем — инверсию 4-го столбца.

X	$X > 2$	$X > 3$	$(X > 2) \rightarrow (X > 3)$	$\neg((X > 2) \rightarrow (X > 3))$
1	0	0	1	0
2	0	0	1	0
3	1	0	0	1
4	1	1	1	0

Таким образом, высказывание $\neg((X > 2) \rightarrow (X > 3))$ истинно только тогда, когда $X = 3$.

Задача № 5

Каково наибольшее целое число X, при котором истинно высказывание

$$(50 < X \cdot X) \rightarrow (50 > (X + 1) \cdot (X + 1))?$$

Решение

Вспомним таблицу истинности операции «импликация»:

A	B	$A \rightarrow B$
0	0	1
0	1	1
1	0	0
1	1	1

Проверим истинность высказывания при $A = 0$ и $B = 0$.

Если $X^2 > 50$ ложно, значит $X^2 <= 50$, а это возможно при $X <= 7$. А так как высказывание $(X + 1)^2 < 50$ также ложно, то получается, что $(X + 1)^2 >= 50$, то есть $X = 7$.

Задача № 6

Сколько различных решений имеет уравнение

$$((K \vee L) \rightarrow (L \wedge M \wedge N)) = 0,$$

где K, L, M, N — логические переменные?

Решение

Из таблицы истинности операции «импликация» следует, что равенство

$$((K \vee L) \to (L \wedge M \wedge N)) = 0$$

верно тогда, когда одновременно

$$K \vee L = 1 \text{ и } L \wedge M \wedge N = 0.$$

Первое уравнения верно, если 1) K = 1, L = 0; 2) K = 0, L = 1; 3) K = 1, L = 1.

Рассмотрим три случая:

1) если K = 1 и L = 0, то второе равенство выполняется при любых M и N; таких комбинаций существует **4**;
2) если K = 0 и L = 1, то второе равенство выполняется при $M \wedge N = 0$; таких комбинаций существует **3**;
3) если K = 1 и L = 1, то второе равенство выполняется при $M \wedge N = 0$; таких комбинаций существует, как мы уже определили, **3**.

Следовательно, таким образом, всего решений получаем

$$4 + 3 + 3 = 10.$$

Задача № 7

Классный руководитель пожаловался директору, что у него в классе появилась компания из трех учеников, один из которых всегда говорит правду, другой всегда лжет, а третий говорит через раз то ложь, то правду. Директор знает, что их зовут Коля, Саша и Миша, но не знает, кто из них правдив, а кто — нет. Однажды все трое прогуляли урок астрономии. Директор знает, что никогда раньше никто из них не прогуливал астрономию. Он вызвал всех троих в кабинет и поговорил с мальчиками. Коля сказал: «Я всегда прогуливаю астрономию. Не верьте тому, что скажет Саша». Саша сказал: «Это был мой первый прогул этого предмета». Миша сказал: «Все, что говорит Коля, — правда». Директор понял, кто из них кто. Определите, кто из мальчиков говорит всегда правду, всегда лжет, говорит правду через раз.

Решение

Нам достоверно известно, что все трое прогуляли урок астрономии в первый раз.

Запишем высказывания мальчиков.

Коля: 1. Я всегда прогуливаю астрономию.
2. Саша врет.

Саша: 1. Я в первый раз прогулял астрономию.

Миша: 1. Коля говорит правду.

Первое высказывание Коли — ложь, Саша сказал правду, следовательно и второе высказывание Коли — ложь. Делаем вывод, что Коля всегда лжет. Следовательно, высказывание Миши — ложь, Миша говорит правду через раз. Получается, что Саша всегда говорит правду.

Задача № 8

Перед началом «Турнира четырех» болельщики высказали следующие предположения по поводу своих кумиров:

A) Макс победит, Билл — второй;

B) Билл третий, Ник — первый;

C) Макс — последний, а первый — Джон.

Когда соревнования закончились, оказалось, что каждый из болельщиков был прав только в одном из своих прогнозов. Какое место на турнире заняли Джон, Ник, Билл, Макс?

Решение

Запишем высказывания трех болельщиков в форме таблицы:

	A	B	C
1	Макс	Ник	Джон
2	Билл		
3		Билл	
4			Макс

Если предположим, что Макс действительно занял первое место, как и сказал «А»; в этом случае

- С ошибся, поставив на первое место Джона;
- ошибиться С может только один раз, тогда получается, что С угадал, что Макс будет на четвертым;
- значит, в своем первом прогнозе А ошибся и на втором месте — Билл;
- В угадал победу Ника;
- следовательно, места распределились следующим образом:

1 — Ник; 2 — Билл; 3 — Джон; 4 — Макс.

Задача № 9

Алеша, Боря и Гриша нашли в земле сосуд. Каждый из них высказал по два предположения.

Алеша: «Это сосуд греческий, V в.».

Боря: «Это сосуд финикийский, III в.».

Гриша: «Это сосуд не греческий, IV в.».

Учитель истории сказал ребятам, что каждый из них прав только в одном из двух своих предположений. Где и в каком веке был изготовлен сосуд?

Решение:

	Алеша	Боря	Гриша
III		фин	
IV			фин
V	греч		

Так как и Боря, и Гриша назвали сосуд финикийским, но указали разные даты (а ошибка может быть только одна), то делаем вывод, что сосуд изготовлен в Финикии в V в.

ГЛАВА 3

СОСТАВ И РАБОТА ПЕРСОНАЛЬНОГО КОМПЬЮТЕРА

3.1. Аппаратное обеспечение персонального компьютера

Функциональная схема компьютера

Назначение компьютера — перерабатывать, хранить и выдавать информацию посредством выполнения команд пользователя.

Центральное устройство ПК, производящее все вычислительные операции и преобразующее информацию, называется **центральным процессором**. Характеристиками микропроцессора служат тактовая частота и разрядность.

Разрядность процессора определяет размер машинного слова, обрабатываемого компьютером. Машинное слово — число бит, к которым процессор имеет одновременный доступ. С увеличением размера слова увеличивается объем информации, обрабатываемой процессором за один такт, что ведет к уменьшению количества тактов, необходимых для выполнения сложных операций. Чем больше размер слова, тем с большим объемом памяти может работать компьютер. Микропроцессор обрабатывает информацию трех типов: данные, адреса и команды программ. Он оперирует с информацией, представленной в виде машинных кодов, длиной от 8 до 64 бит. Микропроцессор выполняет сотни различных операций со скоростью до нескольких сотен миллиардов в секунду. Разработка новых типов микропроцессоров позволяет постоянно увеличивать эту цифру.

Микропроцессор выполняет в основном функции двух устройств:

1) устройство управления, которое формирует и подает во все блоки ПК в нужные моменты времени определенные сигналы (управляющие импульсы), формирует адреса ячеек памяти, используемых выполняемой операцией, и передает их в другие блоки компьютера;
2) арифметико-логическое устройство, предназначенное для выполнения арифметических и логических операций над информацией.

Процесс обработки информации компьютером осуществляется дискретно, т.е. элементарные операции выполняются последовательно с определенной скоростью, зависящей от частоты импульсов, получаемых процессором от генератора тактовых импульсов. Эта характеристика, называемая ***тактовой частотой***, определяет производительность ПК, его ***быстродействие*** (количество операций, выполняемых в единицу времени). Тактовая частота задает ритм жизни компьютера. Чем выше тактовая частота, тем меньше длительность выполнения одной операции и тем выше производительность компьютера. Производительность компьютера — это количество элементарных операций, выполняемых процессором за одну секунду. Тактовая частота определяет число тактов работы процессора в секунду. Под тактом понимают промежуток времени, в течение которого может быть выполнена элементарная операция. Современный персональный компьютер может выполнять сотни миллиардов элементарных операций в секунду. Тактовую частоту можно измерить и определить ее значение. Для этого используется единица измерения частоты — гигагерц (ГГц — миллиард тактов в секунду).

Для введения и выведения информации служат устройства **ввода-вывода** (клавиатура, монитор, принтер и др.).

Для хранения информации служит **запоминающее устройство**. Основная память ПК состоит из оперативного запоминающего устройства (ОЗУ) и постоянного запоминающего устройства (ПЗУ).

Оперативная память (RAM — Random Access Memory — память с произвольным доступом) является очень важным элементов компьютера. В ней хранятся программы и данные, с которыми непосредственно работает ПК. Основу ОЗУ составляют большие интегральные схемы (БИС), содержащие матрицы полупроводниковых элементов. Структурно ОЗУ состоит из миллиардов отдельных ячеек памяти емкостью 1 байт каждая. Поэтому основной характеристикой оперативной памяти является ее объем, измеряемый в байтах. Его величина определяет перечень программ, которые можно использовать при работе ПК. Название «оперативное» обусловлено быстротой чтения и записи данных в память ПК. При выключении компьютера содержимое ОЗУ стирается, т.е. вся несохраненная информация теряется.

Постоянная память (ROM — Read-Only Memory — память только для чтения) используется для хранения неизменяемой информации: загрузочных программ и операционной системы, программ тестирования устройств компьютера и выполнения базовых функций по их обслуживанию. Большая часть этих программ связана с обслуживанием процессов ввода-вывода, поэтому содержимое ПЗУ часто называют

BIOS (Base Input-Output System или базовая система ввода-вывода). ПЗУ в отличие от ОЗУ является энергонезависимым (информация при выключении компьютера сохраняется).

Каждая ячейка основной памяти имеет свой адрес. Основная память имеет для ОЗУ и ПЗУ единое адресное пространство — совокупность ячеек памяти, к которым можно обращаться с использованием машинного адреса.

Аппаратные средства персонального компьютера

Персональный компьютер (ПК) — это сложная система взаимосвязанных аппаратных (технических) и программных средств, способных принимать, хранить, перерабатывать и выдавать информацию по определенному алгоритму или программе.

Под ***аппаратными средствами*** понимается оборудование ПК, участвующее в автоматической обработке данных.

Для выполнения процесса обработки данных в состав компьютера входят различные устройства, выполняющие определенные функции. Эти устройства принято разделять на ***центральные*** и ***периферийные*** (устройства, осуществляющие связь с различными источниками и получателями информации).

С помощью периферийных устройств осуществляется связь с различными источниками (поставщиками) и получателями (потребителями) информации.

К периферийным устройствам относятся устройства ввода-вывода и внешняя память.

К аппаратным средствам ввода информации в ПК относятся:

1) клавиатура — устройство ввода текста, чисел и управляющей информации в основную память ПК;
2) устройства ввода алфавитно-цифровой, графической и речевой информации (например, манипулятор «мышь», сканер и др.).

К аппаратным средствам вывода информации из ПК относятся:

1) принтер, графопостроитель — устройства для регистрации (печати) информации на бумажном носителе;
2) видеомонитор (дисплей) — устройство для отображения визуальной информации (текстов, чисел, графических изображений и т.д.);
3) синтезаторы речи — устройство для вывода звуковой информации.

Связь всех функциональных устройств ПК определяется **интерфейсами**. Интерфейс — это совокупность правил и средств, устанавлива-

ющих единые принципы взаимодействия устройств ПК. Например, интерфейс периферийного устройства включает в себя: техническое устройство, наборы передаваемых сигналов и правила обмена информацией с центральным устройством.

Системный интерфейс персонального компьютера — система связи и сопряжения узлов и блоков ПК между собой. Она представляет собой совокупность электрических линий связи (проводов), схем сопряжения с устройствами компьютера, протоколов (правил) передачи и преобразования сигналов.

В большинстве современных ПК в качестве системного интерфейса используется системная шина.

Системная (общая) шина — это канал передачи электрических сигналов для обмена информацией между основными и дополнительными устройствами компьютера. Она обеспечивает три направления передачи информации:

1) между микропроцессором и основной памятью;
2) между микропроцессором и портами ввода-вывода внешних устройств;
3) между основной памятью и портами ввода-вывода внешних устройств в режиме прямого доступа.

Управление работой системной шины осуществляется микропроцессором. Важнейшими характеристиками системной шины являются:

- количество обслуживаемых ею устройств;
- пропускная способность (скорость передачи информации).

Пропускная способность зависит от тактовой частоты, на которой работает шина, и от разрядности шины. Разрядность — это максимальное количество разрядов двоичного числа, над которым одновременно производится элементарная машинная операция, в том числе и операция передачи информации. Чем больше разрядность, тем при прочих равных условиях будет больше производительность. Например, 64-разрядная шина за один такт передает два 32-разрядных числа, а 32-разрядная шина — одно.

Все устройства ПК объединены внутрисистемным интерфейсом и взаимодействуют по адресному принципу: все подчиненные устройства и их составные части имеют конкретные адреса, по которым к ним обращаются устройства управления работой компьютера. Для связи процессора с периферийными устройствами используются специальные устройства сопряжения и обмена, называемые адаптерами (каналами). Через эти каналы осуществляется передача информации между отдельными частями ПК. Существует общий канал, через кото-

рый взаимодействуют все входящие в состав ПК устройства, называемый системной шиной.

Персональные компьютеры появились в начале 1980-х гг. в США, когда появилась необходимость иметь ЭВМ, не занимающую много места, которая была бы проста в использовании и доступна по цене любому человеку и имела бы большие возможности.

Персональная ЭВМ включает в состав следующие основные устройства:

- системный блок;
- монитор;
- клавиатуру.

Компьютеры выпускаются и в портативном варианте — в «наколенном» (лэптоп) или «блокнотном» (ноутбук) исполнении. Здесь системный блок, монитор и клавиатура заключены в один корпус: системный блок спрятан под клавиатурой, а монитор сделан как крышка к клавиатуре. В настоящее время широкое распространение получили планшетные компьютеры и нетбуки. **Нетбук** — это компактный ноутбук с относительно невысокой производительностью, его предназначение — выход в Интернет и работа с офисными приложениями. Обладает небольшой диагональю экрана 7—12 дюймов, низким энергопотреблением и небольшим весом. **Планшетные персональные компьютеры** представляют собой разновидность ноутбуков. Они оборудованы сенсорным экраном и позволяют работать при помощи стилуса или пальцев как с использованием, так и без использования клавиатуры и мыши.

Системный блок конструктивно состоит из набора определенных устройств, собранных в едином корпусе. В нем располагаются важнейшие устройства ПК:

- процессор и блоки памяти;
- жесткий магнитный диск (винчестер);
- блок питания;
- дополнительные устройства (модем, дисковод для CD-DVD-дисков, звуковая карта и др.)

На корпусе системного блока в доступном для пользователя месте (как правило, на лицевой стороне) расположены по меньшей мере две кнопки или клавиши (включения и перезагрузки компьютера), а также световые сигнальные индикаторы. На задней панели системного блока располагаются разъемы для подключения электропитания и для связи с другими устройствами. Разъемы конструктивно выполнены таким образом, что их невозможно перепутать при подключении соединительных кабелей.

Устройства внешней памяти или внешние запоминающие устройства (ВЗУ) предназначены для долговременного и энергонезависимого хранения информации. Они различаются по виду носителя информации, типу конструкции, принципу записи и считывания информации и т.д. Основу любого ВЗУ составляет носитель информации — материальный объект, способный хранить информацию.

Жесткие магнитные диски или **винчестеры** (устройства хранения информации произвольного доступа, основанные на принципе магнитной записи) являются обязательным компонентом персонального компьютера.

Лазерные диски. Лазерные дисководы используют оптический принцип чтения информации. На лазерных дисках CD (CD — Compact Disk, компакт диск) и DVD (DVD — Digital Video Disk, цифровой видеодиск) информация записана на одну спиралевидную дорожку (как на грампластинке), содержащую чередующиеся участки с различной отражающей способностью. Лазерный луч падает на поверхность вращающегося диска, а интенсивность отраженного луча зависит от отражающей способности участка дорожки и приобретает значения 0 или 1.

На диски CD-R и DVD-R информация может быть записана только один раз. На дисках CD-RW и DVD-RW информация может быть записана/перезаписана многократно.

Для работы с дисками (чтение и запись информации) используются специальные устройства, которые называются **дисководами**.

Сейчас уже получили широкое распространение пишущие лазерные дисководы, которые позволяют не только читать информацию с диска, но и записывать ее на лазерный диск.

Флэш-память. Носители на ее основе называются твердотельными, поскольку не имеют движущихся частей. Многие производители вычислительной техники видят память будущего исключительно твердотелой.

Устройства ввода информации

Основным устройством ввода информации в ПК является **клавиатура**. При нажатии на любую клавишу срабатывает миниатюрный переключатель, сигнал от которого отслеживается специальным микропроцессором, посылающим соответствующие сообщения в компьютер, где они обрабатываются операционной системой.

Для управления работой современных программ используются различные **манипуляторы**. Манипуляторы осуществляют непосредственный ввод информации, указывая курсором на экране монитора

команду или место ввода данных. Манипуляторы, как правило, подключаются к коммуникационному порту.

Джойстик представляет собой ручку управления и наиболее часто используется в компьютерных играх. Джойстики управляют перемещениями курсора по экрану.

Мышь — наиболее распространенный вид манипулятора. Движение мыши отражается на экране монитора перемещением ее указателя.

Трекбол (шаровой манипулятор) — это шар, расположенный на поверхности клавиатуры вместе с кнопками. Перемещение указателя по экрану обеспечивается вращением шара.

Сенсорные устройства ввода. Принцип ввода данных в сенсорных устройствах аналогичен принципу ввода в манипуляторах-координаторах.

Сенсорный манипулятор — класс координатных устройств — представляет собой коврик без мыши. В данном случае управление курсором производится простым движением пальца по коврику. *Сенсорный экран* — представляет собой поверхность, которая покрыта специальным слоем. Это устройство дает возможность выбирать действие или команду, дотрагиваясь до экрана пальцем. Такими устройствами ввода пользуются в банковских компьютерах, аэропортах, а также в военной сфере и промышленности.

Световое перо имеет светочувствительный элемент на своем кончике. Соприкосновение пера с экраном замыкает фотоэлектрическую цепь и определяет место ввода или коррекции данных. Световое перо используется в различных системах проектирования и дизайна.

Графический планшет, дигитайзер пользуется для ввода в компьютер чертежей или рисунков. Условия создания изображения приближены к реальным, достаточно специальным пером или пальцем сделать рисунок на специальной поверхности. Результаты работы дигитайзера воспроизводятся на экране монитора и в случае необходимости могут быть распечатаны на бумаге. Дигитайзерами пользуются архитекторы, дизайнеры.

Устройства сканирования. В этих устройствах изображение преобразуется в цифровую форму для дальнейшей обработки компьютером или воспроизведения на экране монитора.

Сканеры находят широкое применение в издательской деятельности, системах проектирования, анимации, а также при создании иллюстративных материалов для презентаций, докладов, рекламы.

К *устройствам распознавания символов* относятся, например, терминалы больших универмагов, они оснащены разнообразными устройствами считывания штрихкодов, специальных символов и ме-

ток для определения условий приобретения товара. Считанная информация преобразуется, выводится на экран или бумажный чек и по линиям связи передается на главный, более мощный компьютер.

Устройства распознавания речи: с помощью микрофона речь человека вводится в компьютер и преобразуется в цифровой код. Большинство систем распознавания речи могут быть настроены на особенности человеческого голоса. Это реализуется путем сравнения сказанного слова с образцами, записанными в памяти компьютера. Некоторые системы могут определять одинаковые слова, сказанные разными людьми. Системы распознавания речи находят широкое применение в сфере образования.

Устройства вывода информации

Монитор (дисплей) — это устройство для отображения информации в визуальной форме.

Монитор получает видеосигнал в готовом виде от видеоконтроллера, расположенного в системном блоке.

Для вывода данных из ПК на бумажный носитель информации используются специальные печатающие устройства — **принтеры**.

Современные принтеры позволяют выводить на печать текстовую информацию, а также рисунки и графики.

По способу получения изображения на бумаге, способу нанесения красящего материала принтеры бывают: матричные, струйные, лазерные, термические, литерные.

Матричные принтеры относятся к ударным печатающим устройствам. Изображение формируется с помощью иголок, ударяющих по бумаге через красящую ленту.

Струйные принтеры относятся к безударным устройствам, так как головка печатающего устройства не касается бумаги. Для получения изображения используют чернила. Головка принтера представляет собой чернильницу, в которой из дырочек-сопел выбрасываются тонкие струи чернил. В отличие от матричных струйные принтеры работают почти бесшумно и обеспечивают лучшее качество печати, особенно цветной.

Лазерные принтеры для формирования изображения используют лазерный луч. С помощью систем линз тонкий луч лазера формирует скрытое электронное изображение на светочувствительном барабане. К заряженным участкам электронного изображения притягиваются частички порошка-красителя, который затем переносится на бумагу. Закрепление изображения на бумаге осуществляется разогревом тоне-

ра до температуры плавления. Лазерные принтеры обеспечивают наилучшее качество и высокую скорость печати, но являются наиболее дорогими.

Рассмотрим назначение и принципы действия некоторых дополнительных устройств.

Модем — это устройство для обмена информацией между компьютерами с использованием телефонной сети. Свойства модема определяются большим числом специфических характеристик, отраженных в его маркировке. Основными характеристиками модемов являются:

- скорость передачи данных;
- конструктивное исполнение (внешнее и внутреннее);
- принцип обмена информацией (односторонний или в обоих направлениях).

Факс-модем — это устройство, сочетающее возможности модема и средства для обмена факсимильными изображениями с другими факс-модемами и обычными телефаксными аппаратами.

Саундбластер или звуковая карта позволяет осуществлять запись и воспроизведение звукового сигнала.

К звуковой плате подключается микрофон, акустические колонки и DVD-дисковод. Существует множество моделей звуковых карт, обладающих различными возможностями.

Видеобластер или видеокарта используется в ПК для вывода статичных и подвижных изображений на экран монитора. Она позволяет обрабатывать изображение, поступающее с видеокамеры, видеомагнитофона или телевизора. К видеокарте может быть подключен микрофон и акустические системы.

Графопостроитель (или плоттер) — это чертежная машина, позволяющая с высокой точностью и скоростью вычерчивать сложные графические изображения: чертежи, схемы, карты, графики и т.д. Плоттеры бывают барабанного (работают с рулоном бумаги) и планшетного типа (лист бумаги закреплен на рабочей плоскости).

Выше приведены краткие описания только части периферийных устройств ПК. На практике их значительно больше и постоянно появляются новые. Часть из них находит широкое применение на практике, часть используется для решения специфических проблем или в специальных областях человеческой деятельности. (Например, дигитайзер, цифровая фотокамера, электронное перо и др.).

Мультимедиа компьютер — это компьютер с мощным процессором, большой оперативной памятью и жестким диском, имеющий звуковую карту с микрофоном и колонками, DVD-дисковод (или Blu ray-проигрыватель), и к которому можно подключить сканер, графиче-

ский планшет и принтер. Мультимедиа компьютер позволяет работать с мультимедиа технологиями.

Термин мультимедиа (от английского слова *multimedia*), можно перевести как многие среды (от *multi* — много и *media* — среда).

Мультимедиа технология позволяет интегрировать различные виды представления и обработки информации.

3.2. Программное обеспечение персонального компьютера

Персональный компьютер не может работать без программного обеспечения. Для его работы необходим комплекс различных программ. В зависимости от назначения программное обеспечения ПК можно разделить на три группы:

- системное обеспечение;
- системы программирования;
- прикладное программное обеспечение.

Системное обеспечение ПК состоит из операционных систем и средств контроля и диагностики.

Операционная система (ОС) — это совокупность программ, обеспечивающих управление процессом обработки информации и взаимодействие между аппаратными средствами и пользователем. Операционная система — программа, которая загружается при включении компьютера. Она осуществляет диалог с пользователем, управление компьютером, его ресурсами, запускает другие программы на выполнение. Операционная система обеспечивает пользователю и прикладным программам удобный способ общения (интерфейс) с устройствами компьютера.

Драйверы. Важным классом системных программ являются программы — драйверы. Они расширяют возможности операционной системы по управлению устройствами ввода — вывода компьютера (клавиатурой, жестким диском, мышью и т.д.). С помощью драйверов возможно подключение к компьютеру новых устройств или нестандартное использование имеющихся устройств.

К системным программам можно также отнести большое количество утилит, т.е. программ вспомогательного назначения.

Программы — архиваторы (упаковщики) позволяют за счет применения специальных методов «упаковки» информации сжимать информацию на дисках, т.е. создавать копии файлов меньшего размера.

Существует много программ-архиваторов, имеющих различные показатели по степени и времени сжатия, эти показатели могут быть разными для различных файлов (текстовых, графических, исполняемых и т.д.), т.е. один архиватор хорошо сжимает текстовый файл, а другой — исполняемый. Среди самых известных и часто используемых программ выделяются следующие: ARJ, PKZIP, RAR и др.

Антивирусные программы. Компьютерный вирус — это специально написанная небольшая по размерам программа, которая может «приписывать» себя к другим программам («заражать» их), а также выполнять различные нежелательные действия на компьютере. Пока на компьютере заражено относительно мало программ, наличие вируса может быть практически незаметно. Но по прошествии некоторого времени на компьютере начинает твориться что-то странное (некоторые программы перестают работать; на экране вводятся посторонние сообщения, символы; работа на компьютере существенно замедляется и т.д.).

Для эффективной борьбы с многочисленными вирусами создаются антивирусные программы:

- программы-детекторы обнаруживают файлы, зараженные вирусом;
- программы-доктора, или фаги — «лечат» программы, восстанавливая их первоначальный вид и удаляя при этом из них вирус;
- программы-фильтры — перехватывают обращения вирусами к операционной системе, используемые для размножения и нанесения вреда, и сообщают о них пользователю.

Разработка антивирусных программ требует профессиональных знаний и навыков. К наиболее известным антивирусным программам относятся периодически обновляемые и дополняемые программы КАСПЕРСКИЙ и DRWEB.

Программы — кэши для диска убыстряют доступ к информации на диске путем организации в оперативной памяти кэш-буферов, содержащих наиболее часто используемые участки диска. Чаще всего для кэша используется дополнительная или расширенная память компьютера.

Разумеется, многообразие вспомогательных программ для IBM отнюдь не исчерпывается описанными выше типами программ.

Системы программирования включают в себя языки программирования и трансляторы и позволяют разрабатывать как системное, так и прикладное программное обеспечение. Современные системы программирования для ПК обычно предоставляют пользователю весьма мощные и удобные средства для разработки программ. В них входят:

- компилятор, осуществляющий преобразования программ на языке программирования в программу в машинных кодах; или интерпретатор, осуществляющий непосредственно выполнение текста программы на языке программирования высокого уровня;
- библиотеки программ, содержащие заранее подготовленные программы, которыми могут пользоваться программисты;
- различные вспомогательные программы.

Системы программирования различаются по тому, какой язык программирования они реализуют.

Языки программирования — это специально созданные языки для описания алгоритмов обработки данных на ЭВМ. В настоящее время их насчитывается несколько сотен. На языках программирования разрабатываются программы для ПК.

В последнее время приобретают популярность визуальные оболочки для языков, которые позволяют работать с ними даже непрофессионалу в области программирования.

Программа — это совокупность команд, записанных в соответствии с принятым синтаксисом и управляющих действиями компьютера. Для того чтобы программа могла быть исполнена компьютером, исходный модуль программы, написанной на языке программирования, необходимо перевести в совокупность машинных команд или в объектный модуль. Роль такого переводчика выполняют трансляторы.

Транслятор — переводчик с языка программирования на язык ЭВМ, т.е. в команды, состоящие из машинных кодов. Различают основные виды трансляторов: интерпретаторы и компиляторы.

Интерпретатор — транслятор, который обеспечивает покомандный перевод в машинные коды с одновременным их исполнением. Достоинством интерпретатора является возможность организации работы в режиме диалога «пользователь — компьютер». Недостаток состоит в низкой скорости выполнения программы.

Компилятор — транслятор, который переводит всю программу целиком в машинные коды без ее выполнения. В результате работы компилятора создается отдельный модуль, еще не готовый к выполнению. Для выполнения программы необходимо создать загрузочный (выполняемый) модуль, который может включать несколько объектных модулей, необходимых для выполнения программы.

Прикладное программное обеспечение — совокупность программ различного назначения для автоматизации обработки различного рода информации и выполнения расчетов в разных сферах деятельности человека.

Прикладные программы — специальные программы, которые разрабатываются для конкретных пользователей или самими пользователями для решения определенного круга задач.

Программы для дизайна применяются для построения графических объектов. К ним можно отнести такие программы, как Adobe Photoshop, Adobe Illustrator и др. Данные программные продукты позволяют пользователю компьютера создавать различные плоские и трехмерные рисунки, а также обрабатывать фотографии и отсканированные изображения в режиме реального времени и при этом видеть каждый этап работы.

Коммуникационные программы позволяют использовать ресурсы Интернета, дают возможность общения с другими пользователями на уровне текстовых сообщений, аудио- и видеосигнала. Эти программы можно разделить на несколько подвидов:

браузеры — программы, которые отражают содержание ресурсов в компьютерных сетях, могут быть построены на технологии HTML, FTP или Telnet. Наиболее известны Microsoft Internet Explorer, Opera, Google Chrome;

почтовые программы применяются для пересылки текстовых сообщений (писем) и любых файлов пользователями компьютерных сетей. Например, программы Microsoft Outlook Express, The Bat!, Eudora;

коммуникационные программы позволяют пользователям общаться посредством текстовых сообщений (ISQ, AOL Instant Messenger, Yahoo! Pager), потокового аудио (Microsoft Net Meeting) или потокового видео (Quick Movie).

Служебные программы применяются каждым грамотным пользователем. Некоторые из них помогают следить за состоянием «здоровья» компьютера (Reestr Cleaner), вести дела (Display Notes), менять текущий язык на компьютере (Snoop!), отслеживать трафик Интернет-соединений (NetStat) и правильность перекачки файлов из Интернета (ReGet).

Мультимедийные программы сочетают в себе наличие видеофрагментов и звука, статических картинок и гипертекста. Эти программы применяются для создания компьютерных собраний коллекций музыкальных произведений, музеев искусств, видеофильмов, обучающих программ.

Самым распространенным классом мультимедийных программ являются компьютерные игры — сетевые и несетевые.

Программы для корпоративного пользования можно разделить на несколько групп, не зависящих функционально друг от друга, но применяющихся для одной цели — обеспечения работоспособности от-

дельно взятой организации (хотя некоторые из них применяются и в домашних условиях).

К ним относятся программные продукты автоматизации предприятия, офисные программы для делопроизводства, программы автоматизации бухгалтерии и документооборота, программы-переводчики. Наиболее известными программами в этой области являются программы пакета Microsoft Office.

Пакет Microsoft Office содержит большую часть необходимого программного обеспечения: в него входят и текстовый редактор, и программа для работы с электронными таблицами, и программа деловой графики. Самое ценное в этом пакете то, что элементы интерфейса входящих в него программ оформлены одинаково и все эти программы «понимают» друг друга, что очень важно при передачи данных из одной программы в другую: например, вставить диаграмму в текст или презентацию и т.п.

Название пакета Office подсказывает, что он содержит мощные прикладные программы для коммерческого применения, которые значительно облегчают работу с текстами, числами, таблицами и изображениями. Практически любую работу в офисе небольшой фирмы можно выполнить, используя этот пакет. В его состав входят:

1) **текстовый редактор WORD**;
2) **электронные таблицы EXCEL**;
3) **СУБД Access**;
4) **программа создания презентаций PowerPoint**;
5) **Outlook — настольная информационная система**;
6) **Publisher** — средство подготовки печатной продукции;
7) **Visio** — независимая система построения диаграмм, предлагающая средства для наглядного представления идей, информации и систем;
8) **Frontpage** — средство для создания, поддержки и развития вебсайтов.

Версия Microsoft Office 2007.

В Word, Excel, PowerPoint, Access и некоторых областях Outlook место меню и панелей заняли ленты — горизонтальные панели в виде закладок, разделенные в группы согласно выполняемым ими задачам.

Приложения Office 2007 хранят файлы документов в одном сжатом, Zip-совместимом формате. Такое изменение упрощает процедуру восстановления файла в случае его повреждения. В Word формат имеет расширение — .DOCX в Excel — .XLSX, а в PowerPoint соответственно. PPTX.

3.3. Назначение и основные функции операционной системы; файловая система

Операционная система

Компьютер — это сложная электронно-механическая система, состоящая из двух участников — аппаратного и программного обеспечения, в персональном компьютере можно указать и третьего участника — человека-пользователя. Взаимосвязь между участниками системы называется интерфейсом. Различают следующие виды интерфейса:

- аппаратный — взаимодействие между узлами компьютера;
- программный — взаимодействие между программами;
- аппаратно-программный — взаимодействие между аппаратурой и программами;
- пользовательский — взаимодействие пользователя с программой.

Интерфейс, т.е. взаимосвязь, можно обеспечить только тогда, когда участники будут придерживаться общих правил, или общего протокола. Аппаратный интерфейс компьютера обеспечивается изготовителями оборудования, программный — авторами программ, аппаратно-программный и пользовательский интерфейс обеспечивает операционная система (ОС) — специально созданная совокупность программ, которая координирует работу компьютера, управляет размещением программ и данных в оперативной памяти, дает указания компьютеру, как распределять аппаратные ресурсы для выполнения задания и как управлять периферийными устройствами и обеспечивает взаимодействие человека и компьютера. Все это входит в функции операционной системы.

Пользователь в первую очередь имеет дело с интерфейсом операционной системы. Если ОС принимает команды, вводимые с клавиатуры в командной строке, то это неграфическая ОС и она имеет интерфейс командной строки. Современные ОС управляются мышью и для общения с пользователем применяют графические изображения. Они называются графическими и имеют графический интерфейс пользователя. В недалеком будущем возможно появление речевого интерфейса и ОС, управляемых голосом.

ОС должна решать две задачи:

1 — организация пользовательского интерфейса;

2 — организация совместной работы всех узлов компьютера и выполнение обязанностей диспетчера вычислительного процесса.

Для удобства работы с данными на дисках их размещают в файлах. Файл — это данные или программа, имеющие собственное имя и занимающие определенное место на диске.

Функциональная часть ОС, обеспечивающая выполнение операций над файлами, называется файловой системой.

Каждый файл, записанный на диске, имеет обозначение, состоящее из двух частей — имени и расширения. Они записываются рядом и разделяются точкой. Расширение файла описывает его содержание, состоит из трех символов и не является обязательным. Например:

.exe — программы, готовые к выполнению;

.txt или .doc — текстовый файл;

.bas — исходный текст программы на языке Бейсик.

Имена файлов регистрируются на дисках в каталогах, потому что их может быть очень много и бессистемное хранение может сильно затруднить работу пользователя. В каталог записываются имена файлов, сведения об их размерах и т.п.

На каждом магнитном диске имеется так называемый корневой каталог, который обозначается именем диска. В него занесены все записанные на этом диске каталоги и файлы. Любой каталог может содержать не только файлы, но и каталоги более низкого уровня, образуя целое «дерево» каталогов.

3.4. Операционная система Windows

Управление с помощью мыши

Система Windows управляется с помощью клавиатуры и мыши. Клавиатура используется в основном для ввода текстовой информации. Место ввода обозначается на экране текстовым курсором — мигающей вертикальной чертой. Многие операции можно быстро выполнить с помощью клавиатуры, однако мышь является основным средством управления. Указатель мыши имеет различный вид в зависимости от ситуации. Основной кнопкой мыши является левая кнопка.

Приемы управления с помощью левой кнопки мыши.

Щелчок — для выделения объекта (т.е. подготовки его к использованию), приведения в действие элементов управления (команд меню, кнопок на панелях и в диалоговых окнах и т.д.), установки текстового курсора в нужное место.

Двойной щелчок по объекту — для запуска приложений, открытия папок и документов.

Перетаскивание — для перемещения объектов. Надо установить указатель мыши на объект, нажать левую кнопку и, не отпуская ее, переместить указатель в нужное место, где отпустить кнопку.

Протягивание при нажатой **левой** кнопке — для изменения размеров объектов (например, окна), выделения фрагментов документа.

Наведение указателя на **объект** (или **зависание указателя** на объекте) — для вызова всплывающих подсказок.

Приемы управления с помощью правой кнопки мыши.

Правый щелчок — это щелчок правой кнопкой по объекту для вызова контекстного меню.

Специальное перетаскивание — для более надежного контроля над выполняемой операцией. Прием выполняется как перетаскивание, но правой кнопкой. При отпускании кнопки появляется небольшое меню из четырех пунктов: Переместить, Копировать, Создать ярлык, Отменить.

Основные элементы интерфейса

Запуск Windows не требует участие пользователя, загрузка Windows производится автоматически после включения компьютера.

1. **Рабочий стол.** Это основной объект операционной системы. На Рабочем столе размещаются объекты Windows и управляющие элементы. Рабочий стол Windows каждый пользователь может оформить по своему вкусу.

На Рабочем столе располагаются пиктограммы в виде значков, и ярлыков и панель задач.

2. **Пиктограмма.** Это условное обозначение информационного объекта Windows. Пиктограмма обычно включает некоторый рисунок или символ и название объекта.

Значок — небольшая картинка, представляющая конкретный объект (компьютер, диск, папку, файл, внешние устройства). Работая со значком, мы работаем с объектом, который он представляет. Значок «**Мой компьютер**» открывает доступ ко всем объектам компьютера. Значок «**Корзина**» — специальная папка, в которую временно помещаются удаляемые объекты.

Ярлык — разновидность значка. Он не представляет объект, а только на него указывает, ускоряя доступ к объекту. Внешне отличается от значка наличием стрелки. У объекта может быть много ярлыков.

Панель задач — элемент управления, располагается в нижней части экрана, содержит кнопку Пуск, которая открывает *Главное меню*. При открытии объектов на Панели появляются кнопки, соответствующие

окнам этих объектов. Справа на Панели находятся кнопки *Индикатора клавиатуры* и *Часы*. Индикатор клавиатуры показывает, какой регистр в данный момент установлен: EN — латинский, RU — русский. Переключение делается щелчком мыши по индикатору, затем щелчком по нужному варианту.

3. Окно. Так называется прямоугольная часть экрана, с которой можно работать как с отдельным экраном. Одновременно на экране может располагаться несколько окон, но в каждый момент времени допустимо работать лишь с одним. По окончании работы с программой или файлом окно нужно закрыть.

В состав Windows входит большое число различных по назначению программ. При запуске каждой такой программы она выводится в свое окно.

Многие программы создают в свою очередь окна, в которые можно выводить файлы, документы.

Просмотр системной информации. Для того чтобы узнать тип операционной системы, тип процессора и параметры памяти, нужно выделить на Рабочем столе папку «Мой компьютер» и открыть ее. В левой части окна выбрать закладку **Просмотр сведений о системе**.

Использование справочной системы. При решении возникающих вопросов целесообразно пользоваться специальной справочной системой, вызвать которую можно с помощью клавиши **Пуск/Справка**.

Выключение компьютера. При выключении компьютера очень важно помнить, что вся информация хранится в памяти и компьютер нужно выключать только после того, как ее сохранили на жестком диске. Выключение производится с помощью клавиши **Пуск/Выключение**. В появившемся окне нужно выбрать операцию **Выключение**.

Работа с меню

В нижней части Рабочего стола находятся элементы управления: кнопка Пуск и Панель задач. Панель задач — это строка внизу экрана.

Слева располагается кнопка Пуск, справа — Панель индикации. На этой панели можно вывести текущее время, индикатор раскладки клавиатуры (русский или английский), мелкие значки наиболее часто используемых объектов. Посередине располагаются кнопки открытых окон и приложений (задач). Эту часть панели называют панелью быстрого запуска. Так как система Windows является многозадачной, то может быть запущено несколько программ, что отражается на Панели задач.

Кнопка Пуск — это один из элементов управления. При щелчке по этой кнопке указателем мыши появляется Главное меню системы.

В системе Windows имеется несколько видов меню.

1. **Главное меню** — вызывается на экран щелчком по кнопке **Пуск**, содержит следующие пункты: Программы, Документы, Настройка, Поиск, Справка, Выполнить, Завершение работы. Стрелка справа от названия пункта означает, что для пункта имеется подменю.

2. **Меню окна приложения и окна папки** (называется также **горизонтальным меню, строкой меню, главным меню окна**) — находится под строкой заголовка окна.

Ниспадающее меню — выводится на экран при выборе пункта горизонтального меню. Пункты ниспадающего меню обычно называют командами.

Серый цвет некоторых команд означает, что в данный момент эти команды недоступны.

Многоточие после названия команды говорит о том, что появится диалоговое окно для запроса дополнительной информации. Некоторые команды являются **флажками**. После выбора такой команды перед ней появляется галочка. При повторном выборе команды галочка исчезает.

Группа команд, перед которыми при их выборе могут появляться кружочки, образуют **группу полей выбора**. В каждый момент может быть только один кружок, отмечающий выбранный вариант. Щелчок по другой команде перемещает кружок на эту команду.

Стрелка справа от названия команды означает, что команда имеет подчиненное меню. Строка меню вместе с ниспадающим меню позволяет воспользоваться всеми функциональными возможностями приложения.

3. **Панели инструментов** (пиктографическое меню) состоят из кнопок (пиктограмм), служат для быстрого вызова команд щелчком по кнопке. Панели обычно располагаются под горизонтальным меню, их можно выводить на экран и убирать. Для настройки панелей используются команды меню **Вид → Панели инструментов и Сервис → → Настройка**.

4. **Контекстное меню** — появляется в любой точке экрана по щелчку правой кнопки.

5. **Системное меню** — вызывается щелчком по кнопке в верхнем левом углу окна. Служит для управления размером окна, формой представления окна.

Настройка рабочего стола. Щелкнуть правой кнопкой по свободной части Рабочего стола. В контекстном меню рабочего стола выбрать **Свойства**, появится диалоговое окно **Свойства: Экран**.

Изменение фона. В качестве фона рабочего стола можно использовать либо «обои», либо узор.

1. Выбор «Обоев».
   - На вкладке **Фон** в списке **Рисунок рабочего стола** выбрать подходящий рисунок.
   - В списке **Поместить**» выбрать вариант **По центру**, **Рядом**, **Растянуть**.
   - Щелкнуть по кнопке **ОК**.
2. Выбор узора.
   - На вкладке **Фон** в списке **Рисунок рабочего стола** выбрать **Отсутствует**.
   - Щелкнуть по кнопке **Узор**, в списке **Узор** выбрать подходящий узор, щелкнуть по кнопке **ОК**.
   - На вкладке **Фон** щелкнуть по кнопке **ОК**.
3. Добавление экранной заставки.
   - На вкладке **Заставка** выбрать подходящий вариант, в окне просмотра посмотреть выбранную заставку.
   - В счетчике **Интервал** установить число минут бездействия компьютера, по истечении которых появится заставка
   - Выбрать, если надо, параметры заставки после щелчка по кнопке **Настройка**.
   - Щелкнуть по кнопке **ОК**.
4. Изменение цветовой схемы.
   - Выбрать вкладку **Оформление**.
   - В списке **Схема** выбрать цветовую схему.
   - Можно выбрать цвет и стиль отдельных элементов, используя список **Элемент**.
   - Щелкнуть по кнопке **ОК**.
5. Изменение способа отображения значков.
   - Выбрать вкладку **Эффекты**.
   - Для замены значка выделить его, затем щелкнуть по кнопке **Сменить значок**.
   - Выбрать значок, **ОК**.
   - Щелкнуть по кнопке **ОК**.
6. Изменение разрешения экрана.

Разрешение характеризуется числом используемых пикселей (или точек) по горизонтали и по вертикали. Чем выше разрешение, тем четче изображение, но меньше размер значков и текста.

- Выбрать вкладку **Настройка**.
- Установить разрешение с помощью ползунка **Область экрана**.
- В списке **Цветовая палитра** выбрать количество цветов.

7. Настройка параметров мыши.
   - Выполнить команду **Пуск → Настройка → Панель управления**.
   - Дважды щелкнуть по значку **Мышь**, появится диалоговое окно **Свойства: Мышь**.
   - На вкладке **Кнопки мыши** выбрать и проверить скорость двойного нажатия.
   - На вкладке **Указатели** выбрать набор указателей.

Выбор объекта. Запуск приложений и открытие документов

Объект системы можно выбрать несколькими способами: выбрав его ярлык на Рабочем столе либо воспользовавшись услугами Главного меню, или через папку, где он находится. Например, чтобы прочитать содержимое дискеты нужно:

Щелкнуть дважды по ярлыку Мой компьютер и выбрать соответствующий объект.

А для запуска, например, текстового редактора, можно воспользоваться его ярлыком на Рабочем столе или с помощью Главного меню.

Открыть документ в системе Windows можно, выбрав нужную папку, в которой этот документ находится, и щелкнув дважды мышью по значку документа, например в папке Мои документы находится папка ПРЕДМЕТ ИНФОРМ., в которой находится нужный нам файл ДОКЛАД.

Разновидности окон в системе Windows

Основным объектом в системе Windows являются окна. Можно выделить следующие виды окон:

1) окна папок содержат значки объектов Windows и элементы управления окном;
2) окна приложений содержат сам документ, обрабатываемый этим приложением, и элементы управления приложением;
3) диалоговые окна содержат только элементы управления, с помощью которых можно управлять как приложениями, так и самой операционной системой;
4) окна справочной системы содержат элементы управления справочной системой и информацию по работе с операционной системой и ее приложениями.

Управление размером и положением окон на экране

При запуске приложения оно читается с диска в оперативную память, открывается его окно, на панели задач внизу экрана появляется

кнопка с его именем. Если запущено несколько приложений, только одно из них активно. Заголовок его окна выделен темным цветом, а кнопка на панели задач нажата.

Переключиться на другое открытое приложение (сделать его активным) можно двумя способами:

- щелкнуть по любой части нужного, но неактивного окна;
- нажать кнопку нужного приложения на панели задач.

Окна приложений имеют три варианта представления: нормальное, свернутое и полноэкранное. Изменить представление окна можно с помощью кнопок  в правом верхнем углу окна.

Свернуть временно ненужное окно приложения можно, щелкнув по кнопке  . Свернутое окно считается открытым, так как приложение остается в оперативной памяти. Когда оно понадобится, восстановить окно, щелкнув по кнопке этого окна на панели задач.

Развернуть окно на весь экран можно, щелкнув по кнопке  , которая превратится в кнопку  . Щелкнув по ней, можно восстановить нормальные размеры окна.

При работе с несколькими окнами надо их удобно расположить на экране, для этого можно изменять размеры окон — подвести указатель мыши к его границе или углу. Когда указатель станет двунаправленной стрелкой, перетащить границу или угол окна при нажатой левой кнопке. Чтобы перетащить окно на другое место, надо установить указатель на заголовок окна и перетащить при нажатой левой кнопке.

Автоматическое расположение нескольких окон делается щелчком правой кнопки на свободном месте панели задач и выбором в контекстном меню одной из команд Окна каскадом, Окна сверху вниз, Окна слева направо.

Для просмотра содержимого окна, имеющего полосы прокрутки (вертикальную и горизонтальную), надо щелкать по кнопкам-стрелкам на концах полос или по полосе между стрелкой и бегунком, либо при нажатой левой кнопке мыши тащить бегунок вверх или вниз.

Работа с дисками, папками и файлами

Изменение вида представления папок и файлов. В строке меню окна щелкнуть пункт Вид и выбрать один из вариантов: эскиз страницы, плитка, значки, список или таблица.

Изменение порядка отображения папок и файлов в окне. Для изменения порядка отображения файлов выполните команду меню **Вид** → **Упорядочить значки** и выберите один из вариантов: по имени, по типу, по размеру, по дате.

Просмотр свойств файла. Получение информации об объекте (диске, папке или файле) одним из способов:

- выделить в окне объект щелчком, выбрать в меню команду **Файл → Свойства**;
- щелкнуть правой кнопкой по объекту и выбрать в контекстном меню команду **Свойства**.

В обоих случаях появится диалоговое окно, в котором надо просмотреть вкладки в поисках нужной информации.

Копирование, перемещение и удаление файлов.

К основным операциям с файлами (объектами) следует отнести:

1) выделение файла или группы файлов;
2) удаление файлов;
3) копирование файлов;
4) создание файлов.

1. Выделение объекта происходит при однократном щелчке мыши, при этом выделенный объект меняет цвет.

Выделить группы рядом стоящих объектов можно либо «протягивая» указатель мыши, либо двумя щелчками мыши при нажатой клавише Shift. Если объекты расположены не рядом, тогда их выделяют щелчком левой клавиши мыши с одновременно нажатой клавишей Ctrl. Снять выделение можно, просто щелкнув левой клавишей мыши. Еще проще выделить все содержимое окна выбрав в верхнем меню **Правка → Выделить все**.

2. Удаление объектов. При удалении объект попадает в папку Корзина. Это наиболее безопасное удаление — при необходимости из Корзины объект можно снова «достать». Если же объект не нужен — Корзину можно «очистить».

Удаление объектов можно выполнить несколькими способами:

1) «перетаскивая» объект в **Корзину**;
2) удаление выделенного объекта с помощью кнопки **Удалить**;
3) воспользоваться командой меню **Файл → Удалить**:
4) самый простой способ — выделить объект и нажать клавишу **Delet** на клавиатуре.

3. Копирование объектов.

1. Копирование с помощью перетаскивания. Для этого при перетаскивании нужно нажать клавишу **Ctrl**. Можно при перетаскивании использовать правую клавишу мыши, тогда появится меню, где нужно выбрать пункт **Копировать**.
2. Копирование с использованием кнопок панели инструментов.
3. Копирование командами меню производится с использованием команды меню **Правка → Копировать** и **Правка → Вставить**.

4. Можно быстро скопировать объект, используя клавиатуру: копирование **Ctrl + C**, вставка объекта в новое место **Ctrl + V**.

Буфер обмена. В Windows существует возможность обмена данными между различными приложениями, папками с помощью буфера обмена. *Буфер обмена* — это область памяти, в которую временно помещаются вырезанные или скопированные папки, файлы или фрагменты документа. Управляет работой обмена специальная программа Окно папки обмена.

При копировании и перемещении с помощью команд строчного меню или контекстного меню объект помещается в буфер обмена. Эти операции с помощью мыши выполняются без буфера обмена.

Помещенные в буфер папку, файл можно вставить в другую папку. Фрагмент документа из буфера можно вставить в другое место того же документа либо в другой документ. Содержимое буфера сохраняется там до тех пор, пока туда не поступят новые скопированные или вырезанные данные. В этом случае прежнее содержимое буфера обмена теряется безвозвратно, заменяясь новыми данными. При выходе из Windows содержимое буфера теряется.

4. Создание объектов. Для хранения своих документов пользователю иногда нужно создать свою папку. Ее можно создать и на Рабочем столе, и в уже имеющейся папке. Рассмотрим оба эти варианта.

На Рабочем столе нужно щелкнуть правой кнопкой мыши и выбрать нужный пункт в контекстном меню.

После этого вводится имя новой папки и она появится на Рабочем столе.

В открытой папке нужно щелкнуть правой клавишей мыши на свободном месте и выбрать пункт меню Создание новой папки.

При необходимости можно и переименовать выделенную папку, используя контекстное меню, которое вызывается при нажатии правой клавиши мыши.

Операции поиска

Для поиска данных можно воспользоваться командой **Пуск → Поиск → Папки и файлы**.

В верхнем поле указывается имя разыскиваемого файла, причем можно использовать так называемые подстановочные символы, такие как «?» — он заменяет любой один символ, и «*» — этот символ заменяет группу символов. Например, указав имя *.doc, получим все файлы с расширением doc. По имени 199? будут найдены все объекты, в названии которых присутствует любое число с 1990-го по 1999-й. Во вто-

ром окне можно задать ключевое слово. В окне **Где искать** — указывается адрес папки либо имя диска, на котором нужно произвести поиск. Процедура поиска начинается щелчком по клавише **Найти**. На вкладке **Дата** можно узнать, когда вносились изменения в разыскиваемый файл, а вкладка **Дополнительно** позволит найти файл и по фрагменту текста.

3.5. Операционная система Windows 7

Главными преимуществами Windows 7 являются улучшенная **доступность** и **легкость** использования. Она является вполне неплохим компаньоном для повседневной работы. Windows 7 оснащена **трехмерным пользовательским интерфейсом**. Microsoft называет новый интерфейс Aero.

При включении компьютера появляется панель центра начальной настройки, с помощью которой можно изменить настройки, просмотреть сведения о компьютере и многое др.

Также эта панель позволяет ознакомиться с возможностями Windows 7 с помощью специальных демонстрационных видеороликов.

Рабочий стол Windows 7

Рис. 3.1

Окно системы поиска

В Главном меню, которое появляется после нажатия на кнопку Пуск встроена поисковая строка для быстрого нахождения установленных программ.

Достаточно набрать в этой строке название программы (или его часть), и она будет найдена.

Изменился вид окна папки и значительно расширились возможности по организации файлов.

В нижней части окна отображается дополнительная информация о файле. Теперь каждому файлу можно дать краткий комментарий, указать автора, а также ключевые слова. Кроме этого, в верхней части окна находится список параметров, по которым можно производить сортировку файлов — при выборе одного из параметров будут показаны только те файлы, что ему соответствуют. Все это вместе взятое является очень полезным нововведением, позволяющим значительно упростить упорядочивание и поиск нужных данных.

Windiws 7 распознает, файлы какого вида хранятся в папке, и в зависимости от этого несколько меняет вид окна (в том числе и цвет). Так, если в папке находятся картинки, то добавляется кнопка **Показ слайдов**, а уменьшенное изображение выбранной картинки показывается в левой нижней части окна. Все картинки, размещенные в папке, по умолчанию тоже выглядят не в виде стандарного значка, характеризующего графический формат, а в виде миниатюрного изображения.

Центр специальных возможностей позволяет пользоваться компьютером и людям с ограниченными физическими возможностями. С помощью этого окна можно настроить работу компьютера без монитора, без мыши и клавиатуры, настроить изображение и т.п.

Восстановление системы. Windiws 7 поддерживает точки восстановления. Это временные образы системы, которые позволяют вернуться к предыдущему состоянию, если система оказалась повреждена вирусом или новый драйвер оказался сбойным или приложение повредило установку Windows. Функция отката Windiws 7 более мощная, она позволяет восстанавливать предыдущее состояние отдельных файлов или папок.

Боковая панель. В боковую панель (Sidebar) можно установить некоторые специальные мини-приложения. Это могут быть часы, слайдшоу с фотографиями или RSS-поток, адресная книга или панель быстрых заметок. На боковую панель можно вывести прогноз погоды, курс валют или что-то еще ценное.

Windiws 7 вобрала в себя немало улучшений. Она обеспечит лучший комфорт, связь, повышенную безопасность и платформу на будущее.

ГЛАВА 4

ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ ОБРАБОТКИ ГРАФИЧЕСКОЙ ИНФОРМАЦИИ

4.1. Виды компьютерной графики

В графическом виде информация становится более наглядной, лучше воспринимается человеком. Поэтому возникла идея поручить компьютерам осуществлять графическую обработку информации. Так появились графопостроители (или плоттеры), с помощью которых компьютер смог рисовать графики, чертежи, диаграммы. Однако это был только первый шаг в компьютерной графике. Следующим, принципиально новым шагом стало создание графических дисплеев. Работой графического дисплея управляет графический адаптер, состоящий из двух частей: видеопамяти и дисплейного процессора. Видеопамять (часть ОЗУ) служит для хранения видеоинформации — двоичного кода изображения. Дисплейный процессор управляет лучами электронно-лучевой трубки дисплея в соответствии с информацией, хранящейся в видеопамяти. Дисплейный процессор непрерывно «просматривает» (80—100 раз в секунду) содержимое видеопамяти и выводит его на экран.

Появление графических дисплеев существенно расширило возможности компьютерной графики. Для построения, коррекции, сохранения и получения «бумажных» копий рисунков и других изображений используется специальная программа — графический редактор.

Один из самых распространенных способов кодирования графики заключается в том, что изображение раскладывается на точки очень маленького размера (пиксели). В простейшем случае черно-белое изображение может быть представлено в виде набора из 2 битов. Нулем представляется точка белого цвета, а единицей — точка черного. Если изображение цветное, тогда каждая точка будет представлена в памяти компьютера не одним, а несколькими битами. Для полноцветных рисунков требуется большой объем памяти. Изображения, закодированные таким образом, называются битовыми картами, растрами или растровыми изображениями. Качество растрового изображения

определяется размером изображения (числом пикселей по горизонтали и вертикали) и количеством цветов, которые могут принимать пиксели. Растровые изображения очень чувствительны к масштабированию (увеличению или уменьшению). Когда растровое изображение уменьшается, несколько соседних точек превращаются в одну, поэтому теряется разборчивость мелких деталей изображения. При укрупнении изображения увеличивается размер каждой точки и появляется ступенчатый эффект, который виден невооруженным глазом. Этот эффект называют эффектом пиксилизации.

Растровый способ кодирования достаточно широко используется. Фотографии и рисунки, введенные в компьютер, хранятся в виде растровых изображений

Основными достоинствами растровой графики является легкость автоматизации ввода изобразительной информации и фотореалистичность. Недостатками — большие объемы файлов, невозможность увеличения размеров изображения без потери информации.

Растровое представление изображений существует не только в цифровом виде. Экран монитора или телевизора, отпечатанная иллюстрация также состоят из отдельных элементов — пикселей или точек.

В векторном представлении изображения строятся с помощью математических описаний объектов (так называемых примитивов), в качестве которых могут выступать линии, дуги, окружности, кривые Безье, текст и т.п. Векторную графику называют также «объектно-ориентированной», так как файл изображения формируется из дискретных, не связанных между собой элементов изображения, размеры, форма и цвет которых могут быть независимо друг от друга изменены быстро и без потери качества.

Практическими преимуществами векторного представления являются сравнительно небольшой объем файлов, независимость от разрешения устройства вывода и удобство редактирования.

Фрактальная графика, как и векторная, основана на математических вычислениях. Базовым элементом фрактальной графики является сама математическая формула, т.е. никаких объектов в памяти компьютера не хранится и изображение строится исключительно по уравнениям. Таким образом строят как простейшие регулярные структуры, так и сложные иллюстрации, имитирующие природные ландшафты и трехмерные объекты.

Фрактальная графика, как и векторная, — вычисляемая, но отличается от нее тем, что никакие объекты в памяти компьютера не хранятся. Изображение строится по уравнению (или по системе уравнений), поэтому ничего, кроме формулы, хранить не надо. Изменив коэффи-

циенты в уравнении, можно получить совершенно другую картину. Способность фрактальной графики моделировать образы живой природы вычислительным путем часто используют для автоматической генерации необычных иллюстраций.

4.2. Форматы графических файлов

Все форматы графических файлов можно условно разделить на растровые и векторные в соответствии с двумя типами графических изображений.

Растровые форматы

TIFF (Tagged Image Format). На сегодняшний день самый распространенный графический формат для растровых изображений. TIFF поддерживается практически всеми. Поддерживает все цветовые модели. Включает схемы сжатия для уменьшения размера файла. Применяется для хранения сканированных изображений и размещения их в программах графического дизайна и издательских системах.

PSD (PhotoShop Document). Внутренний формат программы Adobe Photoshop. Поддерживает 48-разрядное кодирование цвета и различные цветовые модели. Используется для хранения редактированных документов. Перед использованием в других приложениях, необходимо преобразовать в формат TIFF.

JPEG (Joint Photographic Experts Group). Возник как формат сжатия файлов. При каждом сохранении в JPEG происходит потеря качества. Поддерживает полутоновые и полноцветные изображения в моделях RGB и CMYK.

GIF (Graphics Interchange Format). Разработан специально для передачи растровых изображений по сетям. Поддерживает алгоритм. Файл GIF может содержать не одну, а несколько растровых изображений, которые демонстрируются поочередно с указанной в файле частотой (GIF-анимация).

Векторные форматы

Большинство векторных форматов могут также содержать не только векторную информацию, а также внедренные в файл растровые объекты или ссылку на растровый файл (технология OPI).

PDF (Portable Document Format). Формат PDF был создан фирмой Adobe в качестве формата электронного документооборота. PDF-

файлы сохраняют форматирование документа на разных компьютерных платформах и в различных прикладных программах. Из-за небольших объемов PDF-файлов наилучшим способом их пересылки стали средства электронной почты и Internet.

Adobe PostScript. Является языком описания страниц. Такие файлы содержат в себе сам документ, все связанные файлы (растровые и векторные), использованные шрифты, информацию о цветоделении и растрировании, управлении цветом и другие данные. Используются графическими программами и издательскими системами.

DXF. Формат DXF является форматом обмена чертежными данными в системах автоматизированного проектирования.

4.3. Представление цвета в компьютере. Цветовые модели

Каждый пиксель растрового изображения содержит информацию о цвете.

Представление информации в компьютере основывается на двоичной системе счисления. Минимальный размер цветовой информации в пикселе — 1 бит, т.е. в простейшем случае пиксели на экране могут быть «включены» или «выключены», представляя собой белый и черный цвет. Количество оттенков, которые может воспроизводить отдельный пиксель, определяется глубиной цвета (максимум — 32 бита), позволяющей показывать на экране монитора до 16,7 млн цветовых оттенков.

К **полноцветным** относятся типы изображений с глубиной цвета не менее 24 бит, т.е. каждый пиксель такого изображения кодируется как минимум 24 битами, что дает возможность отобразить не менее 16,7 млн оттенков. Поэтому иногда полноцветные типы изображения называют **True Color** (истинный цвет).

Если мы работаем с черно-белыми изображениями, то цвет кодируется нулем или единицей. Никаких проблем в этом случае не возникает. Для несложных рисунков, содержащих 256 цветов или столько же градаций серого цвета, нетрудно пронумеровать все используемые цвета. Но для изображений в истинном цвете, содержащих миллионы разных оттенков, простая нумерация не подходит. Для них разработаны несколько моделей представления цвета, помогающих однозначно определить любой оттенок.

Цветовые модели позволяют с помощью математического аппарата описать определенные цветовые области спектра.

Цветовая модель (режим) представляет собой правило обозначения цветов пикселей документа. Так как компьютер использует для обозначений цветов числа, необходимо ввести некоторое правило преобразования этих чисел в отображаемые устройствами вывода цвета и наоборот. Таких правил может быть несколько, поэтому каждое из них получает свое название.

Основные цветовые модели:

- RGB;
- CMY (Cyan Magenta Yellow);
- CMYK (Cyan Magenta Yellow Key, причем Key означает черный цвет);
- HSB;
- Lab;
- другие.

Разные режимы нужны для того, чтобы отобразить в файле особенности последующего вывода изображения на какое-либо устройство или сохранения в файле. Разные устройства вывода изображений могут работать по различным принципам, используя физические явления, не имеющие друг с другом практически ничего общего. Например, на экране монитора с электронно-лучевой трубкой (а также аналогичного телевизора) изображение строится при помощи засветки люминофора пучком электронов. При таком воздействии люминофор начинает излучать свет. В зависимости от состава люминофора этот свет имеет ту или иную окраску. Для формирования полноцветного изображения используется люминофор со свечением трех цветов — красным, зеленым и синим. Поэтому такой метод формирования цвета называют ***RGB*** **(Red, Green, Blue — Красный, Зеленый, Синий)**. Сами по себе зерна люминофора разных цветов позволяют получить только чистые цвета (чистый красный, чистый зеленый и чистый синий). Промежуточные оттенки получаются за счет того, что разноцветные зерна расположены близко друг к другу. При этом их изображения в глазу сливаются, а цвета образуют некоторый смешанный оттенок. Регулируя яркость зерен, можно регулировать получающийся смешанный тон. Например, при максимальной яркости всех трех типов зерен будут получен белый цвет, при отсутствии засветки — черный, а при промежуточных значениях — различные оттенки серого. Если же зерна одного цвета засветить не так, как остальные, то смешанный цвет не будет оттенком серого, а приобретет окраску. Такой способ формирования цвета напоминает освещение белого экрана в полной темноте разноцветными прожекторами. Свет от разных источников складывается, давая различные оттенки.

Поэтому такое представление цвета (цветовую модель) называют **аддитивной** (суммирующей).

При выводе изображения на печать используются другие технологии. Это может быть, например, струйная печать или многокрасочная печать на типографской машине. В этом случае изображение на бумаге создается при помощи чернил разных цветов. Накладываясь на бумагу и друг на друга, чернила поглощают часть света, проходящего сквозь них и отражающегося от бумаги. Если чернила густые, то они сами отражают свет, но не весь. Таким образом отраженный от картинки цвет приобретает ту или иную окраску в зависимости от того, какие красители и в каких количествах были использованы при печати. Обычно при таком способе цветопередачи для получения промежуточных оттенков используются чернила четырех цветов: голубой, пурпурный, желтый и черный. Такую цветовую модель называют ***CMYK*** — **Cyan, Magenta, Yellow, Black (Голубой, Пурпурный, Желтый, Черный)**. Теоретически для получения любого из оттенков достаточно только голубого, желтого и пурпурного цветов.

Однако на практике крайне сложно получить их смешением чистый черный цвет или оттенки серого. Так как в цветовой модели CMYK оттенки образуются путем вычитания определенных составляющих из белого, ее называют **субтрактивной** (вычитающей). Кроме различных печатающих устройств эта цветовая модель используется в фотопленке и фотобумаге. Там также содержатся слои, чувствительные к голубому, желтому и пурпурному свету.

В файлах изображений, сохраненных в режимах RGB и CMYK, для каждого пикселя записываются значения всех трех или четырех компонентов.

Таблица значений некоторых цветов в модели RGB

Цвет	R	G	B
Красный (red)	255	0	0
Зеленый (green)	0	255	0
Синий (blue)	0	0	255
Фуксин (magenta)	255	0	255
Голубой (cyan)	0	255	255
Желтый (yellow)	255	255	0
Белый (white)	255	255	255
Черный (black)	0	0	0

В модели CMYK количество каждого цвета можно задать в диапазоне от 0 до 100. Например:

- желтый (C-0, M-0, Y-100, K-0);
- голубой (C-100, M-0, Y-0, K-0);
- пурпурный (C-0, M-100, Y-0, K-0);
- красный (C-0, M-100, Y-100, K-0);
- зеленый (C-100, M-0, Y-100, K-0).

Модель **HSB** (Hue Saturation Brightness = Тон Насыщенность Яркость) построена на основе субъективного восприятия цвета человеком. Эта модель тоже основана на цветах модели RGB, но любой цвет в ней определяется своим цветом (тоном), насыщенностью (т.е. добавлением к нему белой краски) и яркостью (т.е. добавлением к нему черной краски). Фактически любой цвет получается из спектрального добавлением серой краски. Эта модель аппаратно-зависимая и не соответствует восприятию человеческого глаза, так как глаз воспринимает спектральные цвета как цвета с разной яркостью (синий кажется более темным, чем красный), а в модели HSB им всем приписывается яркость 100%. Модель является аппаратно-зависимой.

Цветовая модель **Lab** была специально разработана для получения предсказуемых цветов, т.е. она является аппаратно-независимой и соответствующей особенностям восприятия цвета глазом человека. Lab является трехканальной моделью. Цвет в ней определяется светлотой (яркостью) и двумя хроматическими компонентами: параметром a, изменяющимся в диапазоне от зеленого до красного и параметром b, изменяющимся в диапазоне от синего до желтого. Так как яркость в этой модели полностью отделена от цвета, это делает модель удобной для регулирования контраста, резкости и других тоновых характеристик. Цветовой охват Lab очень широк: он включает в себя RGB и CMYK, и другие цвета, непредставимые в двух предыдущих моделях. Цветовая модель Lab очень важна для полиграфии. Именно она используется при переводе изображения из одной цветовой модели в другую, между устройствами и даже между различными платформами. Кроме того, именно в этой модели удобнее всего проводить некоторые операции по улучшению качества изображения.

Для записи изображений в форматах, ограничивающих допустимое число цветов (таких как GIF), эти изображения надо предварительно перевести в ***режим индексированных цветов***. При этом составляется палитра, которая и используется при дальнейшей работе.

Палитра (palette) — набор цветов, используемых в изображении или при отображении видеоданных. Палитру можно воспринимать как таблицу кодов цветов (обычно в виде RGB-троек байтов в моде-

ли RGB). Палитра устанавливает взаимосвязь между кодом цвета и его компонентами в выбранной цветовой модели. Палитра может принадлежать изображению, части изображения, операционной системе или видеокарте. При попытке использовать не входящий в палитру цвет он заменяется ближайшим цветом, занесенным в нее.

4.4. Графические редакторы

Графический редактор — это программное обеспечение, которое используется для создания, редактирования, хранения и вывода графических изображений.

Основные функции графических редакторов.

- Вырезать, склеивать, стирать фрагменты изображений.
- Применять для рисования краски и кисти.
- Запоминать рисунки на внешних носителях, осуществлять поиск и воспроизведение.
- Увеличивать фрагмент изображения для проработки мелких деталей.
- Добавлять к рисункам текст.
- Масштабировать, перемещать и поворачивать изображение или его части.
- Добавлять различные эффекты (например, изменять цветовое решение, придать вид акварели, вид изображения на холсте, добавить источник света и многое др.).

В соответствии с двумя типами изображений графические редакторы можно разделить на растровые и векторные.

Растровые редакторы. В них изображение формируется из решетки пикселей. Для обработки изображений такие программы позволяют пользователю выбирать нужные электронные кисти, цвет и краску. Большинство цифровых изображений сначала поступают в компьютер при помощи сканера или цифрового фотоаппарата. С помощью сканера можно оцифровать слайд, диапозитив, фотографию путем преобразования изображения в цифровые данные, затем откорректировать цвет и отретушировать изображение, что наиболее часто используется в печатной компьютерной продукции, в первую очередь при создании рекламных объявлений и обложек журналов. Компьютер может поменять цвет прически или глаз, отретушировать изъяны, изменить цвет или фон фотографии, а также убрать все недостатки и дефекты. Можно добавить к фотографиям в журналах и рекламным объявлениям специальные эффекты, создавая сложные коллажи.

Растровые редакторы предназначены в основном для редактирования изображений, обеспечивая возможность цветокоррекции, ретуши и создания специальных эффектов на базе цифровых изображений. Они также широко используются производителями мультимедиа для создания текстовых и фоновых эффектов и изменения количества цветов изображения.

Среди растровых редакторов очень популярны такие редакторы, как **Adobe Photoshop** — один из самых мощных графических пакетов для любых применений, который позволяет работать с эффектами, слоями и множеством инструментов. Эта программа фактически является стандартом для ретуширования и работы с фотоизображением.

Программу **Corel Photo-Paint** выгодно отличает удобство в использовании, она имеет множество текстур, настраиваемый интерфейс, хорошие заготовки для веб-дизайнеров, поддержку анимации.

Векторные редакторы создают изображения, основываясь на математических формулах, а не на координатах пикселей. Составляющие основу таких изображений кривые и прямые линии называются векторами. Так как при задании объектов на экране используются математические формулы, то отдельные элементы изображения, создаваемые в векторных редакторах, можно легко перемещать, увеличивать или уменьшать без проявления «эффекта пикселизации». Так, для перемещения объекта достаточно перетащить его мышью. Компьютер автоматически пересчитывает его размер и новое местоположение.

Поскольку в этом случае изображение создается математически, векторные программы обычно используются тогда, когда нужны четкие линии. Они часто применяются при создании логотипов, шрифтов для вывода на плоттер и построения различных чертежей.

Так как качество изображения не основывается на разрешении, то изображение, созданное в векторных редакторах, как правило, имеет меньший объем файлов, чем построенное в растровых.

Универсальным векторным редактором, применяемым для решения абсолютно всех задач векторной графики является **Corel Draw**.

3D-графика и компьютерная анимация — еще одно широкое и посвоему сложное направлениекомпьютерной графики. 3D-графика — это создание искусственных предметов и персонажей, их анимация и совмещение с реальными предметами и интерьерами. На сегодняшний день определилось несколько перспективных направлений ее использования.

- Широкое применение 3D-графика находит в индустрии компьютерных игр. Анимационные заставки, интерфейсы и персонажи компьютерных игр создаются в программах 3D-графики.

- Другая область применения 3D-графики — телевизионная реклама и оформление телевизионных каналов.
- Многие архитекторы и дизайнеры пользуют 3D-графику для построения макетов зданий и трехмерных моделей архитектурных памятников, которых еще не существует в природе.

Освоение 3D-графики требует немало времени и мощных системных ресурсов.

Программа **Adobe Dimensions** позволяет придать трехмерное измерение графике и тексту. Здесь возможно создание высококачественных 3D моделей из любох объектов 2D графики, а также включение 3D в любой графический пакет Adobe. Выполняет поддержку эффектов освещенности, теней, деформации векторных и растровых объектов.

ГЛАВА 5

ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ ОБРАБОТКИ ТЕКСТОВОЙ ИНФОРМАЦИИ

5.1. Редактирование текстовых документов

Текстовые документы — это письма, статьи, справки, повести или романы и т.п.

Редактирование текста — это весь комплекс операций по внутренней (смысловой) и внешней (оформительской) работе над текстом. Из каждого текста можно вырезать куски, соединять их, вставлять в рабочий материал части из других текстов, менять их местами и пр. Можно менять положение текста на странице, формат строк и абзацев, вставлять в текст иллюстрации (рисунки, графики, схемы и пр.).

На современных компьютерах лучше использовать Microsoft Word. Простые вещи в нем делаются просто, плюс к тому существуют неограниченные возможности роста. В этом редакторе можно делать все — от простейших документов до газетных полос с иллюстрациями. Мы изучим только главные, основные возможности Word, как для краткости называют Microsoft Word.

5.2. Интерфейс текстового редактора Microsoft Word 2007

В верхней части окна текстового редактора Word 2007 находятся вкладки и лента меню. При включении вкладки **Главная** появляется лента главного меню. Его кнопки сгруппированы в панели по функциональным признакам: Буфер обмена, Шрифт, Абзац, Стили, Редактирование.

На панель вынесены наиболее часто используемые кнопки. Если нужной кнопки не оказывается на панели, то ее можно найти, нажав на небольшую стрелочку в правом нижнем углу определенной группы.

При этом появляется всплывающая подсказка, которая информирует о предназначении инструментов, что значительно упрощает знакомство с меню.

Рис. 5.1

В левом углу находится кнопка Office. Щелкнув по этой кнопке, мы получаем возможность выполнить следующие операции с документом: создать, открыть, сохранить, сохранить как..., печать и т.д.

Рядом с этой кнопкой находится панель быстрого доступа, по умолчанию с помощью этой панели можно сохранить файл, отменить или возвратить действие. Добавить или удалить кнопки на эту панель можно, щелкнув по правой клавише со стрелкой.

В нижней части окна находится строка состояния. В этой строке указываются количество страниц и номер текущей страницы, количество слов, язык ввода текста, режим просмотра документа, масштаб.

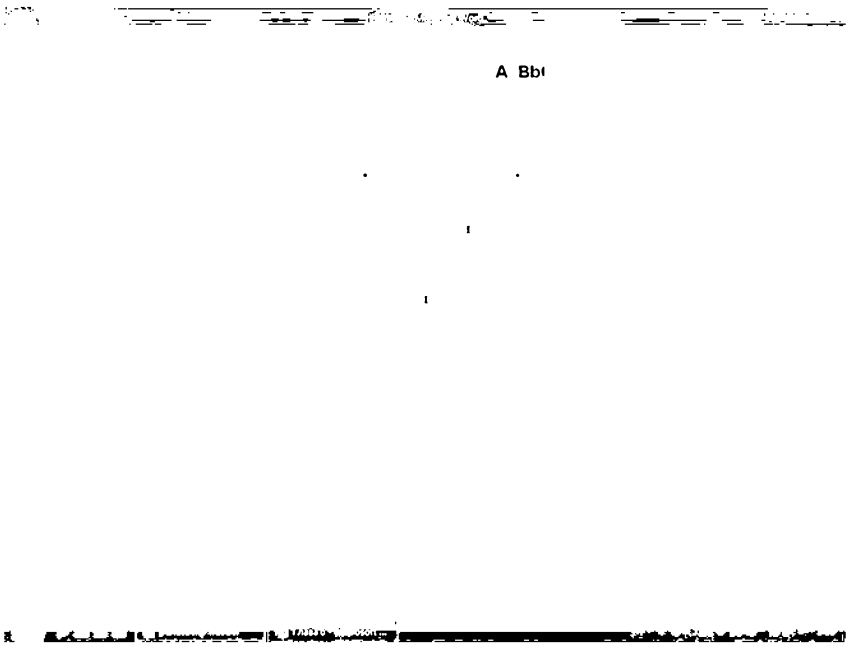


Рис. 5.2

5.3. Создание и сохранение документов в Microsoft Word 2007

Для создания нового документа предназначен пункт **Создать** меню Office. При его выборе появляется окно **Создание документа**. В левой части окна нужно выбрать категорию шаблонов, на основе которых будет создан документ. По умолчанию будет выбран вариант **Пустые и последние**. При нажатии кнопки **Создать** появится окно нового пустого документа.

Документ, созданный старой версией Word, будет открыт в режиме ограниченной функциональности — об этом будет сигнализировать строка заголовка:

Рис. 5.3

В этом режиме не все функции программы будут доступны. Для использования всех функций Word 2007 необходимо конвертировать файл при помощи меню **Преобразовать** кнопки Office.

Сохранение файлов по умолчанию в Word 2007 происходит в формате. docx, в котором не могут читаться старые версии программы. Для совмещения документа с предыдущими версиями Word необходимо сохранять файл в «режиме ограниченной функциональности». Это делается с помощью меню **Сохранить как...** кнопки Office.

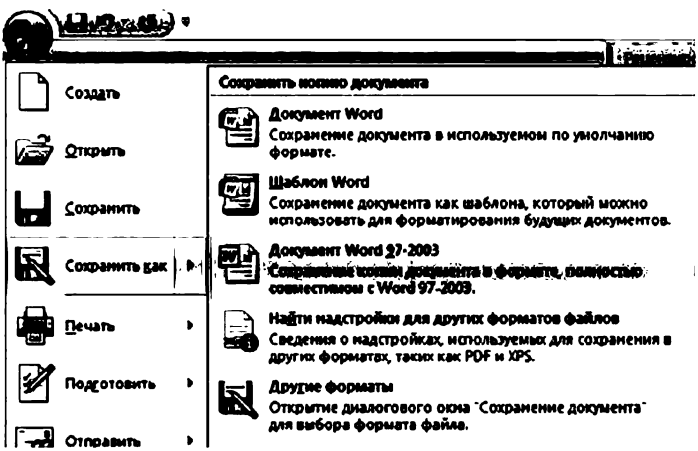


Рис. 5.4

5.4. Форматирование документов

Чтобы произвести действия с набранным текстом, его надо выделить либо протяжкой мыши (при этом должна быть нажата левая кнопка мыши), либо при помощи клавиш управления курсором при нажатой кнопке Shift.

В Word 2007 существует специальный режим выделения текста, для переключения в этот режим нужно нажать клавишу F8. После этого текст можно выделять клавишами управления курсора (или щелчком мыши в нужном месте), при этом использовать кнопку Shift не нужно. Для выхода из этого режима необходимо нажать клавишу Escape. Несколько нажатий F8 последовательно выделяют слово, предложение, абзац, весь текст.

Основные инструменты форматирования размещены на ленте вкладки **Главная** — это панели Буфер обмена, Шрифт, Абзац, Стили и Редактирование.

На панели **Буфер обмена** расположены четыре основные кнопки: Вставить, Вырезать, Копировать, Формат по образцу. Кнопка **Вставить** активна, если в буфере обмена есть объект. Кнопки **Вырезать** и **Копировать** активны, если выделен фрагмент текста, рисунок, таблица и т.д.

Рис. 5.5

Кнопка **Формат по образцу** переносит параметры форматирования указанного объекта на выделяемый фрагмент. Чтобы перенести все параметры форматирования на новый абзац, необходимо:

- установить курсор в любом месте абзаца, параметры форматирования которого будут использоваться;
- нажать кнопку **Формат по образцу**;
- выделить текст, на который надо перенести форматирование.

С помощью панели инструментов **Шрифт** можно изменять размер, тип и начертание шрифта, увеличить (уменьшить) размер шрифта; применить эффект надстрочного (подстрочного) начертания; изменить регистр текста; его цвет; цвет выделенного фрагмента. При применении эффекта подчеркивания можно сразу указать вид линии. Кнопка **Очистить формат** удаляет измененные параметры форматирования.

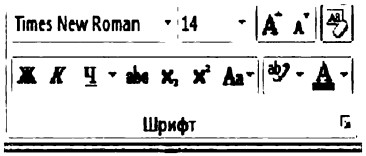


Рис. 5.6

Дополнительные параметры форматирования можно настроить при помощи диалогового окна **Шрифт**.

Word 2007 предоставляет возможность быстрого форматирования текста. При выделении фрагмента текста рядом с ним появляется прозрачное окно форматирования. При наведении курсора на это окно оно приобретает нормальный цвет. Окно содержит наиболее часто встречающиеся команды форматирования.

Панель **Абзац** предназначена для абзацного форматирования. На этой же панели находятся и кнопки работы с таблицами.

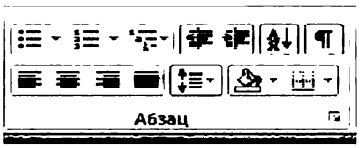


Рис. 5.7

Верхние левые три кнопки предназначены для работы с маркированными, нумерованными и многоуровневыми списками. Следующие кнопки увеличения (уменьшения) абзацного отступа, кнопка сортировки табличных значений по алфавиту и кнопка включения (выключения) непечатаемых символов. В нижнем ряду находятся кнопки выравнивания текста в абзаце (по левому краю, центру, правому краю, ширине), кнопка установки междустрочного интервала, заливка ячеек и установка видимых границ.

Более тонкие настройки форматирования абзаца можно производить с помощью диалогового окна **Абзац**.

Панель **Стили** предназначена для автоматизация работы с документами. ***Применение стилей.*** В любой книге существуют несколько уровней заголовков — главы, пункты, подпункты и т.д. Для единообразного оформления книги существуют так называемые стили. Они позволяют очень быстро, а главное стандартно, менять оформление фрагментов текста. В Word имеется некоторое количество готовых стилей.

Рис. 5.8

Последняя панель **Редактирование** предназначена для быстрого поиска (замены) нужного фрагмента текста.

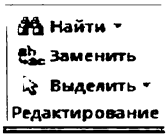


Рис. 5.9

5.5. Лента Вставка Microsoft Word 2007

Лента Вставка дает возможность пользователю быстро осуществить вставку страниц, объектов, таблиц, надписей и т.п., а также оформить страницы колонтитулами, номерами. Первая панель **Страницы** позволяет вставить титульный лист, пустую страницу, разрыв страницы.

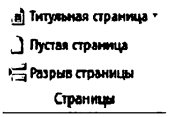


Рис. 5.10

Для вставки таблицы служит кнопка **Таблица**, расположенная на одноименной панели. При нажатии на эту кнопку можно в интерактивном режиме выбрать необходимое количество строк и столбцов для будущей таблицы.

Если таблица очень большая и количество предлагаемых ячеек недостаточно, нужно воспользоваться опцией **Вставить таблицу** и в появившемся окне задать необходимое количество строк и столбцов.

При создании сложной таблицы нужно воспользоваться кнопкой **Нарисовать таблицу**.

Кроме вышеупомянутых вариантов создания таблицы есть несколько вариантов готовых экспресс-таблиц.

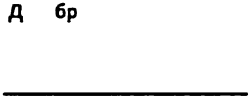


Рис. 5.11

Как и в предыдущих версиях, Word 2007 позволяет вставлять документ таблицы Excel. Для этого служит кнопка **Таблица Excel**.

Уже набранный текст можно преобразовать в таблицу.

Для этого необходимо выделить нужный блок текста и выбрать пункт меню **Преобразовать в таблицу**. В появившемся окне надо задать параметры будущей таблицы.

После того как таблица вставлена, в окне текстового редактора появляется контекстный инструмент **Работа с таблицами**, содержащий две ленты: **Конструктор** и **Макет**.

Для работы с графикой служит панель **Иллюстрации**.

Рис. 5.12

Для вставки рисунка необходимо воспользоваться кнопкой **Рисунок**. Кнопка **Фигуры** служит для быстрого создания графических примитивов.

Графика SmartArt позволяет быстро создавать разнообразные красочные схемы.

Для вставки готового рисунка-клипа необходимо нажать кнопку **Клип**. Коллекция Clip Art содержит подборку набора картинок текстового редактора.

Построение диаграммы в Word 2007 осуществляется с помощью кнопки **Диаграмма**. В появившемся окне надо выбрать тип диаграммы и ее вид.

Закладки предназначены для быстроты и удобства навигации по документу — они позволяют быстро переходить к ранее помеченным местам в тексте. Для того чтобы сделать закладку, необходимо установить курсор в нужном месте документа и нажать кнопку **Закладка.**

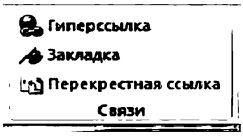


Рис. 5.13

В появившемся окне необходимо ввести имя закладки. Следует иметь в виду, что имя должно начинаться с буквы и не содержать пробелов. При помощи этой же кнопки и окна можно перемещаться по закладкам, добавлять новые и удалять ненужные.

Панель Колонтитулы, как и следует из названия, предназначена для работы с колонтитулами.

После вставки колонтитул становится доступен для редактирования, при этом появляется контекстная лента работы с колонтитулами **Конструктор**. Она позволяет быстро произвести такие настройки колонтитула, как:

- различные колонтитулы для четных и нечетных страниц;
- отдельный колонтитул для первой страницы;
- скрытие основного текста во время работы с колонтитулами;
- вставка и редактирование номера страницы;
- управление положением колонтитула;
- вставка в колонтитул различных объектов: текущие дата и время, рисунки, стандартные блоки, объекты ClipArt.

Колонтитулы можно настраивать отдельно для различных разделов, для чего нужно разорвать между ними связь, так как по умолча-

нию все колонтитулы связаны между собой. Для этого надо перейти к тому колонтитулу, который надо оформить по-другому, и «отжать» кнопку **Как в предыдущем разделе**.

Быстрый переход между колонтитулами и основным текстом документа можно осуществлять двойным щелчком мыши на нужном элементе (верхнем (нижнем) колонтитуле или на основном тексте).

Отредактированный колонтитул можно добавить в галерею колонтитулов при помощи опции **Сохранить выделенный фрагмент в коллекцию верхних/нижних колонтитулов**.

Для удаления колонтитулов предназначен пункт **Удалить верхний/нижний колонтитул** соответствующих кнопок колонтитулов.

Для нумерации страниц служит кнопка **Номер страницы**. При нажатии этой кнопки необходимо выбрать вариант размещения номера на самой странице и при необходимости настроить формат самого номера.

Панель **Текст** позволяет вставлять надписи, блоки и объекты WordArt — красиво оформленный текст на основе готовых шаблонов.

5.5. Оформление страниц документов

Если стандартные установки не подходят для создаваемого документа, нужно установить параметры страницы, для чего служит лента **Разметка страницы**, состоящая из панелей: *Темы; Параметры страницы; Фон страницы; Абзац; Упорядочить.*

В Word 2007 появилась новая функция — **Темы оформления**, которые можно применять к текстовым документам. На вкладке **Темы**, нажав кнопку «Темы», можно попасть в галерею, содержащую несколько вариантов оформления документа.

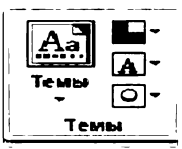


Рис. 5.14

Темы можно удалять и редактировать с помощью кнопок: *Цвета темы; Шрифты темы; Эффекты темы.* При изменении параметров шрифтов будут модифицированы используемые в документы стили. Чтобы сохранить новую тему в виде отдельного файла, нужно нажать кнопку **Темы** и выбрать пункт **Сохранить текущую тему**. Тема добавится в галерею, в которой появится область «Пользовательские».

Панель Параметры страницы

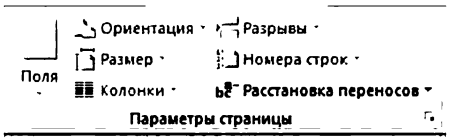


Рис. 5.15

Кнопка Поля служит для установки значений полей документа. Если из предложенных стандартных вариантов ни один не подходит, необходимо воспользоваться пунктом меню **Настраиваемые поля...** . В появившемся окне можно произвести более тонкие настройки полей документа.

Кнопка Ориентация задает расположение текста на листе: Книжная, Альбомная.

Кнопка Размер задает размер бумаги при выводе на печать. Для выбора нестандартного размера служит опция «Другие размеры страниц...».

Следующая **кнопка Колонки** служит для разбивки текста страницы на несколько колонок (подобно газетной верстке). Опция «Другие колонки...» служит для гибкой настройки колонок. Все функции настройки интуитивно понятны, к тому же в окне «Образец» сразу показано, как будет выглядеть страница.

Чтобы начать новую страницу есть специальная опция — **Разрывы**. Word 2007 предоставляет четыре варианта разрыва разделов: *Следующая страница; Текущая; Четная страница; Нечетная страница.* Чтобы видеть разрывы разделов (как, впрочем, и страниц), нужно включить опцию отображения непечатных символов. Для этого на ленте «Главная» на панели «Абзац» необходимо нажать правую верхнюю кнопку с изображением значка абзаца. Для удаления раздела необходимо выделить его значок и нажать кнопку **Delete**.

Опция Номера строк предназначена для нумерации строк документа в различных вариациях. Из практики можно сказать, что к подобной нумерации прибегают довольно редко. Но в отдельных случаях она может быть весьма полезной.

Word 2007 по умолчанию работает в режиме автоматического размещения текста. Программа может и расставлять и переносы слов. Для этой цели служит опция **Расстановка переносов**. Возможны два варианта: Автоматическая настройка; Ручная настройка. Пункт **Параметры расстановки переносов** позволяет сделать тонкую настройку параметров расстановки переносов.

Фон страницы

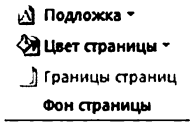


Рис. 5.16

Кнопка Подложка дает возможность добавлять подложку на страницы. В качестве подложки можно использовать текст или рисунок. Если не подошла ни одна из предложенных подложек, можно создать свою. Для этого предназначен пункт **Настраиваемая подложка...** .

Для создания текстовой подложки надо установить переключатель в положение «Текст», ввести нужный текст, настроить необходимые параметры: язык, шрифт, цвет и расположение надписи, прозрачность.

Для создания графической подложки надо установить переключатель в положение «Рисунок» и нажать кнопку **Выбрать**. Затем указать место размещения нужного файла изображения.

При желании можно отредактировать представленные в галерее стандартные подложки. Для этого надо щелкнуть на выбранном варианте правой кнопкой мыши и выбрать команду **Изменить свойства**. Удалить подложку из галереи можно с помощью пункта «Удалить подложку».

Кнопка Цвет страницы позволяет установить практически любой цвет для страницы. Следует учитывать, что не каждый цвет может воспроизвести принтер во время печати документа. Поэтому лучше использовать стандартную палитру цветов. Здесь можно выбрать и способ заливки фона страницы (градиентная, узором, текстурная). Или же выбрать какое-либо изображение для фона страницы.

Кнопка Границы страниц устанавливает видимые печатные границы страницы.

Абзац.

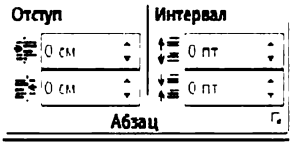


Рис. 5.17

На этой панели расположены две опции форматирования абзаца: **Отступ** и **Интервал**. Они регулируют свободное поле по горизонтали и вертикали соответственно.

Упорядочить

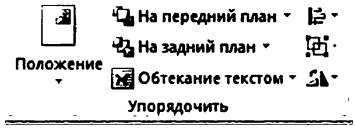


Рис. 5.18

С помощью кнопок на этой панели можно задать положение и обтекание объекта — рисунка, таблицы, диаграммы.

5.6. Редактирование и рецензирование документов

Средства рецензирования и редактирования текстового редактора помогут, когда с одним и тем же документом приходится одновременно работать нескольким пользователям. Эти средства собраны на ленте **Рецензирование**.

Левая панель ленты предназначена для проверки правописания, на ней также расположены кнопки **Справочники**, **Тезаурус** и **Перевод**. Очень удобны услуги автоматического переводчика и автоматический подсчет знаков и строк.

Рис. 5.19

Для добавления (и последующего управления) примечаний в документ предназначена панель **Примечания**. Чтобы создать примечание, надо установить курсор в нужное место документа и нажать кнопку **Создать примечание**. При этом фрагмент текста выделяется красным цветом, а на полях появляется поле для ввода примечания, а на панели **Примечания** становятся доступными кнопки навигации и удаления примечаний.

Рис. 5.20

На панели **Отслеживание** находятся инструменты, позволяющие отслеживать изменения, вносимые в документ. Для этого надо установить кнопку **Исправления** в «нажатое» состояние.

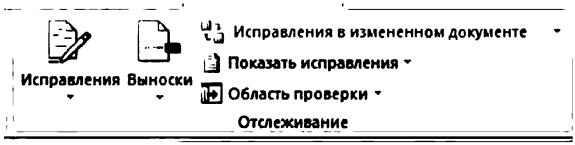


Рис. 5.21

Кнопка Выноски позволяет выполнять следующие действия:

- *«Показывать исправления в выносках»* — примечания и исправления будут отображаться в виде выносок;
- *«Показывать все исправления в тексте»* — все исправления и примечания будут отображаться непосредственно в тексте;
- *«Показывать только примечания и форматирование в выносках»* — в выносках будут отображаться только примечания и форматирование документа.

Кнопка Область проверки открывает дополнительную панель, на которой отображаются в хронологическом порядке внесение исправлений и добавления примечаний.

С помощью верхнего выпадающего списка можно настроить отображение изменений в документе:

- исходный документ;
- исправления в исходном документе;
- измененный документ;
- исправления в измененном документе.

Для выхода из режима отслеживания изменений надо «отжать» кнопку **Исправления**.

Если необходимо скрыть исправления, сделанные в документе, надо снять соответствующие флажки в выпадающем списке *«Показать исправления»*.

На **панели Изменения** собраны кнопки, позволяющие перемещаться между внесенными в документ правками, а также принимать или отклонять сделанные изменения.

Рис. 5.22

Сравнение документов, в которые вносились изменения разными пользователями, производится с помощью панели **Сравнить**.

Для защиты документа от изменений служит панель **Защитить**. После нажатия на кнопку *«Защитить документ»* у правого края окна появляется вертикальная панель *«Ограничить форматирование»*.

Установите флажок *«Ограничить набор разрешенных стилей»* и в опциях *«Настройки...»* укажите, какие элементы оформления можно будет форматировать при дальнейшей работе с документом.

Для ограничения редактирования необходимо установить флажок *«Разрешить только указанный способ редактирования документа»* и из выпадающего списка выбрать пункт *«Запись исправлений»*. Этим самым мы разрешаем добавлять комментарии к документу, удалять, вставлять и перемещать текст. Если же мы хотим другим пользователям разрешить только оставлять примечания, то надо выбрать пункт *«Примечания»*.

Для включения защиты нажмите кнопку *«Да, включить защиту»*.

Чтобы снять защиту, необходимо нажать кнопку *«Защитить документ»* и в появившемся списке снять флажок *«Ограничить форматирование и редактирование»*.

5.7. Лента Вид и печать документа

Панель **Режим просмотра документа** крайняя слева на ленте **Вид**.

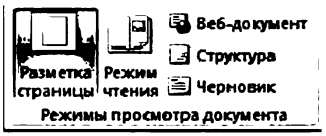


Рис. 5.23

По умолчанию документ в Word отображается в таком виде, в каком он будет напечатан, т.е. в режиме **Разметка страницы**.

Режим чтения — просмотр документа в полноэкранном режиме чтения. При этом из окна исчезают почти все элементы интерфейса, на экране находится один текст и несколько кнопок управления.

Для перехода в режим структуры документа служит кнопка **Структура**, при этом появляется контекстная лента **Структура**.

В режиме просмотра структуры документа отображается иерархия элементов оформления текста. Прежде чем использовать этот режим, нужно отформатировать документ с применением стандартных заголовков.

Черновик — режим для быстрого редактирования документа. В этом режиме не отображаются некоторые элементы документа.

На панели **Показать или скрыть** устанавливаются или снимаются дополнительные элементы окна.

- *Линейка* — служит для быстрой настройки полей, отступов, табуляции (очень рекомендую включить).
- *Сетка* — помогает форматировать документ, содержащий таблицы и рисунки.
- *Схема документа* — используется при работе с большими документами.
- *Эскизы* — просмотр общего вида страниц всего документа.

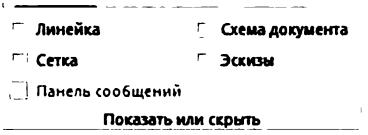


Рис. 5.24

Панель **Масштаб** ленты **Вид** содержит пять кнопок.

- **Кнопка Масштаб** содержит все инструменты быстрой и точной настройки масштаба.
- **Кнопка 100%** — отображает документ 1:1.
- **Кнопка Одна страница** — на экране будет отображена страница целиком.
- **Кнопка Две страницы** — на экране будет отображено две полных страницы документа.
- **Кнопка По ширине страницы** — изменение масштаба документа таким образом, что ширина страницы документа будет равна ширине окна.

Рис. 5.25

При одновременной работе пользователя с несколькими документами кнопки панели **Окно** значительно упрощают его работу.

- *Новое окно* — создает новое окно для просматриваемого документа.
- *Упорядочить все* — размещает рядом окна всех документов, открытых на данный момент.

- *Разделить* — разделяет окно документа на две части. При этом в каждой из частей можно листать документ независимо от другой. Очень удобно, когда приходится часто работать в разных частях большого документа.
- *Рядом* — располагает окна открытых документов рядом для сравнения их содержимого.
- *Синхронная прокрутка* — становится активной при нажатой кнопке «Рядом» и позволяет синхронно прокручивать документы.
- *Восстановить расположение окна* — изменение положения окон сравниваемых рядом документов таким образом, чтобы каждое из них занимало половину экрана.
- *Перейти в другое окно* — переключение между окнами открытых документов.

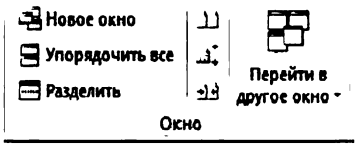


Рис. 5.26

Печать документов

После того как документ набран и отформатирован, в 99% случаев его нужно вывести на печать. Для этого служит пункт «Печать», находящийся в меню кнопки «Office» (сочетание клавиш Ctrl + P).

5.8. Сложное форматирование документов

Операции сложного форматирования осуществляются при помощи ленты **Ссылки**.

Оглавление

Оглавление — это список заголовков документа.

Рис. 5.27

Для того чтобы быстро сделать оглавление, документ должен быть отформатирован согласно встроенным форматам уровней структуры или стилей заголовков.

Затем, установив курсор в месте вставки оглавления, нажмите кнопку *Оглавление одноименной* панели. В появившемся окне можно выбрать нужный формат оглавления.

Сноски

Сноски предназначены для добавления к тексту комментариев, объяснений, указания источника информации.

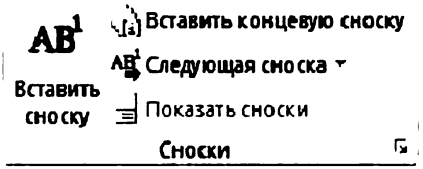


Рис. 5.28

Сноски бывают обычные (в конце страницы) и концевые (в конце всего текста).

Сноски нумеруются автоматически в соответствии с выбранной системой нумерации. При добавлении новой сноски или удалении существующей остальные перенумеровываются.

Для работы с библиографией и цитатами служит панель **Ссылки и списки литературы**

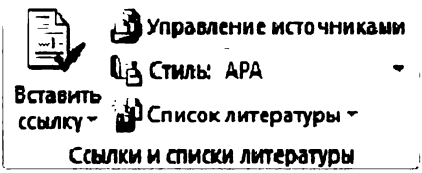


Рис. 5.29

Данные элементы форматирования присутствуют, как правило, в научных работах.

Для быстрого перехода к нужному элементу документа служат перекрестные ссылки. Перекрестные ссылки можно создавать на следующие элементы: заголовки, сноски, закладки, названия, нумерованные абзацы. Инструменты для работы с перекрестными ссылками находятся на панели **Названия**.

Рис. 5.30

Перекрестные ссылки создаются только между элементами одного документа.

Предметный указатель — это список терминов, встречающихся в документе, с указанием страниц где они расположены.

Рис. 5.31

Предметный указатель можно создать для следующих элементов:

- отдельные слова, фразы, символы;
- разделы;
- ссылки.

ГЛАВА 6

ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ ОБРАБОТКИ ЧИСЛОВОЙ ИНФОРМАЦИИ

6.1. Электронные таблицы

Одной из самых продуктивных идей в области компьютерных информационных технологий стала идея электронной таблицы. **Электронная таблица** — это просто прямоугольная таблица, состоящая из строк и столбцов. Например:

Дата	№ школы	Кол-во учащихся	Кол-во отличников	Кол-во хорошистов	Кол-во троечников	Кол-во неуспевающих
01.09	87	1 000	300	400	200	100
01.09	99	1 200	500	350	200	150
01.09	115	1 300	500	350	300	150
01.09	628	450	300	100	50	0

Прикладные программы для работы с электронными таблицами часто называют табличными процессорами. Табличные процессоры — удобный инструмент для тех, кому приходится работать с большими массивами числовой информации: экономистов, бухгалтеров, инженеров, научных работников. Эти программы позволяют создавать динамические таблицы, содержащие вычисляемые поля, значения которых автоматически пересчитываются по заданным формулам при изменении значений исходных данных, содержащихся в других полях.

Microsoft Excel — это электронная таблица, позволяющая производить вычисления, анализировать их и представлять в графическом виде. Excel — составная часть программ для работы именно в офисе, она является верным помощником для огромного количества людей, благодаря своему мощному арсеналу математических, финансовых, статистических и логических функций.

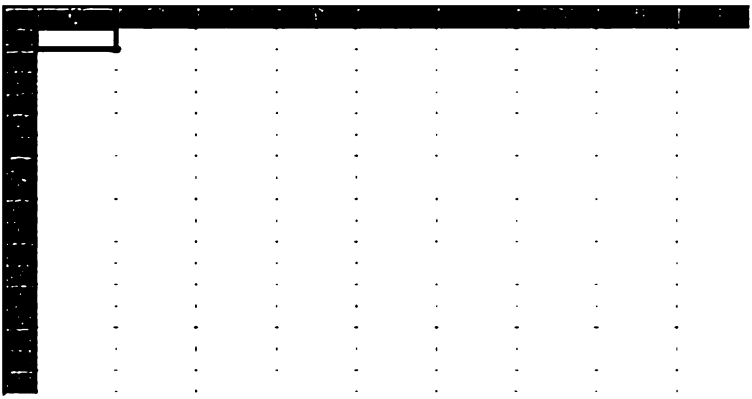


Рис. 6.1

Электронная таблица Excel разделена на клетки, подобно шахматной доске, эти клетки принято называть ячейками таблицы. Строки и столбцы таблицы имеют обозначения: строки — числовую нумерацию, а столбцы — буквы латинского алфавита. Каждая ячейка имеет свой адрес, который состоит из имени столбца и номера строки, например: A1, C2, B15.

В каждую ячейку можно занести текст, число или формулу.

Документом Excel является файл с произвольным именем и расширением XLS (в Excel 2007 — .XLSX). Такой файл называется рабочей книгой. В ней размещаются электронные таблицы, каждая из которых называется рабочим листом.

6.2. Работа с электронными таблицами в программе Microsoft Excel 2007

В Excel 2007 возросло количество ячеек на рабочем листе. Максимальное число строк превышает миллион, столбцов стало больше 16 тысяч, это позволяет импортировать и обрабатывать огромные объемы данных в одной таблице. Обозначение столбцов заканчивается на символьной комбинации XFD.

При запуске программы Excel 2007 появится окно программы, которое выглядит так:

Вверху находятся семь лент с инструментами: *Главная*, *Вставка*, *Разметка страницы*, *Формулы*, *Данные*, *Рецензирование*, *Вид*. Некоторые из них очень похожи на ленты из Word 2006.

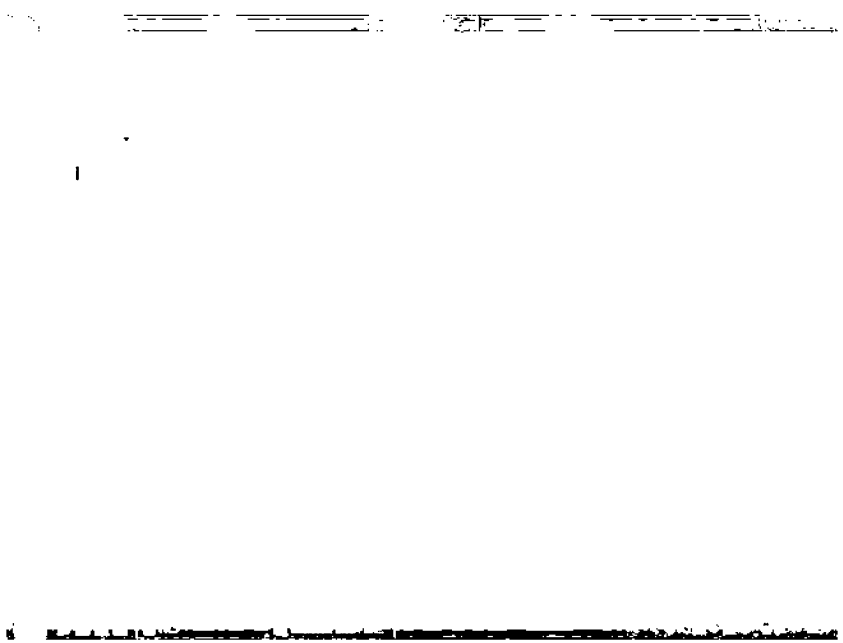


Рис. 6.2

Каждая лента состоит из панелей, на которых расположены инструменты для работы с электронными таблицами. На панель вынесены наиболее часто используемые инструменты. Для вызова полного набора инструментов той или иной панели надо открыть окно данной панели, нажав на стрелочку в правом нижнем углу.

В левом верхнем углу окна программы находится кнопка Office.

По умолчанию программа сохраняет файл с расширением xlsx, который не читается предыдущими версиями Excel. Для того чтобы документ был совместим с ранними версиями электронных таблиц, необходимо во время сохранения файла выбрать соответствующую опцию.

Если задача требует не одной таблицы, а нескольких, то информацию, связанную тематически, удобнее хранить в одном файле, но на разных листах одной рабочей книги.

Здесь представлен пример рабочей книги, раскрытой на листе «Смета». С листами можно выполнять многие операции, их можно перелистывать, вырывать, вставлять, давать им имена, а информацию, хранящуюся на одном листе, можно использовать на другом листе.

Если закладка листа не поместилась на панели, ее можно передвинуть с помощью кнопок, расположенных в левом нижнем углу экрана:

Рис. 6.1

Крайние кнопки устанавливают указатель на первый или последний лист. Средние — передвигают указатель вправо-влево.

Для того чтобы вставлять, переставлять, удалять, копировать и переименовывать листы, необходимо для начала щелкнуть по нужной закладке ПРАВОЙ клавишей мыши и в появившемся меню выбрать нужный пункт.

Для перемещения по таблице используется маркер, маркер — прямоугольник, который указывает на нужную клетку. Обведенная рамкой ячейка называется активной. Вводить данные можно только в активную ячейку. Передвигать маркер можно клавишами-стрелками, но лучше работать с мышью. Если нужная клетка не видна на экране, то для перемещения по таблице используют ползунки внизу и справа окна.

Для того чтобы ввести информацию в ячейку, надо подвести маркер к нужной клетке, набрать содержимое и нажать клавишу Enter.

У Excel существует несколько форм указателя мыши, которые меняются при перемещении по листу:

- при выборе и выделении ячеек используется указатель, который имеет вид большого белого креста;
- при заполнении ячеек указатель мыши принимает вид черного крестика. Он появляется тогда, когда указатель мыши расположен ниже правого нижнего угла активной ячейки у черного квадрата, который называется маркером заполнения;
- при установке указателя мыши на границу активной ячейки он принимает вид стрелки, с помощью которой можно перемещать активную ячейку.

При вводе повторяющихся данных можно произвести копирование ячеек, протянув маркер заполнения вниз, выделяя ячейки, в которые будет скопирована информация.

Использование автозаполнения для копирования и увеличения скорости ввода данных. Программа Excel обладает средством автозаполнения ячеек: содержимое ячейки запоминается, этим можно воспользоваться при заполнении другой ячейки в этом же столбцы. Если в ячейке A1 набрать текст: Количество учащихся, затем в ячейке A2 напечать букву К, программа может продолжить ввод автоматически, при про-

должении ввода другого текста автоматически продолженный вариант будет заменен.

Если в столбце остается свободная ячейка, то автозаполнение сбрасывается. Это свойство очень полезно, когда столбцы данных должны содержать повторяющиеся названия фирм, городов, стран или другую подобную информацию.

При вводе данных возможны ошибки, которых можно избежать, если четко знать, каким условиям должны удовлетворять вводимые данные. Для этого нужно выделить диапазон ячеек, в которые будут вводиться данные, затем выбрать инструмент *Проверка данных* на панели **Работа с данными** ленты **Данные**. В появившемся списке нужно выбрать операцию *«Проверка данных...»* и задать условия проверки.

Excel 2007 предоставляет мощные и удобные инструменты условного форматирования. Такое форматирование удобно для анализа данных — можно залить ячейки с данными разными цветами так, что каждый цвет будет соответствовать определенным данным. Это поможет для выявления проблемных мест одним взглядом.

Для применения условного форматирования служит кнопка *Условное форматирование* на панели **Стили** ленты **Главная**.

Инструменты **Сортировки и фильтрации** очень помогают в работе, если таблицы содержат большое количество данных, которые представлены в виде списка. Данные в столбцах должны быть однотипными. Список не должен содержать пустых строк или столбцов. Если в списке присутствуют заголовки, то они должны быть отформатированы другим образом, нежели остальные элементы списка.

Упорядочивание списков или сортировка облегчает поиск информации в таблице, так как отсортированные записи отображаются в порядке, определенном значениями столбцов (по алфавиту, по возрастанию (убыванию) цены и пр.).

Для того чтобы произвести сортировку списка, нужно выделить его, а затем нажать кнопку *Сортировка и фильтр* на панели **Редактирование** ленты **Главная**.

Для сортировки списка по нескольким полям предназначен пункт *«Настраиваемая сортировка...»*.

При фильтрации в отличии от сортировки показываются не все данные списка, а только те, которые удовлетворяют условиям фильтрации.

Фильтры бывают двух типов: *обычный фильтр*, или автофильтр, и *расширенный фильтр*.

Для применения автофильтра выделите выбранный диапазон ячеек и нажмите ту же кнопку, что и при сортировке — *Сортировка и фильтр* и выберите пункт *Фильтр*.

Более сложные условия отбора формируются с помощью предназначен пунктов *Текстовые фильтры* или *Числовые фильтры*.

Связанные таблицы — это набор данных, которыми можно управлять как единым целым.

Для создания связанной таблицы предназначена кнопка *Форматировать как таблицу* на панели **Стили** ленты **Главная**.

Нужно выбрать стиль будущей таблицы и задать диапазон ячеек, на основе которого будет создана связанная таблица.

Для создания диаграммы необходимо воспользоваться инструментами панели **Диаграммы** ленты **Вставка**.

В появившемся окне выбрать тип предложенной диаграммы. После этого надо указать диапазон данных для построения диаграммы. Если данные берутся из всей таблицы, то достаточно указать любую ячейку таблицы.

Вычисления в таблицах. Возможность использования формул и функций является одним из важнейших свойств программы Excel.

Текст формулы, которая вводится в ячейку таблицы, должен начинаться со знака равенства. После знака равенства в ячейку записывается математическое выражение. В качества аргументов в формуле используются стандартные компьютерные символы операций. Например: + (сложение); – (вычитание); * (умножение); / (деление), заключение в скобки, больше и меньше.

Формула может содержать ссылки на ячейки, которые расположены на другом рабочем листе или даже в таблице другого файла. Однажды введенная формула может быть в любое время модифицирована.

Программа Excel позволяет работать со сложными формулами, которые содержат несколько операций.

Создание простейших формул. Пусть число, которое находится в ячейке A1 таблицы, нужно умножить на число в ячейке B1 и результат поместить в ячейку C1. Для этого нужно активизировать ячейку C1, нажать знак =, щелкнуть по ячейке A1, ввести знак операции, щелкнуть по ячейке B1 и завершить ввод формулы кнопкой ENTER на клавиатуре, или щелкнув зеленую галочку Enter на панели меню. Формула может содержать несколько ячеек и операций.

Ссылка на ячейки в формулах и функциях может производиться с использованием относительных и абсолютных координат.

Относительные ссылки. В формуле A1 + B1, записанной в ячейку C1, использована относительная ссылка, которая основана на относи-

тельной позиции ячейки, содержащей формулу, и ячеек, на которую указывает ссылка.

При копировании формулы в ячейку C2 автоматически изменяется и ссылка на A2 + B2:

Абсолютные ссылки. При абсолютной адресации на ячейку в формуле происходит ссылка на ячейку, которая расположена в определенном месте. Абсолютная ссылка на ячейку A1, записывается A1, и при изменении позиции ячейки, содержащей формулу, абсолютная ссылка не изменяется. По умолчанию в новых формулах используются относительные ссылки.

Смешанные ссылки. Смешанная ссылка на ячейку в формуле может содержать либо абсолютно адресуемый столбец и относительно адресуемую строку, либо наоборот. Абсолютная ссылка на столбец имеет вид: $A1, $B1 и т.д., а на строку: A$1, B$1 и т.д. При переходе на другую ячейку, содержащую формулу, относительная ссылка изменяется, а абсолютная ссылка нет.

6.3. Базы данных

База данных — средство организации хранения и управления большим количеством упорядоченной разнородной информации.

Ее характеризуют жесткая внутренняя структура и взаимосвязь между отдельными элементами хранящихся данных. И вместо того чтобы иметь дело с большим количеством отдельных файлов, например текстовых, табличных и графических, мы оперируем единым интерфейсом, посредством которого добавляем новые записи, редактируем или удаляем уже имеющиеся. Кроме того, база данных подразумевает наличие механизма составления аналитических отчетов, который избавляет пользователя от расчета каких-либо сложных показателей вручную и поиска необходимых фрагментов в различных файлах.

СУБД (система управления базами данных) использует несколько моделей данных: иерархическую, сетевую (с 1960-х годов) и реляционную (с 1970-х). Основное различие данных моделей в представлении взаимосвязей между объектами.

- Иерархическая модель данных строится по принципу иерархии объектов, т.е. один тип объекта является главным, все нижележащие — подчиненными. Иногда для некоторого главного типа существует несколько подчиненных типов объектов.

- Сетевая модель данных строится по принципу «главный и подчиненный тип одновременно», т.е. любой тип данных одновременно может быть главным и подчиненным.
- Реляционная модель данных строится по принципу взаимосвязанных таблиц. Все современные СУБД поддерживают реляционную модель данных.

Базы данных бывают «настольными» и распределенными. Понятие «настольная» СУБД указывает на то, что все операции с базой данных осуществляются на локальном компьютере пользователя. Именно здесь находится физическое место хранения информации, а также работают средства управления и организации запросов. Противоположностью настольной системе является распределенная база данных, т.е. такая архитектура, при которой ядро БД работает на выделенном сервере; там же обычно хранятся и данные. Через локальную или глобальную сеть пользователь посредством установленного на своем компьютере программного обеспечения посылает запросы и получает ответы. Такие системы предназначены для работы с большим количеством клиентов, и зачастую в качестве серверов в них функционируют компьютеры более сложные и мощные, чем персональные.

База данных и системы управления базами данных — это не одно и то же. База данных — это файл, в котором хранятся в специальном формате данные, а СУБД — это программа, с помощью которой в базу данных может быть введена информация и производятся какие-либо действия над этими данными: просмотр, сортировка, фильтрация, поиск и т.д. **СУБД** — это комплекс программных средств, предназначенных для создания структуры новой базы, наполнения ее содержимым, редактирования содержимого и визуализации информации.

Базу данных можно представить в виде таблицы с конечным числом столбцов и неопределенным числом строк. Примером базы данных может служить классный журнал или итоговая ведомость, куда заносятся оценки за четверть. Количество столбцов ограниченно — их столько, сколько предметов изучается, а количество учащихся может изменяться. Пустая ведомость — это база данных, из которой удалены все записи, но остаются названия полей базы, т.е. ее структура. При создании базы данных необходимо сначала разработать ее структуру, а затем заполнить ее информацией — эти функции и выполняются с помощью СУБД. Вторым этапом является ввод и редактирование записей в таблицу. БД считается созданной, даже если она пустая.

Столбцы в базе данных называют **полями**, а **строки — записями**. Каждое **поле** имеет свое **имя** и содержит отдельный элемент информации. Для каждого поля необходимо указывать его **имя**, **тип данных, размер.** От типа и размера поля зависят скорость доступа к БД и объем файла. **Тип данных** поля определяется значениями, которые предполагается вводить в поле.

СУБД Access входит в состав Microsoft Office и предназначена для работы с **реляционными БД**, т.е. представленными в табличной форме.

Access во многом похож на Excel. Прежде всего обе программы являются продуктами для Windows, следовательно можно использовать свой опыт применения специфичных для Windows соглашений. Данные таблицы или запроса Access отображаются в виде электронной таблицы, которую называют *таблицей данных*. Размер строк и столбцов таблицы данных можно изменять так же, как в рабочих таблицах Excel. Режим ввода данных Access ничем не отличается от аналогичного режима Excel. Основное различие между таблицей базы данных (БД) и электронной таблицей — в системе адресации; в электронной таблице адресуется каждая ячейка, а в таблице БД — только поля текущей записи. В электронной таблице каждая ячейка обрабатывается индивидуально, а в таблице БД обработка идет по записям, причем записи обрабатываются однотипным образом, что позволяет повысить скорость обработки и количество обслуживаемой информации. Характерной особенностью баз данных, созданных в Access, является хранение создаваемых таблиц и средств для обработки данных в одном файле, имеющем расширение .mdb. Достоинством Access является возможность создания СУБД (т.е. программы управления) без программирования. Однако для сложных СУБД применение программирования на встроенном языке Visual Basic for Applications (VBA) позволяет повысить эффективность системы управления.

Основные объекты окна БД имеют следующее назначение:

- **таблица** — основное средство для хранения информации в БД;
- **запрос** — это инструмент для извлечения необходимой информации из исходных таблиц и представления ее в удобной форме;
- **форма** — это основное средство для ввода данных, управления СУБД и вывода результатов на экран монитора;
- **отчет** — это специальное средство для формирования выходных документов и вывода их на принтер;
- **макросы** в Access представляют собой совокупность внутренних команд, предназначенных для автоматизации работы с БД;
- **модули** являются программами, создаваемыми средствами языка VBA, и похожи на макросы в Word и Excel.

6.4. СУБД Access

Технология разработки СУБД содержит несколько этапов, основными из которых являются:

- проектирование структуры БД и связей между таблицами;
- разработка структуры отдельных таблиц и ввод данных в таблицы;
- разработка запросов;
- разработка схемы данных, реализующей запроектированные связи между таблицами и запросами;
- разработка макросов и программных модулей для управления БД;
- разработка форм для реализации интерфейса управления БД;
- разработка отчетов для печати документов.

Приведенная последовательность этапов не является жесткой. Обычно разработчику СУБД приходится многократно возвращаться к одним и тем же этапам, постепенно уточняя проект.

СУБД Access создает и обрабатывает реляционные базы данных, т.е. она позволяет хранить данные не в одной, а в нескольких таблицах и устанавливать связь между ними. Такие таблицы называются связанные, т.е. объединенные в единую базу. Для задания связи таблицы должны иметь поля с одинаковым типом данных. Связь между таблицами устанавливает отношение между совпадающими значениями в этих полях. Такая организация позволяет уменьшить избыточность хранимых данных, упрощает их ввод, удаление, поиск.

Для установления связи между таблицами необходимо, чтобы:

- связываемые поля имели одинаковый тип данных. Исключение составляет поле счетчик, так как поле счетчик может быть связано с числовым полем, имеющим размер длинное целое;
- таблицы хранились внутри одной БД;
- главная таблица связывалась с подчиненной по ключу.

Для обеспечения целостности данных для связанных таблиц нужно помнить, что:

- в подчиненную таблицу не может быть добавлена запись с несуществующим в главной таблице ключом связи;
- в главной таблице нельзя удалить запись, если не удалены связанные с ней записи в подчиненной таблице;
- изменение значений ключа связи главной таблицы должно приводить к изменению соответствующих значений в записях подчиненной таблицы.

Та таблица, от которой идет связь, называется главной, а таблица, к которой эта связь ведет, — подчиненной. Для примера рассмотрим несколько таблиц.

1. Список учащихся школы.
2. Сведения о родителях.
3. Успеваемость за каждую четверть.
4. Учет полученных учебников.
5. Учет занятости в кружках.

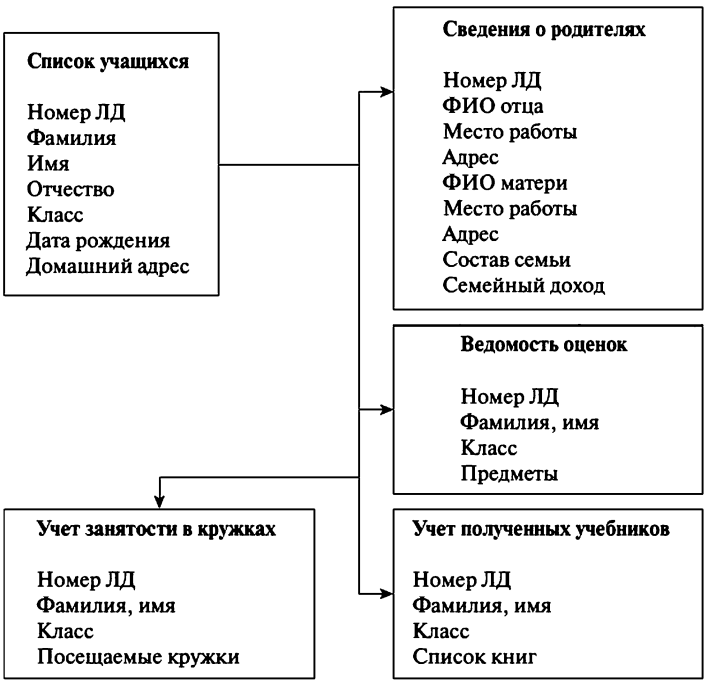


Рис. 6.5

Каждая таблица имеет свои уникальные поля: например, в таблицу «Список учащихся» не следует заносить сведения о полученных теннисных ракетках, а в таблицу «Успеваемость за четверть» — сведения о доходах родителей. Поля таблицы определяют ее структуру и свойства данных, записываемых в ячейках.

В каждой таблице должно быть уникальное поле, с помощью которого можно связать таблицы между собой, — этим полем в нашем случае может стать поле «Номер личного дела учащегося». Такое поле называется полем первичного ключа, или первичным ключом.

Для того чтобы понять принципы разработки реляционных баз данных, требуется дать определения различных типов реляционных ключей и таблиц.

Базовая таблица. В реляционной базе данных базовой таблицей называется таблица, включающая один или несколько столбцов свойств объекта и содержащая **первичный ключ**, который однозначно определяет этот объект. Более того, базовая таблица должна содержать первичный ключ. Базовые таблицы часто называют **первичными**, поскольку они имеют **первичный ключ**.

Промежуточная таблица. Таблица, не являющаяся базовой (так как она не объединяет свойства объекта или не содержит поле первичного ключа), которая используется для обеспечения связей между другими таблицами, называется **таблицей отношений**. Ключевые поля в таблицах отношений должны быть **внешними ключами**, связанными с первичными ключами базовой таблицы. Проще говоря, таблица отношений состоит только из внешних ключей и не содержит независимых элементов данных.

Первичный ключ. Первичный ключ состоит из набора значений, которые однозначно определяют запись базовой таблицы. Любому значению первичного ключа должна соответствовать одна и только одна строка таблицы. Первичный ключ включает одно поле только в том случае, если это поле не содержит повторяющихся значений.

Составные ключи. Если для выполнения условий, накладываемых на значения первичного ключа, заданный ключ включает несколько полей таблицы, то тогда он называется **составным**.

Внешние ключи. Внешний ключ — это столбец, значения которого соответствуют значениям первичного ключа другой связанной таблицы.

В нашем случае главной таблицей будет таблица Список, а все остальные таблицы будут подчиненными.

В программе предусмотрены пять возможностей создания таблицы:

1) **импорт таблиц** из другой базы. В зависимости от обстоятельств из импортируемой таблицы может поступить структура полей, их названия и свойства, а также и содержимое базы. Необходимые правки вносят вручную;
2) режим **Связь с таблицами** применяется в тех случаях, когда речь идет о чужой таблице, которая находится на удаленном сервере и которую нельзя импортировать целиком. Это напоминает подключение к таблице для совместного использования ее данных;
3) **Мастер таблиц.** Это программа, ускоряющая создание структуры таблицы. Мастер задает ряд вопросов и, руководствуясь

полученными ответами, создает структуру таблицы автоматически;

4) **Режим таблицы** открывает заготовку, в которой все поля имеют формальные имена: Поле1, Поле2... и т.д. и один стандартный текстовый тип. Такую таблицу можно сразу наполнять информацией;
5) **Конструктор** предоставляет возможность одновременно задавать поля будущей таблицы и устанавливать свойства этих полей.

Первоначальную загрузку таблицы БД и выполнять их просмотр позволяют осуществлять Формы, которые также помогают производить **корректировку данных**. Форма позволяет отображать одновременно все поля одной или нескольких записей. В форме каждое поле можно разместить в заданном месте, выбрав для него цвет или заливку и добавив элементы управления текстом для эффективного ввода данных. В форму могут быть вставлены такие объекты, как рисунки и графики. Форма может быть:

- однотабличной — источник данных одна таблица или запрос;
- многотабличной, т.е. построенная на основе нескольких взаимосвязанных таблиц;
- без источника данных (чаще всего используется для реализации пользовательского кнопочного меню).

Мощным средством обработки данных, хранимых в таблицах Access, являются запросы. С их помощью можно просматривать, анализировать и изменять данные из нескольких таблиц, а также можно использовать запросы в качестве источника данных для форм и отчетов. Запросы позволяют вычислять итоговые значения и выводить их в компактном формате, подобном формату электронной таблицы, а также выполнять, вычисления над группами записей.

В Access можно создавать следующие типы запросов.

Запрос на выборку. При его выполнении данные, удовлетворяющие условиям отбора, выбираются из одной или из нескольких таблиц и выводятся в определенном порядке. Простые запросы на выборку практически не отличаются от фильтров, которые можно сохранять как запросы. Этот запрос можно использовать, чтобы сгруппировать записи для вычисления сумм, средних значений, пересчета и других действий.

Запрос с параметрами. Это запрос, при выполнении которого в его диалоговом окне пользователю выдается приглашение ввести данные, на основе которых будет выполняться запрос.

Перекрестный запрос. Перекрестные Запросы предназначены для группирования данных и представления их в компактном виде. Позволяют представить большой объем данных в виде, удобном для

восприятия, анализа, сравнения. Могут использоваться в качестве базового при создании отчета.

Отчеты позволяют выбирать из базы данных нужную информацию, оформить ее в виде документа и распечатать. Источником данных могут быть таблица, запрос или несколько взаимосвязанных таблиц. Отчеты и формы похожи, разница в том, что в отличии от форм отчеты не предназначены для ввода и корректировки данных. Отчеты состоят из разделов, подобных разделам форм.

В процессе конструирования отчета формируется состав и содержимое разделов отчета, размещение в нем значений, выводимых из полей связанных таблиц баз данных, формируются заголовки, размещаются вычисляемые поля. Отчет позволяет сгруппировать данные по нескольким уровням, для каждого из которых производится вычисление итогов, определяются заголовки и примечания. В Access существует несколько способов создания отчетов.

6.5. Интерфейс Access 2007

В число новых элементов интерфейса Microsoft Office Access 2007 входит новое стартовое окно:

Рис. 6.6

Access 2007 позволяет быстрее создавать и работать с базами данных, даже если пользователь не имеет навыков их проектирования. При запуске программы в окне **Приступая к работе** предлагается выбор готовых шаблонов. Их можно использовать без каких-либо модификаций или настройки, а можно адаптировать в соответствии с характером информации, которую требуется сохранить в соответствии с желаемым способом ее отслеживания.

Для создания новой таблицы следует щелкнуть мышью на кнопке **Таблица** панели **Создать**.

Access 2007 автоматически определит тип данных во время их ввода. Кнопка **Добавление нового поля** позволяет добавить в таблицу новое поле, которому можно задать требуемый тип данных или форматирование с помощью самонастраивающейся полосы инструментов. Можно также создать новую таблицу, вставив таблицу Excel из буфера обмена, причем программа автоматически настроит все нужные поля и определит их тип данных.

ГЛАВА 7

СОЗДАНИЕ ЭЛЕКТРОННЫХ ПРЕЗЕНТАЦИЙ

7.1. Программа создания презентаций PowerPoint

Презентацией называется набор слайдов, которые содержат информацию на определенную тему и могут сопровождаться устными или печатными комментариями. В Microsoft Office для создания электронных презентаций имеется превосходное средство — PowerPoint. С помощью этой программы мы можем не только подготовить выступление с использованием слайдов, которые демонстрируются на экране, но и создать конспект доклада и материал для раздачи слушателям.

Рассмотрим презентацию как слайд-фильм. Перед тем как его создать, необходимо:

- придумать тему и содержание презентации;
- разбить содержание на логические эпизоды;
- к каждому эпизоду разработать эскиз слайда;
- подобрать рисунки, диаграммы и тексты к каждому слайду.

Окно программы выглядит следующим образом:

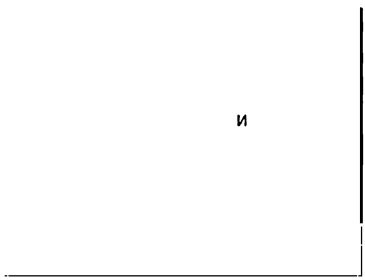


Рис. 7.1

В центре окна находится меню, которое позволяет выбрать способ создания презентации.

Существуют следующие стандартные способы создания презентаций:

- создание пустой презентации с использованием меню **Файл** → → **Создать**;
- создание презентации на основе шаблона оформления;
- создание презентаций с помощью мастера автосодержания.

В левом нижнем углу окна находятся клавиши, которые позволяют выбрать различные режимы работы с презентацией.

1. **Режим слайдов.** В этом режиме на экране изображен слайд, который можно редактировать, переход к следующему слайду осуществляется с помощью правой клавиши прокрутки.
2. **Режим структуры.** На экране изображена структура презентации с описанием слайдов. В правом углу находится уменьшенное изображение выделенного слайда.
3. **Режим сортировщика слайдов.** В этом режиме можно удалять слайды или менять порядок их показа.
4. **Режим страниц заметок.** В этом режиме можно ввести заметки к слайду, а затем распечатать и раздать слушателям.
5. **Показ слайдов.** С помощью этой кнопки можно запустить презентацию, т.е. начать показ слайд-фильма.

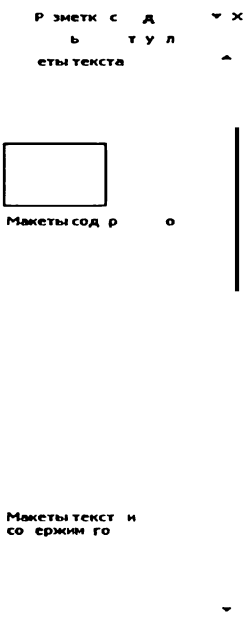


Рис. 7.2.

Для создания собственного слайд-фильма можно воспользоваться пустой презентацией, выполнив **Файл** → **Создать**.

Появится окно выбора авторазметки (рис. 7.2).

Передвигая маркер по примерам авторазметки слайдов, можно выбрать тот вид слайда, который нужен.

Все объекты на слайдах презентаций занимают определенную область, выделенную значками, которые служат для изменения размеров области выделения.

Наиболее распространенным объектом слайдов презентаций — электронных иллюстраций являются текстовые объекты. Они бывают двух видов: обычный текстовый объект и объемный текстовый объект WordArt.

Для того чтобы создать **текстовый объект**, нужно:
- выбрать на панели инструментов или в меню **Вставка** команду **Надпись**;
- выделить прямоугольную область ввода текста перетаскиванием указателя по диагонали области выделения. *При этом область будет ограничена шириной строки и высотой, соответствующей одному интервалу;* выбрать шрифт, начертание символов, цвет символов с помощью команд из строки **Форматирование** и панели **Дополнительные команды форматирования**;
- ввести текст с клавиатуры, а после этого щелкнуть мышкой вне текста.

Для того чтобы ввести в текст **символ**, нужно:
- выделить область ввода текста;
- выбрать  ;
- в таблицах найти нужный символ;
- выделить символ щелчком мышки;
- нажать <Enter>;
- закрыть окно таблицы.

Следует обратить внимание, что на выделенные символы распространяются все команды форматирования.

Для ввода **формул** необходимо:
- выбрать  ;
- опытным путем разобраться с редактором формул;
- набрать формулы;
- закрыть редактор формул.

Формулы вставляются как графический объект. К ним команды форматирования неприменимы.

В заголовках, титрах, колонтитулах электронных иллюстраций часто используют объемные текстовые объекты, которые формирует специальная подпрограмма WordAr. Эти объекты вставляются как графические.

Кроме формул, рисунков ClipGalleri, рисунков из файлов и со сканера, диаграмм и графиков в презентацию PowerPoint можно включить объекты, создаваемые другими приложениями не только Microsoft Offise, но и другими программами.

Самым распространенным объектом слайдов являются **Автофигуры**. Их число не ограничено. Обычный текст и текст WordArt — также автофигуры.

Размещение на слайде таких важных элементов презентации, как таблицы и диаграммы, в сущности мало отличается от вставки любых других объектов. Ведь на самом деле за эти объекты «отвечает»

не PowerPoint, а приложение, в котором их создают, — Microsoft Graph, Word или Excel.

Для создания диаграммы можно выбрать в диалоговом окне Создание слайда автомакет слайда с диаграммой. Если дважды щелкнуть пустую рамку диаграммы или в пустом слайде нажать кнопку Вставить диаграмму, на экране появится диаграмма Microsoft Graph и таблица связанных с ней данных. Можно ввести в таблицу свои данные, соответственно изменится и диаграмма. Щелкнув правой клавишей по области построения диаграммы, можно выбрать другой тип диаграммы.

Создавая слайды, можно также импортировать электронную таблицу или диаграмму Microsoft Excel, а также скопировать данные из другого приложения, для этого выполнить команду **Вставка → Объект** и в появившемся окне выбрать нужный тип готового объекта либо создать новый.

Диаграммы PowerPoint можно «оживить». Это делается с помощью вкладки Эффекты в диаграммах диалогового окна **Настройка анимации**.

Для того чтобы красиво оформить переход от одного слайда презентации к другому, нужно выполнить **Показ слайда → Переход слайда**. И в появившемся окне задать нужные параметры: оформление перехода картинки и звуковое сопровождение.

После того как все слайды презентации будут готовы, нужно настроить саму презентацию, выполнив **Показ слайдов → Настройка презентации**.

Установив соответствующие флажки, можно задать параметры вашей презентации.

Слайды презентации можно распечатать по одному, два и т.д. на странице и использовать в качестве раздаточных материалов. Размещение на странице трех слайдов позволяет включить и разлинованную область для заметок.

Фон слайда можно изменить, выполнив команду **Формат → Фон**, и в появившемся окне выбрать цвет или различные способы заливки: градиентную, текстуру, заливку.

Слайды не рисунки на плакатах. Объекты на них могут появляться и исчезать по ходу изложения материала. Для этого нужно выполнить **Показ слайдов → Настройка анимации**. Появится окно, в котором можно задать анимационные эффекты. С его помощью можно сделать так, чтобы каждый объект на слайде появлялся определенным образом в определенный момент. Это может сопровождаться и звуковым эффектом. Диалоговое окно Настройка анимации делится на три части. Нижняя в свою очередь содержит четыре вкладки. На вкладке Время определяют порядок появления объектов. Для этого нужно

выбрать соответствующий объект из списка Объекты для анимации и установить флажок. Включить. Поскольку объект будет появляться по команде выступающего, нужно также включить режим по щелчку мыши. Объект появится в списке Порядок анимации. Справа от него расположены две кнопки со стрелками, указывающими вверх и вниз. Когда в списке окажется несколько объектов, их можно будет менять местами при помощи этих кнопок. Чем ближе объект к началу списка, тем раньше он появится на экране. Выделить объект, который должен появиться первым и включить анимацию, затем — второй и т.д. Здесь же можно задать способ и время появления анимации: по щелчку мыши или автоматически через указанное количество секунд, эффекты и звук.

В оформлении слайдов можно также использовать и кинофрагменты, и звуковое сопровождение (**Вставка → Кино и звук**).

Если вы намерены сделать презентацию самостоятельным учебным или агитационным, или каким-то другим информационным пособием, можете «подключить» к ней звукозапись вашего выступления. Для этого выберите команду **Показ слайдов → Звукозапись** и щелкните на кнопке OK. PowerPoint перейдет в режим показа презентации. С помощью микрофона вам нужно записать все, что вы собирались изложить публике.

Одно из важнейших преимуществ электронной презентации — возможность быстро менять последовательность слайдов при помощи гиперссылок. Причем гиперссылка не обязательно может указывать на слайд данной презентации, но и может указывать на слайд в другой презентации или вообще в другой документ или приложение. Гиперссылкой можно служить любой объект как текстовый, так и графический.

Если при демонстрации презентации нужно вставить в слайд стрелку для перехода назад или вперед, на следующий слайд, нужно выполнить команду **Показ слайда → Управляющие кнопки** и задать вид кнопки. Управляющую кнопку можно установить в любом месте экрана и настроить на определенной действие.

7.2. Обзор Microsoft PowerPoint 2007

Новая версия программы для создания презентаций от Microsoft является более гибкой с точки зрения пользовательской настройки презентации, а также позволяет создавать более интерактивные, наглядные и красивые презентации. Интерфейс PowerPoint 2007 полно-

стью переработан — его основой теперь является Лента, как и версиях других программ пакета Office2007. Она разделена на функциональные вкладки, каждая из которых имеет свое наименование, а всплывающие подсказки, появляющиеся при наведении курсора, содержат полезную информацию об их возможностях.

Рис. 7.3

В PowerPoint 2007 их заменили семь лент.

1. Главная — содержит команды, необходимые для работы со слайдом презентации (вставка, удаление, форматирование слайда и другие).
2. Вставка — позволяет вставить в презентацию объекты и содержит шаблоны объектов (диаграммы, графики, таблицы, картинки, видео и звук).
3. Дизайн — содержит команды, связанные с оформлением слайда.
4. Аннимация — включает настройки анимации по отношению к объектам слайда и к самим слайдам.
5. Показ слайдов — позволяет выбрать тип показа презентации.
6. Рецензирование — обеспечивает рецензирование слайдов.
7. Вид — содержит функции, меняющие вид слайдов.

Общие принципы создания презентаций в PowerPoint сохранились — на ее слайдах можно разместить текст, графические элементы, видеофрагменты и звуковые файлы и применить к ним анимационные эффекты.

После размещения элементов на слайде и их форматирования необходимо применить эффекты оформления ко всему слайду. С помощью ленты **Дизайн** можно установить параметры страницы и задать ориентацию слайда. Кроме того, к слайдам презентации можно применять темы оформления. Тема представляет собой набор элементов форматирования, которые по умолчанию применяются сразу ко всем слайдам презентации. Тема определяет цвет фона и шрифта, типы шрифтов, способы заливки.

Работа с презентацией осуществляется в режиме редактирования, а для ее просмотра необходимо отобразить презентацию в режиме просмотра. По умолчанию презентация отображается в обычном режиме просмотра презентации, кроме него существует еще несколько режимов редактирования презентаций.

Для оформления слайдов в презентации PowerPoint используется набор стандартных макетов оформления. Пользователь может пополнить стандартный набор макетов. Для этого нужно отредактировать стандартные образцы, внеся в них необходимые изменения.

Печать презентации может осуществляться несколькими способами. По умолчанию презентация распечатывается в виде слайдов, т.е. для каждого слайда отводится одна страница. Кроме этого, презентацию можно превратить в раздаточные материалы, распечатав несколько слайдов на одной странице в соответствии с форматом выдачи. Выдача представляет собой настройки форматов для распечатывания содержимого презентации в режиме нескольких слайдов на листе для создания раздаточных материалов.

На слайды презентации кроме привычных и известных ранее объектов (диаграмма, рисунок, таблица, объект SmartArt) можно также вставить фотоальбом. Фотоальбом PowerPoint 2007 представляет собой презентацию, созданную для отображения фотографий. В фотоальбом можно добавить различные спецэффекты, а также настроить подписи, порядок отображения фотографий, установить рамки вокруг рисунков и применить темы оформления.

Для создания фотоальбома необходимо на ленте **Вставка** панели **Иллюстрации** нажать кнопку **Фотоальбом** и в появившемся меню выбрать **Создать фотоальбом**.

Еще одно важное новшество — функция предварительного просмотра. Она еще до применения выбранного шаблона позволяет понять,

как изменится вид документа или его фрагмента, — достаточно навести курсор на соответствующий шаблон, и вы моментально увидите эффект от применения понравившейся цветовой схемы или шрифта в презентации.

PowerPoint 2007 позволяет также мгновенно увидеть, как выглядит тот или иной эффект анимации при смене слайдов. Вместо названий в раскрывающемся списке представлены наглядные картинки эффектов. Данная функция относится лишь к смене слайдов, а эффекты анимации для объектов слайда можно выбирать только по названиям.

Редактирование таблиц стало менее трудоемким. Одним нажатием можно поменять весь стиль таблицы (изменить цвет строк, столбцов и т.д.). Больше не надо тратить уйму времени на подбор оптимального решения — все уже предусмотрено и нужно только определить, какой из доступных вариантов наиболее удачный.

PowerPoint 2007 сохраняет файлы в форматах на базе XML, имеющие расширение pptx. Документы представляют собой zip-архив, в котором текст хранится отдельно от картинок, стилей, комментариев и прочих элементов. Презентации, сохраненные в новом формате pptx, в среднем имеют объем на 15% меньший, по сравнению с форматом ppt. Презентации можно сохранять и в формате PDF.

ГЛАВА 8

КОММУНИКАЦИОННЫЕ ТЕХНОЛОГИИ

8.1. Компьютерные сети

Для чего нужны компьютерные сети? — Для удобства пользователей ПК. В первую очередь это вызвано практической потребностью пользователей удаленных друг от друга компьютеров в одной и той же информации. Сети предоставляют пользователям возможность не только быстрого обмена информацией, но и совместной работы на принтерах и других периферийных устройствах и даже одновременной обработки документов.

Основные средства связи между компьютерами — линия связи и устройства сопряжения компьютера с линией связи.

Линия связи — это физическая среда передачи информации — важнейшая компонента сети. Основная характеристика линии связи — пропускная способность, т.е. максимальная скорость передачи информации. Измеряется в бит/сек. Поток сообщений линии (ее нагрузка) называется **трафиком**.

В компьютерных сетях могут использоваться следующие линии связи:

- **проводные линии.** Эти линии связи используют в компактных сетях — в пределах офиса или дома;
- **телефонные линии** выполняются из медного провода с пластмассовой изоляцией. Электрический речевой сигнал, проходящий по телефонной линии, постепенно ослабляется («затухает») и поэтому требует усиления в критических точках тракта. Промежуточный усилитель усиливает не только полезный речевой сигнал, но и наложившийся на него шум;
- **радиорелейные линии.** Радиорелейная линия представляет собой цепочку ретрансляторов для радиоволн СВЧ-диапазона. Ретрансляторы устанавливаются на высоких башнях, на расстояниях 50—70 км. Атмосферные и промышленные помехи не оказывают значительного влияния на радиорелейную связь. Промежуточным пунктом радиорелейной линии может служить геостационарный искусственный спутник Земли. Поскольку расстояние

до него составляет около 35 тыс. км, при двусторонних переговорах возникает заметная задержка;

- **волоконно-оптические линии.** По этим линиям по двум парам (одна — резервная) стеклянных оптических волокон толщиной с волос передается огромное количество одновременных двусторонних переговоров. Каждое волокно имеет на одном конце источник света, а на другом — фотоприемник. Источником света обычно служит полупроводниковый лазер или миниатюрный светодиод. Лазер преобразует цифровые электрические сигналы в последовательность импульсов инфракрасного света со скоростью следования от 45 млн до 3,5 млрд бит/с. Фотоприемник снова преобразует последовательность световых импульсов в цифровой электрический сигнал. Оптический кабель не воспринимает электрических помех. Такой кабель считается «супермагистралью» систем связи, поскольку обладает очень большой информационной пропускной способностью.

Назначение **устройства сопряжения** — обеспечить **интерфейс** компьютера с сетью. Примером может служить **модем**.

Модем служит для преобразования цифровых сигналов компьютера в аналоговые сигналы телефонной сети и обратно. Основной характеристикой модема является скорость передачи данных — количество бит информации, передаваемых в секунду.

Существующие сети принято в настоящее время делить по территориальному признаку:

1) локальные сети — охватывают небольшую территорию, действуют в пределах одного учреждения;
2) региональные сети — действуют в пределах города, региона.

Глобальные сети — охватывают большие территории: страны, континенты.

8.2. Конфигурация локальной сети

Для связи с внешними устройствами компьютер имеет *порты*, через которые можно принимать и передавать информацию. Через эти порты можно соединить два компьютера отрезком кабеля, и они образуют сеть, для которой не требуется аппаратное и программное обеспечение. Такое соединение называется *прямым*.

Для соединения компьютеров в сеть можно использовать:

- коаксиальный кабель;
- оптоволоконный кабель;
- неэкранированную витую пару.

Соединить компьютеры можно двумя способами — последовательно и звездой. Конфигурация локальной сети называется *топологией*. Последовательное соединение можно в свою очередь разделить на три типа — чисто последовательное, последовательное кольцом и последовательное по общей шине.

1. *Простое последовательное.*

При этом информация передается последовательно с одного компьютера на другой и обратно. Схема работает быстро, но при разрыве одного из соединений или при неисправности одного компьютера вся сеть выходит из строя. Практически эта схема почти не используется.

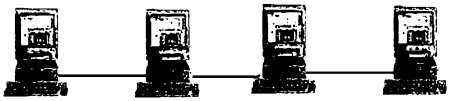


Рис. 8.1

2. *Последовательное кольцом.*

При этом соединении данные также передаются последовательно от компьютера к компьютеру, но по сравнению с простым последовательным соединением данные могут передаваться в двух направлениях, что повышает устойчивость к неполадкам сети. Один разрыв не выводит сеть из строя, но два разрыва делают сеть нерабочей. Кольцевая сеть достаточно широко применяется. В основном из-за высокой скорости передачи данных. Кольцевые сети самые скоростные.

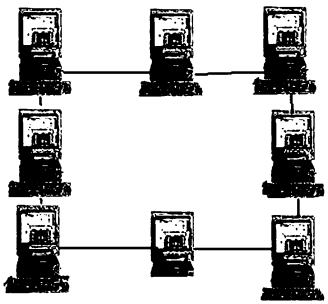


Рис. 8.2

3. *Последовательное по общей шине.*

При таком соединении обмен может производиться между любыми компьютерами сети независимо от остальных. При повреждении связи одного компьютера с общей шиной этот компьютер отключается

от сети, но вся сеть работает. В этом смысле сеть достаточно устойчива, но если повреждается шина, то вся сеть выходит из строя.

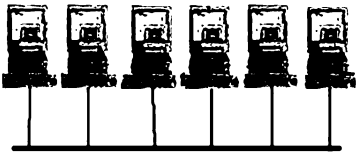


Рис. 8.3

4. *Соединение звездой.*

При соединении звездой сеть очень устойчива к повреждениям. При повреждении одного из соединений от сети отключается только один компьютер. Кроме того, эта схема соединения позволяет создавать сложные разветвленные сети. Устройства, которые позволяют организовывать сложные структуры сетей, называются концентраторами и коммутаторами.

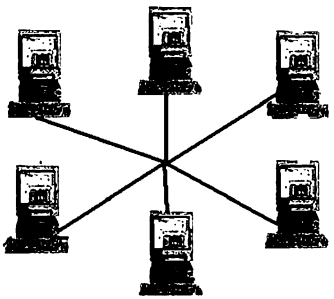


Рис. 8.4

Все указанные схемы могут в свою очередь быть организованы двумя способами. В зависимости от способа организации сети могут быть *одноранговыми и с выделенным сервером.*

Одноранговая сеть построена таким образом, что все компьютеры в сети равноправны. С каждого компьютера есть доступ на каждый компьютер сети.

Сеть с выделенным сервером имеет центральный компьютер — сервер, с которого происходит управление работой сети. Остальные компьютеры называются *рабочими станциями.* Сервер — это компьютер, предоставляющий услуги другим компьютерам сети. При помощи сервера происходит распределение доступа различных пользователей к компьютерам сети и распределение других ресурсов сети. Сеть с выделенным сервером может быть ранжирована, т.е. могут быть выде-

лены компьютеры в сети, к которым будет ограничен доступ с других компьютеров. Кроме того, имеется возможность организовать доступ к общим сетевым принтерам, модемам и другим устройствам с любого компьютера. На сервере могут быть записаны программы, которыми пользуются все компьютеры сети.

Для работы реальной сети требуется, чтобы каждый компьютер имел **сетевую плату.** Сетевая плата вставляется в свободный разъем расширения на системной плате компьютера. К сетевой плате через специальный разъем подключается кабель.

Кроме сетевых плат используется и другое сетевое оборудование — *концентраторы*, *коммутаторы*, *принт-серверы* и другие устройства.

Концентратор — это устройство, которое позволяет соединять в сеть компьютеры по схеме «звезда».

Рис. 8.5

На принципиальной схеме концентратор располагается в центре звезды, а к нему подключаются все компьютеры, компьютеры как бы концентрируются в этом устройстве — отсюда и название.

Каждый компьютер может быть соединен с концентратором кабелем длиной до 200 м. Если расстояние, на котором находится компьютер от концентратора, больше 200 м, то необходимо подключить дополнительный концентратор. Концентраторы могут соединяться между собой прямым кабелем или через другое устройство — *коммутатор.*

Коммутатор — это устройство для подключения концентраторов с целью создания разветвленной сети с множеством сегментов.

При помощи коммутатора можно создать сеть, состоящую из нескольких простых «звезд», а при помощи комбинации коммутатора и концентраторов можно строить сети любой сложности и конфигурации.

Итак, для создания локальной сети необходимо иметь:

- компьютеры;
- провод (разъемы, розетки);
- сетевой адаптер (для каждого компьютера);

- устройство — «центр звезды»;
- hub (мультиповторитель);
- switch (коммутатор).

8.3. Сетевой протокол. Сетевая операционная система

Одной из характеристик компьютерной сети является сетевой протокол. Сетевой протокол — это программа-набор правил, устанавливающих тип используемых данных, стандарты связи, правила обработки ошибок. Существует множество сетевых протоколов — CSMA/CD, SLIP, PPP, UUCP, ISO, OSI, TCP/IP.

Для передачи по сети файл разбивается на несколько частей — пакетов. Каждый пакет передается независимо от остальных. На конечном пункте в компьютере все пакеты собираются в один файл. Так как пакеты передаются независимо, то каждый пакет может дойти до конечного компьютера по своему пути.

Чтобы сеть, состоящая из равноправных компьютеров, работала, каждому компьютеру присваивается имя и в каждый компьютер записывается таблица имен всех компьютеров сети и таблица соединений. Благодаря этим сведениям, каждый компьютер «знает», по какому пути направить пакет. Вначале проверяется кратчайший путь, если он занят или разрушен, то проверяется следующий наиболее короткий путь и т.д. После того как пакеты попадут на адресуемый компьютер, проверяется наличие всех пакетов, составляющих файл. Если какого-либо пакета не хватает, компьютер посылает запрос на компьютер-отправитель и сообщает, какой пакет отсутствует. Нужный пакет заново посылается адресату. Все правила кодирования и пересылки файлов записываются в сетевом протоколе.

Для этого был создан протокол IP, позволявший делить файлы на пакеты и передавать пакеты от узла к узлу. Для объединения сетей, работающих по протоколу IP и сетей, работающих по другим протоколам, необходимо было создать специальный межсетевой протокол. Этот протокол был создан в 1974 г. и назван TCP. Протокол TCP обеспечивал передачу пакетов между компьютером-отправителем и компьютером-приемником, этот протокол позволял также досылать потерянные пакеты. Все эти свойства протокола TCP позволили использовать его для межсетевого обмена файлами.

После объединения в 1982 г. двух протоколов TCP и IP в один протокол TCP/IP стал стандартным протоколом объединенной сети —

Интернет. Сетевой протокол записывается в компьютер в виде специальной программы.

В тех случаях, когда происходит объединение сетей, работающих по разным протоколам, возникает необходимость для перевода данных из формата, принятого в одной сети, в формат, принятой в другой. Компьютеры или программы, выполняющие эту функцию, называют **шлюзами**. Если объединяются две сети, использующие одинаковые протоколы, то оборудование, стоящее между ними, называется **мостами.** Нередко владельцы сетей (например, банковских) подключаются к глобальным сетям, чтобы иметь широкие возможности связи, но не могут допустить подключения внешних пользователей к своей сети. В этом случае шлюзовой компьютер выполняет защитную роль и называется **брандмауэром**. Через него может проходить только разрешенная информация. (В своем первоначальном значении брандмауэр — это сплошная каменная стена, проходящая насквозь через деревянные постройки для защиты от пожаров.)

Кроме сетевого оборудования для работы сети требуется **сетевая операционная система**. По сравнению с обычной операционной системой в сетевой имеются возможности работы в сети. К сетевым операционным системам относятся Windows 10, NetWare, UNIX и др. Локальные компьютерные сети можно объединять друг с другом, даже если между ними большие расстояния. Правда, при этом используют не только специальные соединения, но и другие каналы связи. Разница между ними только в надежности (в уровне помех), скорости передачи данных (пропускная способность линии) и стоимости использования канала связи. Как правило, чем лучше линия, тем дороже стоит ее аренда, но тем больше данных можно пропустить по ней в единицу времени. При соединении двух или более сетей между собой возникает *межсетевое соединение* и образуется *глобальная компьютерная сеть.*

Глобальная компьютерная сеть — это сеть, объединяющая компьютеры, находящиеся на больших территориях в масштабе региона, целой страны, группы стран или всего мира.

8.4. Глобальная сеть Интернет

Толчком создания Интернет явился запуск в Советском Союзе в 1957 г. первого искусственного спутника, в котором Соединенные Штаты увидели для себя угрозу использования ракет для нанесения ядерного удара по США. Ядерный взрыв был способен нарушить работу компьютеров и оставить вооруженные силы без управления и свя-

зи. В этом же 1957 г., при департаменте обороны США было создано Агентство по научно-исследовательским проектам — ARPA. Перед учеными была поставлена задача создания компьютерной сети, которой могли бы пользоваться военные при ядерном нападении на страну. Сеть должна была использоваться для связи между командными пунктами системы обороны.

В 1972 году в Вашингтоне прошла первая Международная конференция по компьютерным коммуникациям. На конференции присутствовали ученые из десяти стран. Участникам конференции была представлена сеть Арпанет. Это было первое публичное представление Арпанет. Сеть Арпанет была первой глобальной сетью, поэтому к ней стали присоединяться другие сети, созданные образовательными, научными и правительственными организациями.

Россия впервые получила доступ к Интернету в начале 80-х гг. Доступ был осуществлен Институтом атомной энергии им. И.В. Курчатова.

Сегодня Интернет — это объединение большого количества сетей. Каждая сеть состоит из десятков и сотен серверов. Серверы соединены между собой напрямую различными линиями связи: кабельными, наземной радиосвязью, спутниковой радиосвязью. К каждому серверу подключается большое количество компьютеров и локальных компьютерных сетей, которые являются клиентами сети. Клиенты могут соединяться с сервером не только по прямым линиям, но и по обычным телефонным каналам.

Интернет предоставляет людям огромные информационные ресурсы по различным областям знания, жизни человека и общества. Тексты, базы данных, графические изображения, музыкальные и видеофрагменты, которые хранятся на серверах Интернета, могут удовлетворить интерес и потребности большинства пользователей сети, а ведь объем информации в Интернете растет с каждым часом! Пользователю доступны различные **сервисы всемирного информационного пространства**, такие как World Wide Web — всемирная паутина — глобальная распределенная информационная гипертекстовая мультимедиа-система, Поисковые службы (роботы, каталоги), Файловые архивы — хранилища программного обеспечения, музыки, фильмов. Следует отметить и сервис **Вики/Открытый код** (Wiki/Open Source). Это две технологии, которые дают возможность пользователю превратиться в активного создателя, а не только в пассивного потребителя. Движение «Открытый код» представляет собой целую культуру с применениями во многих других областях помимо технологий. Суть движения в том, что код, используемый для написания программ, доступен всем, и, таким образом, миллионы программистов-разработчиков по всему

миру могут его регулярно совершенствовать. Лучший пример — продукты Mozilla (браузер Firefox, почтовый клиент Thunderbird, календарь Sunbird). Первая вики-сеть, «Портлендское хранилище образцов» (образцов программного кода), была создана 25 марта 1995 г. программистом Уордом Каннингемом. Слово wiki (произносится [*ви́ки*]), точнее wiki-wiki, он заимствовал из гавайского языка, на котором оно означает «быстро». «Вики» — это специальный тип веб-сайтов, который позволяет пользователям самим менять его содержание веб-сайта (добавлять или редактировать статьи). Наиболее известным представителем этого движения является энциклопедия Wikipedia, соавтором и редактором которой может стать каждый пользователь. Преимущество Wikipedia перед обычной энциклопедией в том, что обычные пользователи смогут написать практически бесконечное число статей на забытые или малоинтересные темы.

Широко распространены также **сервисы коммуникации** на основе Интернет, такие как электронная почта — e-mail, форумы, сервис хранения цифровых фотографий, общение в реальном времени: чаты, ICQ, видеоконференции. Такой сервис, как «Живой журнал» (веблоги или блоги), произвел революцию не только в среде программистов, но и в политике, бизнесе (особенно маркетинге) и массмедиа. Блоги — главная угроза традиционным источникам массмедиа. Например, некоторые политические блоги в США имеют больше читателей, чем десять самых популярных американских газет вместе взятые. Поисковик блогов — Technorati — один из самых интересный проектов последнего времени, который уже сейчас многие называют «новым Google». Блоги — это персональные журналы с широким применением. Кто-то держит их для собственного удовольствия и описывает там самые знаменательные события своей жизни, кто-то публикует на блогах ссылки на особо понравившиеся статьи, кто-то пишет о наиболее интересных бизнес-решениях и технологиях.

Интернет обладает огромными возможностями по организации связи и общения между людьми. Кроме электронной почты сейчас существует возможность голосового общения по Интернету — интернет-телефон. Более того, технология Интернет обогнала обычные телекоммуникационные технологии и предоставила реальную возможность осуществления видеотелефонной связи. При помощи цифровых видеокамер можно организовать *видеоконференции* — общение с одним или несколькими пользователями сети. Во время видеоконференции вы видите собеседника на экране своего монитора. Это очень удобный способ общения, но он требует определенных условий — высокоскоростных линий связи и цифровых видеокамер у всех собеседников.

Если пользователи сети не имеют таких условий, то они могут организовать общение или обсуждение проблем при помощи *телеконференций*. В современную телевизионную эпоху термин «телеконференция» сразу ассоциируется с телевидением, но на самом деле телеконференции ничего общего с телевидением не имеют.

Телеконференция — это способ организации общения в Интернет при помощи текстовых сообщений. Телеконференция базируется на системе электронной почты. Чтобы организовать телеконференцию, необходимо иметь интернет-сервер и почтовый ящик. На сервере публикуется тема телеконференции. Все желающие обсудить тему пересылают свои отзывы по электронной почте. Письма автоматически помещаются на сервер. Так постепенно на сервере появляются тексты с обсуждением темы. Дискуссию по теме может просмотреть любой посетитель сервера. Если тема его заинтересует, он может принять участие в телеконференции.

Кроме телеконференций существуют *доски объявлений*.

Доска объявлений отличается от телеконференции только тем, что нет конкретной темы сообщений, и каждый, кто хочет сделать сообщение, пересылает его по электронной почте.

Все больше появляется фирм, предоставляющих провайдерские услуги, и задача пользователя — выбрать тот тип подключения, который ему удобен.

Использование модемов связано с некоторыми неудобствами — маленькая скорость и занятый телефон.

На смену модему приходят выделенные линии и **беспроводные сети (Wi-Fi** или **радиоинтернет**). Применение выделенных линий значительно увеличивает скорость передачи, а беспроводная связь делает компьютер более мобильным — такие технологии незаменимы для обладателей ноутбуков.

Сейчас можно выйти в Интернет через сотовый телефон или даже фотокамеру (можно отправить фотографию, не подключаясь к компьютеру).

Эти возможности помогают пользователю работать в **Интернете** в любое время и в любом месте.

8.5. История создания и назначение Web-системы

В 1991 году была практически реализована гипермедийная система, названная World Wide Web. Дословно это название переводится как «Паутина шириной во весь мир», но обычно ее называют Всемирной

паутиной, или Web-системой (Веб-системой), или WWW (читается — три-дабл-ю), или W3.

Web-система состоит из серверов, организованных по Web-технологии (Web-серверов). Эта технология позволяет представлять информацию в графическом виде с красивыми иллюстрациями, мультипликацией, видео и звуковыми эффектами — другими словами, использовать все мультимедийные возможности компьютера. Кроме того, Web-система использует гиперссылки на объекты, которые находятся на разных серверах в различных точках сети. В Web-системе используется специальный компьютерный язык HTML и протокол HTTP — протокол передачи гипертекста.

HTML — это специальный язык форматирования текстовых документов. Важно отметить, что речь идет об *электронных*, а не печатных документах. Этот язык создан, чтобы оформлять экранные документы (сайты), которые будут просматриваться неизвестными средствами.

Web-система сделала сеть Интернет привлекательной и способствовала росту ее популярности.

Информация на Web-серверах размещается в виде Web-сайтов. Web-сайт — это способ организации информации на Web-сервере. Web-сайт состоит из одной или нескольких Web-страниц. Web-страница — это информация, размещенная на Web-сервере, которая может быть просмотрена на экране без перелистывания. Возможна только прокрутка содержимого.

Адресация в Интернете. Смысл адреса состоит в том, чтобы с гарантией привести любого желающего в определенное место. Например, имея верный почтовый адрес человека, вы можете отправиться к нему в гости, не боясь при этом, что вы попадете к кому-нибудь другому. Аналогичным образом обстоит дело и с адресами в Интернете.

IP-адрес. Чтобы можно было однозначно обозначить любой компьютер в Интернете, применяется специальная система адресов, называемая IP-адресами. При пересылке информации протоколами TCP/IP используются присвоенные адреса. Каждый компьютер получает свой уникальный адрес. Цифровые адреса в Интернете состоят из четырех целых чисел, каждое из которых не превышает двухсот пятидесяти шести. При записи числа отделяются друг от друга точками, например: 194.84.93.10 или 200.5.78.175. Начало адреса определяет часть Интернета, к которой подключен компьютер, а окончание — номер компьютера в этой части сети.

Компьютеры при пересылке информации использует цифровые адреса, а пользователи в работе с Интернетом используют в основном символьные имена. Потому что людям трудно запомнить адреса, вы-

раженные числами, — им удобнее работать с символьными именами. А серверам Всемирной сети удобнее работать с числами, которые проще сравнивать, и по ним нетрудно определить направление, в котором надо переслать очередной TCP-пакет. Адреса в Интернете могут быть представлены как последовательностью цифр, так и именем, построенным по определенным правилам.

Система доменных имен. Адреса в Интернет строятся по доменной системе адресации (domain name system, DNS). Каждый из десятков миллионов компьютеров, входящих в Интернет, имеет свой собственный уникальный доменный адрес (domain address), часто называемый также доменным именем (domain name) компьютера или просто именем узла (host name).

В мире нет стран с одинаковыми названиями. Таким образом, адрес на основе географических и административных названий однозначно определяет точку назначения.

Виды доменов верхнего уровня и их предназначение. Каждый адрес состоит из нескольких уровней. Составные части доменного адреса называются сегментами и образуют иерархическую систему. Домены бывают географическими (закрепленными за какой-нибудь страной) и экстерриториальными (общемировые). То есть имена доменных зон условно можно разделить на «организационные» и «географические».

Каждая страна (государство) имеет свой географический домен из двух букв: для России это домен ru, для Австралии — au, для Бельгии — be Англии — uk и т.д.

Структура доменных имен. Внутри каждого доменного имени первого уровня находится целый ряд доменных имен второго уровня. Домен верхнего уровня располагается в имени правее, а домен нижнего уровня — левее.

Например, компьютер www.nnn.spb.ru, принадлежит одновременно к домену nnn (nnn.spb.ru), к домену Петербурга (spb.ru) и к домену России (ru). Домены Интернета, как матрешки, вкладываются друг в друга, и чем мельче домен, тем из большего числа сегментов состоит его обозначение.

Универсальные указатели ресурсов. Не только компьютер или пользователь, но и любой файл, объявленный владельцем машины доступным для сети, можно адресовать. При работе в Интернете чаще всего используются не просто доменные адреса, а универсальные указатели ресурсов называемые URL (Universal Resource Locator). Если необходим какой-нибудь конкретный документ, то нужно знать, где он лежит, — нужен его точный адрес.

В указателе кроме собственного адреса имеются сведения о том, каким протоколом следует обращаться к данному ресурсу, какую программу для этого следует запустить на сервере и к какому конкретному файлу следует обратиться на сервере.

Примером указателя может быть http://www.nnnn.ru/ie.

URL-адрес состоит из трех частей. Рассмотрим их по порядку.

1. Первая часть адреса описывает транспортный протокол, который будет использоваться для пересылки данных. В нашем случае это http:// — протокол HTTP (Hyper Text Transport Protocol — протокол передачи гипертекста). При работе с файлами эта часть адреса будет выглядеть так: ftp:// (File Transfer Protocol — протокол передачи файлов).

2. Вторая часть адреса — это так называемое доменное имя сервера в сети Интернет. В нашем примере домен первого уровня (.ru) говорит о том, что сервер находится в России (Russia). Доменное имя второго уровня обычно говорит об организации, которой принадлежит данный сервер. После имени домена второго уровня идет имя компьютера, но вместо него часто используются буквы www (от World Wide Web).

3. Третья часть адреса указывает, где ресурс размещен в компьютере. Таким образом, в третьей части адреса просто перечислены папки, в которые вложен нужный файл документа, и имя самого файла.

Например: http://www.nowoth.ru/obr/vacans/index.html. Этот адрес говорит, что на сервере www.nowoth.ru имеется папка obr, в это папку вложена папка vacans, а в ней находится файл index.htm, который нам нужен. Таким образом, если указать этот адрес в поле адреса браузера, то в итоге вы загрузите файл гипертекстового документа index.htm.

Указатель ftp: //www.nowoth.ru/vacans/index.html описывает, что к файлу index.html, расположенному в каталоге vacans на сервере www.nowoth.ru, следует обратиться по протоколу передачи файлов FTP.

Компьютеры в Интернете работают под управлением операционных систем, отличных от DOS, поэтому расширения файлов могут состоять из большого количества букв.

Адрес электронной почты. Основой любой почтовой службы является система адресов. Без точного адреса невозможно доставить почту адресату. В Интернете принята система адресов, которая базируется на доменном адресе машины, подключенной к сети. Адрес состоит из двух частей: имени пользователя или компьютера и имени домена, разделенных символом @; если речь заходит о компьютере-сервере, первая часть может отсутствовать. Почтовый адрес обычно состоит из полного или сокращенного названия организации и ссылки на регион либо сферу деятельности, а также может отражать, через какие узлы идет передача информации. Кроме того, по стандартам Интерне-

та может быть записан и адрес в другой сети у сообщения, предназначенного для передачи через шлюз, например, адрес ПЕТРОВА_Nina@p14.f521.n1234.z4.fidonet.org соответствует адресу 4:1234/521.14 во всемирной любительской сети FIDOnet.

8.6. Коммуникационные программы

Для подключения к Интернету необходимы кроме технических средств для подключения к каналу связи (для телефонной линии — модем) и компьютера с установленной операционной системой специальные коммуникационные программы.

Браузер — это интерактивная программа, позволяющая организовать доступ в Web-пространство и Интернет. Самые популярные браузеры:

- Microsoft Internet Explorer;
- Google Chrome;
- Mozilla FireFox;
- Opera;
- Safari.

В настоящее время самыми популярными браузерами является Internet Explorer корпорации Microsoft и Netscape Communicator корпорации Netscape Communication.

Сущностью работы в Интернет является доступ со своего компьютера на различные Интернет-серверы и считывание информации с этих серверов в свой компьютер.

Почтовые программы. Существует очень много почтовых программ, значительная часть из них бесплатна. Наиболее распространенные следующие программы для Windows:

- *Microsoft Outlook Express* — входит в состав Microsoft Office;
- Foxmail;
- The Bat!;
- Mozilla Thunderbird.

При первом запуске почтовой программы автоматически запускается мастер по настройке, который запрашивает у вас минимальную необходимую информацию: ваш адрес и название серверов входящей и исходящей почты. Также может отдельно запрашиваться имя пользователя — это та часть вашего адреса, которая находится слева от знака @. При первой проверке почты и при всех последующих проверках вас приглашают ввести пароль, если вы не пометили галочкой «Запомнить пароль», — при этом надо внимательно следить, чтобы он был введен латинскими буквами.

Почтовые пересылки через браузер. Существует большое количество серверов, которые дают возможность завести бесплатный почтовый ящик и позволяют работать с почтой, используя только браузер. Бесплатные почтовые службы живут за счет доходов от рекламы.

Это использование почты имеет многие достоинства. Например: можно менять провайдеров, не меняя свой адрес электронной почты, можно просматривать почту с любого компьютера, подключенного к Интернету. У этого способа есть и недостатки. А именно, нельзя при работе с почтой через браузер минимизировать время подключения к Интернету, общедоступные почтовые сервера часто перегружены.

Лучше всего комбинировать достоинства обоих способов. Можно завести почтовые ящики и у своего провайдера, и в общедоступной почтовой службе.

Для электронной почты сложились следующие правила этикета.

1. Нельзя посылать очень большие файлы в качестве приложений. Для обмена большими файлами есть другие способы.
2. Будьте взаимно вежливы. Подумайте, прежде чем отправить грубое письмо. Письма, написанные в запале, получили название *флеймов*.
3. Шутки принято обозначать явным образом при помощи *смайликов*: :), :^), :-), :-(.
4. Не стоит цитировать в ответе все письмо, на которое отвечаете.

Нельзя посылать свою рекламу в не предназначенные для этого места.

8.7. Поиск информации в сети Интернет

В 1990 году была создана первая система поиска информации в сети Интернет Арчи, в 1991 г. в университете Минессоты создана система Гофер. Гофер — это иерархическая система меню, позволяющая легко находить информацию, имеющуюся на сервере или на нескольких серверах.

Интернет предоставляет оперативный доступ к информации на любую тему, которая находится на сотнях тысячах информационных серверов. Очень часто у пользователя появляется необходимость найти сведения по какой-либо теме. Для этого предназначены специальные поисковые инструменты, они позволяют найти документы, содержащие какие-либо конкретные слова.

Поиск в Интернете требует определенных навыков. Полезно запомнить следующие советы:

- если вопрос поставлен правильно — то это уже половина ответа;
- легко найти редкую информацию и трудно — широко распространенную;
- необходимо знать особенности поисковых систем;
- для разных запросов нужно использовать разные поисковые инструменты;

Пользователь, задав ключевые слова и активизировав поиск, получает список документов. Этот список сортируется по определенным критериям так, чтобы вверху списка оказались те документы, которые наиболее соответствуют запросу.

Поисковые серверы можно разделить на тематические каталоги, роботы индексов (поисковые машины) и системы мета поиска. Также для поиска необходимой информации в Интернет весьма полезны системы поиска в конференциях Usenet и службы поиска людей.

Единой оптимальной схемы поиска в Интернет не существует. В зависимости от специфики информации для ее поиска вы должны использовать соответствующие поисковые службы. Можно пользоваться какой-нибудь одной поисковой системой, например Rambler, но чем грамотнее подобраны поисковые службы и составлен запрос на поиск информации, тем качественнее будут результаты поиска.

Облегчат поиск информации несколько советов.

1. Правильно поставленный запрос, нужно искать по более редкому, уникальному слову.
2. Нужно стремиться получить малое количество документов, не потерять нужное в большом объеме информации.
3. Нужно пытаться использовать гиперссылки, относящиеся к ключевому слову.

Тематические каталоги очень похожи на библиотечные каталоги, информация в них имеет иерархическую структуру, классифицирована по темам: искусство, наука, образование и т.д. Каталогами пользуются тогда, когда не известно точно, что именно нужно искать. Возможен поиск информации и по ключевому слову.

Наиболее популярным во всем мире признан тематический каталог Yahoo! (http://www.yahoo.com). Он представляет собой огромную базу данных адресов сайтов самой различной тематики. Система Yahoo! англоязычная. Если нужно найти информацию на русском языке, то лучше использовать российские каталоги. Стоит отметить первый российский каталог Russia on the Net (http://www.ru), один из крупнейших русскоязычных каталогов List.RU (http://www.list.ru/), российский ва-

риант Yahoo!, также каталог «Созвездие Интернет», содержащий только избранные (заявлено как самые интересные ресурсы), российский проект «Желтые страницы Интернет» (http://www.piter-press.ru/yp), где, как и в одноименной книге, представлена информация о самых разных ресурсах Интернет.

Поисковые машины устроены несколько иначе. Это сервер с огромной базой данных адресов, который автоматически обращается к страницам WWW по всем этим адресам, изучает содержимое этих страниц, формирует и прописывает ключевые слова со страниц в свою базу данных, т.е. индексирует страницы. Все ссылки помещаются в базу данных сервера, в которой пользователи, используя ключевые слова, находят нужную им информацию. Результат поиска состоит из выдержек рекомендованных пользователю страниц, их адресов (URL), оформленных в виде гиперссылок

Самая популярная поисковая машина AltaVista (http://www.altavista.com) содержит 11 млрд слов, извлеченных из 30 млн WWW-страниц.

Наиболее развитый сервис поиска информации на русском языке предоставляет сервер Яndex (http://www.yandex.ru).

В Яndex можно написать фразу по-русски, описывающую то, что нужно найти. Система самостоятельно проанализирует запрос и постарается найти все, что относится к заданной теме. База Яndex содержит порядка 2 млн документов и постоянно обновляется.

Синтаксис запроса Яndex

Пробел или **&** — логическое И (краткое — в пределах одного абзаца).

Пример: школьное образование.

Результат: все документы, где в пределах одного абзаца встречаются слова «школьное» и «образование».

&& — логическое И (в пределах документа).

Пример: школьное&&образование.

Результат: все документы, где встречаются слова «школьное» и «образование».

, или **|** — логическое ИЛИ.

Пример: школьное, образование.

Результат: все документы, где встречается либо слово «школьное», либо слово «образование».

~ — бинарный оператор И НЕ (в пределах одного абзаца).

() — группирование слов.

Пример: (школьное&&образование) ~газета.

Результат: все документы, где встречаются слова «школьное» и «образование», но не в словосочетании со словом «газета».

/(число) — расстояние в словах, где «число» — число слов между словами в запросе плюс единица.

Пример: школьное/1 образование.

Результат: все документы, где встречается словосочетание «школьное образование».

Очень популярна российская поисковая машина Rambler.

У этого сервера еще более полная база данных адресов URL, чем у Яndex. Использует те же логические операторы И, ИЛИ, НЕ, метасимвол * (аналогично расширяющему диапазон запроса символу * в AltaVista), коэффициентные символы + и – для увеличения или уменьшения значимости вводимых в запрос слов.

Стоит также отметить прочие русскоязычные поисковые машины Апорт.

Поисковые машины и тематические каталоги имеют много общего. У каталогов присутствует возможность поиска информации по строке запроса с использованием логических операторов, а поисковые машины содержат свои собственные тематические каталоги.

Системами метапоиска можно воспользоваться исходя из экономии времени и денег. Эти системы не имеют собственных поисковых инструментов и собственной базы данных. Их основная задача состоит в том, чтобы передать запрос пользователя настоящим поисковым системам. Системы метапоиска позволяют задействовать сразу несколько поисковых служб, при этом у пользователя нет необходимости подключаться к каждой из этих служб и многократно вводить запрос.

Системой такого класса является Search.com (http://www.search.com), а для русскоязычных документов можно использовать Savvy Search (http://guaraldi.cs.colostate.edu:2000/form?lang=russian). Еще одна российская система метапоиска «Следопыт» (http://www.medialingua.ru/www/Wwwsearc.htm) работает только с пятью англоязычными поисковыми машинами и одной русскоязычной, но интересна тем, что переводит ваш русскоязычный запрос на английский язык для англоязычных поисковых машин.

Существуют мультипоисковые страницы:

«Все в одном» (http://www.tpi.ac.ru/~mike/search/index.htm);

Search (http://www.informika.ru/windows/intern/poisk/main.html); и др.

Для продуктивного использования Интернета необходимо уметь искать и копировать нужные файлы, для этого служит **FTP-сервис**.

FTP-архивы являются одним из основных информационных ресурсов сети Интернет. По своей сути это огромный склад программ, рисунков, текстов, которые хранятся в виде файлов на различных компьютерах. При этом компьютерные платформы могут быть различных типов. В этом и заключается главная особенность FTP в сети. Если ваш компьютер имеет FTP и подсоединен к Интернету, то вы получите доступ к огромному числу архивов, хранящихся на других системах.

Если вам нужна определенная программа или документ, но вы не знаете, где он находится, то в поисках вам помогут так называемые Archie-серверы, которые ведут списки файлов многих FTP-серверов, постоянно обновляя свои базы данных. Можно, используя Telnet или программу-клиент на вашей системе, войти в такой Archie-сервер и поискать нужные вам файлы. Если такой файл существуют, то вам будет выдан список FTP-серверов, путей, размер и дата последней модификации файла.

Для поиска информации на FTP-серверах можно использовать популярный русский поисковый сервер:

http://www.filesearch.ru

или зарубежный:

http://ftpsearch.iycos.com

Здесь требуется указывать не ключевые слова, а имя файла.

8.8. Беспроводные сетевые технологии

Беспроводные сетевые технологии Wi-Fi в последнее время находят все большее применение. Домашние компьютеры стали практически бесполезны без подключения к Интернету. Беспроводные сетевые технологи позволяют вам, где бы вы ни находились, быть подключенными к сети, обмениваться данными, находить нужную информацию в Интернете или просто играть.

Технология передачи информации по радиоканалу Wi-Fi была разработана и применялась в локальных сетях корпораций и компаний Силиконовой долины США с середины 1990-х гг. Она обеспечивала связь с мобильным абонентом, которым являлся сотрудник компании, снабженный ноутбуком с беспроводным сетевым адаптером.

Термин Wi-Fi первоначально использовался для обозначения технологии связи в диапазоне 2,4 ГГц и работающей по стандарту IEEE 802.11b (скорость передачи информации — до 11 Мбит/с). В настоящее время этот термин применяется к другим технологиям беспроводных локальных сетей. Самые значимые из них определены стандартами IEEE 802.11a и 802.11g (скорость передачи — до 54 Мбит/с, частотные диапазоны соответственно 5 ГГц и 2,4 ГГц).

Одним из основных преимуществ технологии Wi-Fi является отсутствие телефонных или каких либо других линий связи, основанных на проводах. Передача данных сети осуществляется с помощью радиоволн очень высокой частоты, которые не воздействует на человека и не создают помехи для электронной техники. Отсутствие проводов обеспечивает пользователю мобильность, он не привязан к рабочему месту. Эта технология позволяет предоставить высокоскоростной доступ в Интернет, при этом отсутсвует дозвон, установка соединения и т.д. Для выхода в Интернет достаточно просто открыть ваш браузер. Технология Wi-Fi постоянно совершенствуется, что позволяет защитить ваши данные и подключение к Интернету, не опасаться взломов и посторонних проникновений. (Для защиты данных в Wi-Fi используется стандарт шифрования WPA2, который не был взломан до сих пор.) По беспроводной сети в квартире или офисе можно соединить напрямую два или несколько компьютеров без дополнительного оборудования, что позволяет одновременно пользоваться Интернетом, использовать совместное сетевое оборудование и т.д.

Владельцы сотовых телефонов и раньше могли использовать телефон как модем при подключении его к ноутбуку. Это было достаточно неудобно, так как требовало наличие компьютера и специального кабеля, качество связи и скорость передачи данных были низкими, а стоимость нахождения в сети — высокими. Поэтому производители мобильных телефонов приняли решение об использовании в качестве основной технологии «мобильный Интернет» протокола **WAP — протокола беспроводного доступа**. Этот протокол обеспечивает представление информации из сети Интернет на экран мобильного телефона.

Так как экран мобильного телефона имеет небольшое разрешение, то вся информация представляется только в текстовом виде, кстати, это снижает объем перекачиваемой информации. Средняя WAP-страница обычно занимает 200—300 байт.

Для передачи информации используется специальный гипертекстовый язык — WML, с помощью которого данные, отображаемые на дисплее, специальным образом форматируются и представляются в виде WAP-страницы. При настоящем уровне развития технологий

мобильного Интернета скоро каждый информационный сайт будет иметь два варианта — WEB и WAP.

Абонент мобильной связи при помощи WAP-браузера может получит самую широкую информацию и большой спектр различных услуг — купить билеты на самолет, оплатить счета в гостинице и т.п. Удобство использования этой технологии обусловлено тем, что так как местоположение абонента телефона легко отследить, то ему можно оперативно предлагать услуги в соответствии с тем, где он находится в настоящий момент.

Для того чтобы настроить телефон для работы с сетью, нужно найти в соответствующем разделе меню телефона параметры работы с WAP-протоколом и внести основные настройки: логин, пароль, номер телефона, IP-адрес шлюза, порт, скорость передачи данных, максимальное время неактивности.

Нам известно, что Интернет — это всемирная сеть компьютерных сетей. Используя Интернет, можно обмениваться цифровой информацией (наиболее известный пример — электронная почта). В настоящее время технически возможно оцифровать звук или факсимильное сообщение и переслать его аналогично тому, как пересылаются цифровые данные. Под **IP-телефонией** понимается технология, позволяющая использовать Интернет или любую другую сеть в качестве средства организации и ведения международных и междугородных телефонных разговоров и передачи факсов в режиме реального времени. IP-телефония является одним из наиболее сложных и системных приложений компьютерной телефонии.

ГЛАВА 9

ОСНОВЫ ЯЗЫКА ГИПЕРТЕКСТОВОЙ РАЗМЕТКИ HTML

9.1. Элементы разметки

HTML расшифровывается как «язык гипертекстовой разметки».

Гиперссылка — ссылка на другой документ.

Гипертекст — текст, содержащий гиперссылки.

Гипермедиа-документ — документ, включающий текст, рисунки, звуки, видео, любой элемент может быть гиперссылкой.

Web-страница — текстовый файл, документ, размеченный при помощи языка HTML, в котором описано размещение материала на экране. Web-страницы бывают:

- **статические** — существуют на сервере в виде готовых файлов с расширением **.htm** или **.html**;
- **динамические** — создаются на сервере в момент запроса полностью или частично (выбор информации из базы данных) и имеют расширение **.shtml, .asp, .pl, .php**.

Браузер — программа для просмотра Web-страниц на экране (*Internet Explorer*, *Mozilla Firefox*, *Opera*).

Документы, размеченные при помощи языка HTML, показываются браузерами пользователей одинаково. При помощи этого языка создаются Web-сайты. Сайт — это набор страниц в формате HTML, расположенные на Web-сервере, который передает эти страницы по запросу пользователей. Исходный код представляет собой текст, между строк которого вставляются элементы разметки, посетителю страницы эти элементы не видны, а виден результат их воздействия на документ.

Для примера посмотрим, как выглядит на языке гипертекстовой разметки следующий сайт (рис. 9.1).

Щелкнув правой клавишей мыши, получим меню, в котором выберем пункт: Просмотр в виде HTML.

ми — они организуют текст и *форматирующими* — т.е. задающими его стиль.

Тэг — оформленная единица HTML-кода. Теги могут указывать нашему браузеру, каким шрифтом отображать текст, вставить в страницу картинку, гиперссылку, указывать расположение текста и т.д. Теги заключены в треугольные скобки. Заглавные буквы в теге или строчные, значения не имеет. Вот пример тега:

```
<BODY>
</BODY>
```

Тэги бывают начальными (открывающими) и конечными (закрывающими, начинающимися со знака «/»). За каждым открывающим тегом обычно следует закрывающий. Например, тэгам

```
<HEAD>, <FONT…>, <BODY>, <HTML>
```

соответствуют закрывающие тэги

```
</HEAD>>, <</FONT>, </BODY>, </HTML>.
```

Дескриптор <I>, изменяющий начертание шрифта на наклонное, является тегом, требующим закрытия, так как если он не будет закрыт, то весь текст, следующий после него, будет наклонным.

Закрывающим элементом всегда служит тот же самый тег, но со знаком «/» после первой угловой скобки.

Например:

```
<I> Шрифт этого текста наклонный. </I>
```

При использовании вложенных тэгов необходимо руководствоваться следующим правилом — если один закрывающий тег вложен внутрь второго, первым необходимо закрывать вложенный тег. Закрывающие теги должны идти в порядке, обратном открывающим.

Например:

```
<I> Шрифт этого текста наклонный.
<U> Шрифт этого текста не только наклонный,
но и подчеркнутый. </U></I>
```

Теги, не требующие закрытия, обычно выполняют какое-либо действие не над другим объектом, а сами по себе. Примером может послужить тег
, вызывающий разрыв и переход на следующую строку.

Пример:

```
На странице этот текст <BR>
будет располагаться в две строки.
```

9.2. Структура HTML-документа

Каждый HTML-документ начинается со строки декларации версии HTML *<!DOCTYPE>*, которая обычно выглядит так: <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//RU"> С помощью этой строки браузер определяет правильную интерпретацию документа.

После объявления версии и типа документа необходимо обозначить его начало и конец. Это делается с помощью тэга-контейнера <HTML>. Любой HTML-документ открывается и закрывается тэгом <HTML>.

Между тэгами <HTML> и </HTML> размещается заголовок и тело документа.

HTML-документ состоит из двух основных блоков — «заголовка» и «тела документа».

В теле документа находится все то, что отображается на странице: текст, картинки, таблицы.

Элемент TITLE

Этот элемент определяет имя всего документа. Имя отображается в заголовке окна браузера. Данный элемент обязателен для любого HTML-документа и может быть указан не более одного раза. Заголовок страницы включает примерно следующее:

```
<html>
<head>
<title>Заголовок страницы</title>
<meta name="description" content="Краткое описание
страницы">
<meta name="keywords" content="Ключевые слова через
запятую">
</head>
.....
```

Название сайта записано между тегами <TITLE>. Краткое описание содержания сайта записывается с помощью метатега DESCRIPTION, если этого метатега нет, то в качестве описания обычно будет использованы первые строки самого сайта, что не всегда отображает его суть. Ключевые слова из метатега KEYWORDS имеют большее значение при поиске.

Кодирование цветов осуществляется с помощью имени: **red**, **green**, **blue**, **magenta**, **black** или шестнадцатеричного кода, например:

- **#FF0000** — красный;
- **#FFFFFF** — белый;
- **#000000** — черный;
- **#AAAAAA** — серый.

Тэг BODY — задает общие свойства страницы:

- цвет фона и текста

```
<BODY TEXT="white"
BGCOLOR=#00FF00>
```

- цвет гиперссылок

```
<BODY LINK="#FF8C00"
VLINK=green>
...
</BODY>
```

9.3. Создание и запуск HTML-документа

Создадим свой HTML-документ следующим образом.

1. В программе Блокнот набирается текст документа со всеми элементами разметки, например:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0
Transitional//EN">
<!-- saved from url=(0051) http://www.bytecity.
ru/~grinjen/HTML/1/lesson1.html -->
<HTML><HEAD>
```

При закрытии тегов необходимо руководствоваться следующим правилом — если один закрывающий тег вложен внутрь второго, первым необходимо закрывать вложенный тег. Запомните, что закрывающие теги должны идти в порядке, обратном открывающим.

```
</BODY></HTML>
```

2. Набранный файл сохраняется с расширением htm или html.

3. Сохраненный в формате HTML документ записывается в выбранную папку:

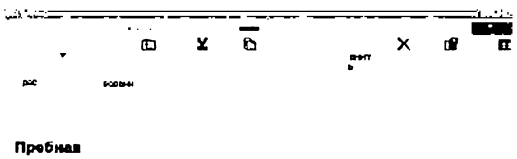


Рис. 9.3

4. Щелчкнув левой клавишей мыши по значку нашей HTML-страницы, мы получим ее на экране в следующем виде:

Рис. 9.4

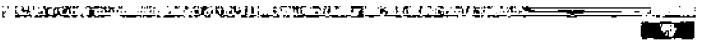
9.4. Ввод и оформление текста

Текст любой веб-страницы обычно начинается с заголовка. Всего существует шесть размеров шрифтов заголовков. Теги заголовков определяют вид текста на экране. Для выделения заголовков браузеры отображают их более крупным и жирным шрифтом в отличие от основного текста. Для того чтобы ввести заголовок с первым размером шрифта, воспользуемся тегами <H1> и </H1>. Например, оформим веб-страницу:

Для этого вставим нужные теги и текст заголовка в текст веб-страницы:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0
Transitional//EN">
<!-- saved from url=(0051)
http://www.bytecity.ru/~grinjen/HTML/1/lesson1.html -->
<HTML><HEAD>
<BODY>
<H1>Запомните правило:</H1>
```

При закрытии тегов необходимо руководствоваться следующим правилом — если один закрывающий тег вложен внутрь второго, пер-



Для разбивки текста на абзацы применяют тег нового абзаца <P>. Этот тэг разделяет фрагменты текста пустой строкой. Этот тег не парный. Ввести его очень просто: указатель мыши подвести к месту разбивки текста и ввести <P>.

Если вводить пустую строку не надо, то вводится тег перевода строки
. Этот тег тоже не парный и вводится в текст в том месте, где нужно перейти на новую строку.

Для организации текста можно применить и тег горизонтальной линии <HR>. Этот тег можно использовать в любом месте HTML-документа.

К тегу <HR> можно добавлять расширения SHADE и NOSHADE, которые позволяют задать горизонтальную контурную линию или черную.

Ширина горизонтальной линии задается с помощью ключевого слова WIDTH и значения ширины линии в процентах: <HR WIDTH=50%> — эта линия займет половину экрана по центру. Выравнивание линии можно задать с помощью слова ALIGN. Всего известно три типа выравнивания — по левому краю, по правому краю и по умолчанию по центру. Например, горизонтальная линия в половину экрана с выравниванием по левому краю задается так:

```
<HR WIDTH=50% ALIGN=LEFT>
```

Для увеличения толщины линии (по умолчанию она равна 2 пикселам) применяют ключевое слово SIZE, а для того, чтобы толстые линии были темного цвета, с этим ключевым словом используют слово NOSHADE.

Например:

```
<HR NOSHADE SIZE=50>.
```

Управление внешним видом текста на веб-странице

Шрифт текста на веб-странице, как и в текстовом редакторе, может быть отображен на экране разного размера, вида и цвета. Для этого служат теги форматирования, которые ставятся в начале и в конце нужного текста:

<CENTER> и </CENTER> — выравнивает текст по центру;
<B> и </B> — выделение текста полужирным шрифтом;
<I> и </I> — выделение шрифта курсивом;
<BLINK> и </BLINK> — выделение шрифта мерцанием;
<S> и </S> — зачеркивание текста;
<U> и </U> — подчеркивание текста;
<BIG> и </BIG> — увеличивает размер шрифта;

<SMALL> и </SMALL> — уменьшает размер шрифта;
_и — вывод текста в виде нижнего индекса;
^и — вывод текста в виде верхнего индекса.

Дополнительные возможности для изменения вида текста дает тег

```
<FONT> и </FONT>:
```

Задание фактического размера текста производится с использованием ключевого слова SIZE=, например, текст с размером шрифта 15 будет выведен, если его выделить тегами

```
<FONT SIZE=15>
```

и

```
</FONT>.
```

Размер шрифта можно задать относительно основному, используя не фиксированное число, как, например 10 или 15, а + 4 или –2.

Изменение начертание текста можно задать ключевым словом FACE=, после него следует указать название шрифта.

Использование ключевого слова COLOR= дает возможность менять цвет текста. После ключевого слова следует ввести название цвета.

Изменить цвет текста на экране можно и с помощью ключевого слова TEXT= в теге <BODY>.

Создание списков и таблиц

Существует несколько типов списков, из них наиболее распространены маркированные и нумерованные списки.

Маркированный список задается тегами <UL> и </UL>. Эти теги ставятся в начало и конец списка. Каждый пункт списка задается тегом <LI>.

Нумерованный список задается тегами <OL> и </OL>, как и в маркированном, каждый пункт этого списка задается тегом <LI>.

Часто данные удобней располагать не в списках, а в таблицах.

Теги	Описание
{PRIVATE}TABLE	Сообщает браузерам, что далее следует описание таблицы. Если нужно, чтобы отображалась сетка таблицы, следует задать ключевое слово BORDER
CAPTION	Текст, отмеченный этим тегом, выводится в виде заголовка
TR	Определяет каждую строку таблицы
TD	Определяет текст каждой ячейки таблицы
TH	Выделяет текст заголовка строки или столбца более жирным шрифтом

Например, при вводе:

```
<TABLE BORDER>
  <TR>
    <TH ROWSPAN=2>КЛАСС</TH>
    <TD>11-А</TD><TD ALIGN="right">28</TD>
  </TR>
  <TR>
    <TD>11-Б</TD><TD ALIGN="right">28</TD>
  </TR>
</TABLE>
```

Получим следующую таблицу:

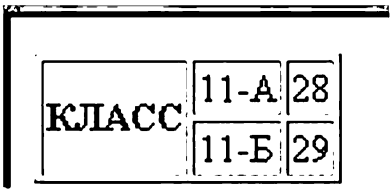


Рис. 9.7

А при вводе:

```
<TABLE BORDER>
  <TR>
    <TH COLSPAN=2>Отличники</TH>
  </TR>
  <TR>
    <TD>11-А</TD><TD ALIGN="right">10</TD>
  </TR>
  <TR>
    <TD>11-Б</TD><TD ALIGN="right">12</TD>
  </TR>
</TABLE>
```

Получим таблицу:

Рис. 9.8

9.5. Графическое оформление веб-страницы

Изображение для своей веб-страницы можно скопировать с любой другой страницы, нарисовать самому с помощью графического редактора или отсканировать понравившийся рисунок. Изображение нужно сохранить в формате GIF и JPEG, лучше в той же директории, в которой записан файл HTML-документа. Формат JPEG существенно эффективней формата GIF, особенно для больших изображений, кроме этого, он лучше сохраняет цвета и детали.

Для того чтобы разместить рисунок на веб-странице, нужно применить тег <IMG> и ключевое слово SRC=, после которого указывается, какая графика должна быть включена в страницу. Например, для того чтобы вставить рисунок logo3.gif:

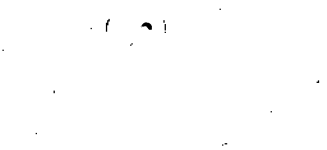


Рис. 9.9

в текст веб-страницы, добавим тег, ключевое слово и имя рисунка:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0
Transitional//EN">
<!-- saved from url=(0051)
http://www.bytecity.ru/~grinjen/HTML/1/lesson1.html -->
<HTML><HEAD>
<BODY>
  <H1>Запомните правило:</H1>
  <img src = "logo3.gif">
  <HR>
```

При закрытии тегов необходимо руководствоваться следующим правилом — если один закрывающий тег вложен внутрь второго,
 первым необходимо закрывать вложенный тег. <P> Запомните, что закрывающие теги должны идти в порядке, обратном открывающим.

```
<HR SHADE>
<HR NOSHADE>
</BODY></HTML>
```

Теперь наша страница станет такой:

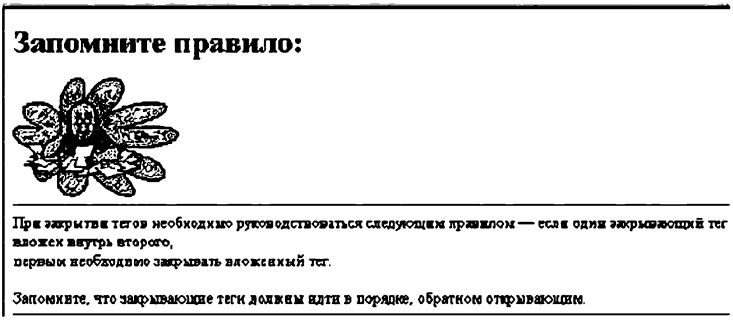
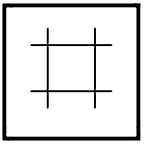


Рис. 9.10

Изменение фона веб-страницы

Для изменения фона страницы применяется ключевое слово BGCOLOR=, которое является частью тега <BODY>, с его помощью можно изменить цвет страницы, используя один из 16 стандартных цветов. Если после ключевого слова BACKGROUND= указать название рисунка, то этот рисунок будет использован в качестве фона страницы. Например, используя имеющийся рисунок клетки, записанный в файле Kl.gif:



можно получить клетчатый фон страницы:

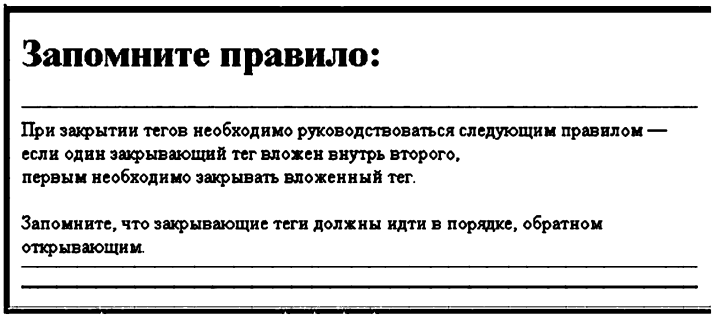


Рис. 9.11

В этом случае текст документа выглядит так:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0
Transitional//EN">
<!-- saved from url=(0051)
http://www.bytecity.ru/~grinjen/HTML/1/lesson1.html -->
<HTML><HEAD>
  <BODY background=Kl.gif>
  <H1>Запомните правило:</H1>
  <HR>
```

При закрытии тегов необходимо руководствоваться следующим правилом — если один закрывающий тег вложен внутрь второго,
, первым необходимо закрывать вложенный тег <P>. Запомните, что закрывающие теги должны идти в порядке, обратном открывающим.

```
<HR SHADE>
<HR NOSHADE>
</BODY></>
```

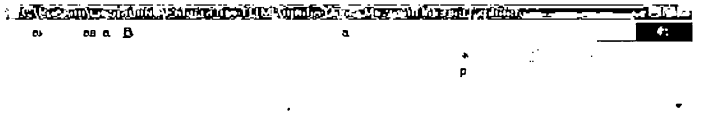
Любой документ может содержать ссылки на другой HTML-документ. Это и является основным свойством WWW. Ссылаться на другие веб-страницы достаточно просто — нужно воспользоваться тегами <A> и </A>. Между ними располагаются ключевое слово гиперссылки HREF=, адрес ссылки и текст. Например, дополним предыдущий текст гиперссылкой:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0
Transitional//EN">
<!-- saved from url=(0051)
http://www.bytecity.ru/~grinjen/HTML/1/lesson1.html -->
<HTML><HEAD>
<BODY>
  <H1>Запомните правило:</H1>
  <img src="logo3.gif">
  <HR>
```

При закрытии тегов необходимо руководствоваться следующим правилом — если один закрывающий тег вложен внутрь второго,
, первым необходимо закрывать вложенный тег <P>. Запомните, что закрывающие теги должны идти в порядке, обратном открывающим.

```
<HR SHADE>
<HR NOSHADE>
<A HREF=проба2/htm>Пример</A>
</BODY></HTML>
```

Теперь наша страница выглядит так:



Запомните правило:

Рис. 9.12

В качестве ссылки можно использовать и изображение, если вставить тег <IMG> в гипертекстовую ссылку, например:

```
<A HREF=ПРОБА2/HTM><IMG SRC="LOGO3.GIF"></A>
```

Фрэймы

Для облегчения навигации по сайту, создания навигационного меню, разбивки окна браузера на несколько областей, каждая из которых представляет собой отдельный HTML-документ, используются фрэймы.

Элементы для создания фрэймов и работы с ними.

FRAMESET — определяет фрэймовую (оконную) структуру документа: размеры и расположение фрэймов на странице. Открывает и закрывает список фрэймов, определяемых с помощью элемента FRAME. Элемент FRAMESET поддерживает вложенные конструкции фрэймов.

Количество и размеры горизонтальных фрэймов (фрэймов-строк) в окне браузера определяет ключевое слово ROWS. Список размеров фрэймов задается в качестве значения через запятую одним из следующих способов:

- в процентах от высоты рабочей области окна браузера. Например: «10%, 40%, 50%»;

- знак «*» указывает, что фрэйм занимает все свободное пространство окна браузера, незанятое другими фрэймами. Например, запись «30%, 30%, *» говорит о том, что окно разбито на три фрейма, причем последний занимает 40% пространства окна;
- в пикселах. Например: «60, *».

Все три способа задания размеров можно совмещать. Запись «30%, 50, *» разобьет экран на три горизонтальных фрэйма, первый из которых будет высотой в треть окна браузера, второй — в 50 пикселов, а третий — займет всю оставшуюся площадь.

Количество и размеры вертикальных фрэймов (фрэймов-столбцов) в окне браузера определяет параметр COLS. Список размеров фрэймов задается так же, как и в ключевом слове ROWS.

Ключевым словом FRAMEBORDER определяется наличие рамок у содержащихся внутри FRAMESET фрэймов. Оно может иметь следующие значения:

Yes — отображать рамки;

No или 0 — не отображать рамки.

Если фрэймы создаются без рамок, тогда необходимо ключевое слово FRAMESPACING. Оно определяет расстояние (так называемую серую область) между фрэймами в пикселах.

Тег FRAME определяет фрэйм и его свойства внутри FRAMESET-структуры, вместе с ним применяются следующие ключевые слова:

SRC — обязательный параметр. Указывает адрес HTML-файла, отображаемого в данном фрэйме;

NAME — определяет имя данного фрэйма, в качестве значения нужно указать любое имя без пробелов с использованием латинских символов и цифр. Имя не должно начинаться с цифр и специальных символов;

MARGINWIDTH — определяет ширину (в пикселах) левого и правого полей фрэйма. Если параметр не указан, браузер самостоятельно определит оптимальный размер отступа;

MARGINHEIGHT — определяет ширину (в пикселах) верхнего и нижнего полей фрэйма. Если параметр не указан, браузер самостоятельно определит оптимальный размер отступа;

SCROLLING — определяет наличие линеек прокрутки содержимого фрэйма. Возможные значения:

yes — отображать линейки прокрутки,

no — не отображать линейки прокрутки,

auto — отображать линейки прокрутки при необходимости (если документ, указанный в параметре SRC, не умещается во фрэйме).

```
<FRAMESET FRAMEBORDER="0" FRAMESPACING="0" BORDER="0"
COLS="265,*">
<FRAME SRC="проба1.htm" NAME="page">
<FRAMESET ROWS="165,*">
<FRAME SRC="проба2.htm" NAME="menu1" MARGINWIDTH="0">
<FRAME SRC="проба3.htm" NAME="menu2" MARGINWIDTH="0">
</FRAMESET>
  <BODY>
  </BODY>
```

В результате окно браузера разделится на три фрэйма, как показано ниже. Причем проба1.htm будет иметь ширину 265 пикселей, а проба2.htm — высоту 165.

9.6. Создание сайта с помощью редактора

В настоящее время существует достаточно много различных редакторов Web-страниц, которые не требуют досконального знаний основ HTML.

Мы воспользуемся редактором, который позволит создать сайт в том виде, в котором он должен быть отображен в окне браузера — это Microsoft Word, позволяющий создавать простые веб-сайты. Создадим в Word простую домашнюю страницу.

Запустим редактор Microsoft Word. Выберем команду Создать из меню Файл, после чего выберем создание новой веб-страницы:

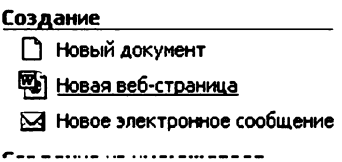


Рис. 9.13

Введем желаемый текст и сохраним документ на диске. Лучше создать для веб-сайта отдельную папку. При сохранении обратите внимание, что расширение у полученного файла htm, типичное для веб-страниц.

Затем создадим документы, на которые можно будет перейти с помощью гиперссылок. Создать гиперссылку на документ можно следующим образом:

- набрать текст гиперссылки;
- выделить ее;

- выбрать меню «Вставка-Гиперссылка» (или выбрать «Гиперссылка» из контекстного меню выделенного текста, нажав на правую кнопку мыши);
- выбрать нужный документ (в созданной нами папке).

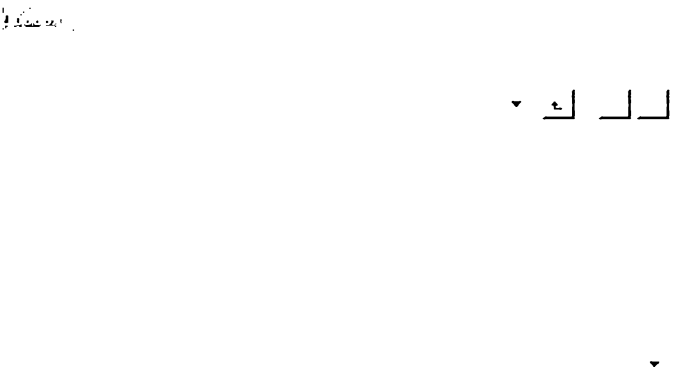


Рис. 9.14

Конечно, текст гиперссылки должен нести информацию о том, на какую страницу будет осуществлен переход.

Можно воспользоваться еще одной возможностью, которую нам предоставляет редактор Microsoft Word — это создание сайта с помощью мастера Web-страниц:

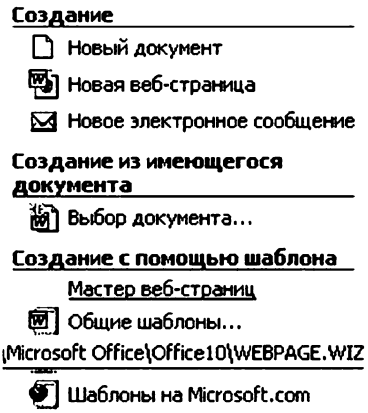
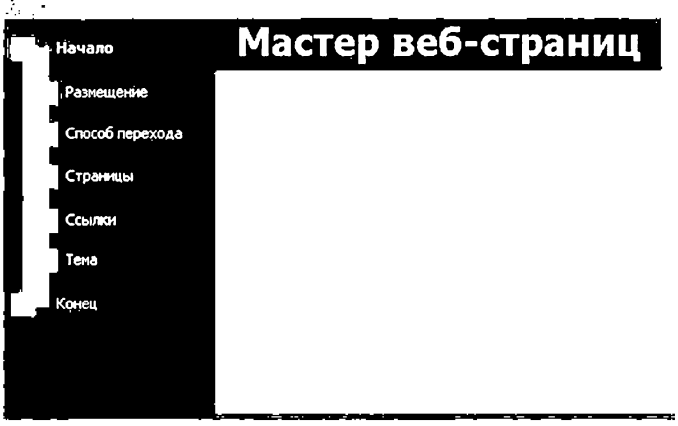
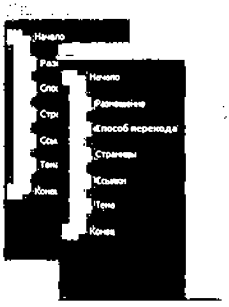


Рис. 9.15


Мастер веб-страниц
Начало
Размещение
Способ перехода
Страницы
Ссылки
Тема
Конец



9.7. Размещение страницы в Интернете

Существуют серверы, которые предоставляют бесплатные услуги на размещение вашего сайта. Следует помнить, что понятие «бесплатно» — несколько условно, и за него придется все-таки платить, правда не деньгами, а появлением на вашем сайте какого-то рекламного банера. Это не всегда бывает удобно — допустим, вы создали сайт своей фирмы вегетарианской пищи — а на сайте появляется банер рекламы мясокомбината. Конечно, такой сайт не лучшим образом повлияет на имидж компании. Поэтому вы всегда сможете обратиться на платный сервер.

Наиболее известны следующие бесплатные хостинги:

- chat.ru одним из первых начал предоставлять бесплатный хостинг;
- boom.ru выделяет 50 Мгб, разместить свой сайт необходимо в течение семи дней;
- narod.ru позволяет создавать файлы при помощи встроенного редактора, пользоваться готовыми гостевой книгой и форумом.

После размещения страницы на сервере обязательно проверьте гиперссылки — ошибка в тексте ссылки делает ее неработоспособной.

ГЛАВА 10

НАЧАЛА АЛГОРИТМИЗАЦИИ

Сущность задачи программирования — записать что-то на чужом, малознакомом языке (мы здесь не ведем речи о профессионалах-программистах, для которых язык программирования ближе, чем родной). Задача, прямо скажем, не новая! Человечество тысячи лет ее решает весьма успешно.

Но программистам удалось сказать здесь новое слово и свести задачу программирования к такой проблеме: «как записать что-то по-французски, если не знаешь, как записать это по-русски (на родном языке)».

Да, именно так — большинство начинающих программистов, составляя программу, сразу начинают писать ее на Бейсике, Паскале и т.д., не имея понятия, как она выглядит на родном, человеческом языке.

Это напоминает ситуацию, когда человек, плохо владеющий иностранным языком, сразу пытается написать статью (рассказ, повесть и т.д.) на этом языке. На наш взгляд, значительно проще написать то же произведение сначала на родном языке, «отшлифовать» его текст, устранить ошибки и затем перевести на иностранный язык. При этом лишь 20—25% времени будет затрачено на работу с иностранным текстом, а не 100%, как при первом подходе.

То же самое и при решении задачи на ЭВМ. Сначала следует описать процесс решения на родном для программиста языке, например в виде схемы алгоритма, «отшлифовать» его и только после этого перевести на язык ЭВМ — Бейсик, Паскаль и др. При таком подходе собственно написание программы займет не более 15—20% времени, потраченного на ее разработку.

Именно такой подход принят в пособии. В соответствии с ним сначала мы учимся составлять только алгоритмы (этому посвящены первые две главы) и лишь после этого переводить их на язык Бейсик.

10.1. Основные понятия

10.1.1. Алгоритмы и ЭВМ

Понятие алгоритма. Цель пособия — научить решать задачи на ЭВМ, т.е. использовать для решения задач ЭВМ, которая эти задачи решать не умеет.

Рассмотрим предварительно простую жизненную ситуацию: что следует сделать, если нужно привлечь к решению задачи человека, не знакомого с ее решением?

Такая ситуация возникает всякий раз, когда вы, например, хотите, чтобы ваш младший брат вместо вас выполнил какую-то неприятную работу вроде мытья полов, но которую он, к сожалению, не умеет делать! Очевидно, в таком случае надо его научить! Каким образом? Так же, как поступают (или следует поступать) и в тысяче подобных случаев:

а) выбирают способ (метод, порядок) решения задачи и изучают его во всех подробностях;
б) сообщают исполнителю выбранный метод в абсолютно понятном для него виде;
в) исполнитель решает «задачу строго в соответствии с методом».

Углубляясь в суть этого процесса, рассмотрим пристальнее каждый из этапов.

Первый этап этого процесса обычно не вызывает затруднений, так как для большинства встречающихся задач метод решения либо известен из практики, либо подсказывается здравым смыслом, либо описан в литературе[1]. Часто главная трудность — из нескольких методов выбрать такой, который в наибольшей степени отвечал бы некоторым требованиям, например минимальная трудоемкость, максимальная эффективность и т.д.

Второй этап значительно сложнее. Дело в том, что, если способ (метод) решения задачи описан произвольно, нет гарантии, что он будет верно понят исполнителем. Поэтому описание метода следует выполнять в соответствии с определенными правилами, а именно:

- выделить величины, являющиеся исходными для задачи;
- разбить процесс решения задачи на такие этапы, которые известны исполнителю и которые он может выполнить однозначно без всяких пояснений;
- указать порядок выполнения этапов;
- указать признак окончания процесса решения задачи;
- указать во всех случаях, что́ является результатом решения задачи.

Описание метода, выполненное в соответствии с этими правилами, называется *алгоритмом решения задачи*. Составить такое описание обычно нелегко, но, следуя ему, механически выполняя все указанные в нем этапы в требуемом порядке, исполнитель может всегда правильно решить задачу.

[1] Рассмотрение задач, метод решения которых отсутствует, относится к научной работе, чего не будем касаться.

Итак, мы подошли к центральному понятию информатики — «алгоритму». Оно, как говорится, красной нитью (а точнее красным канатом) проходит через весь курс. Более строго это понятие можно определить так: *алгоритм — это метод (способ) решения задачи, записанный по определенным правилам, обеспечивающим однозначность его понимания и механического исполнения при всех значениях исходных данных (из некоторого множества значений)*.

В Толковом словаре по информатике (1991) дано общепринятое определение этого понятия: «Алгоритм — точное предписание, определяющее вычислительный процесс, ведущий от варьируемых начальных данных к искомому результату».

Примером алгоритма может служить кулинарный рецепт приготовления блюда.

Рассмотрим простейший алгоритм — алгоритм заварки чая.

1. Подготовить исходные величины — чай, воду, чайник, стакан, ложку.
2. Налить в чайник воду.
3. Довести воду до кипения и снять с огня.
4. Всыпать в чайник чай.
5. Довести воду до кипения (но не кипятить), снять с огня.
6. Чай готов. Процесс прекратить.

Для углубления понятия алгоритма выделим и раскроем его основные свойства, вытекающие из его определения.

1. *Дискретность алгоритма.* Свойство алгоритма, означающее, что процесс решения задачи, определяемый алгоритмом, расчленен на отдельные элементарные действия (шаги) и соответственно алгоритм представляет последовательность указаний, команд, определяющих порядок выполнения шагов процесса.

2. *Определенность алгоритма.* Это свойство означает, что каждая команда алгоритма (предписание, выдаваемое на каждом шаге) должна быть понятна исполнителю, не оставлять места для ее неоднозначного толкования и неопределенного исполнения. Описание алгоритма должно быть таким, чтобы его мог выполнить любой грамотный пользователь.

3. *Результативность алгоритма.* Свойство алгоритма, состоящее в том, что он всегда приводит к результату через конечное, возможно очень большое число шагов.

4. *Массовость алгоритма.* Это свойство заключается в том, что каждый алгоритм, разработанный для решения некоторой задачи, должен быть применим для решения задач этого типа при всех допустимых значениях исходных данных.

Вернемся к первоначальной цели — решению задач на ЭВМ. Оказывается, процесс решения задачи с помощью ЭВМ в целом мало чем отличается от только что рассмотренного процесса решения этой же задачи человеком-исполнителем. Так же следует выбирать и изучать метод решения задачи, так же составлять алгоритм и решать задачу строго в соответствии с ним, только решать должна ЭВМ, а не человек.

Возможность использования ЭВМ вместо человека объясняется соответствием свойств алгоритма и ЭВМ: алгоритм допускает механическое выполнение его для решения задачи, а ЭВМ может механически, не вникая, выполнять операции в заданном порядке. Отличие указанного процесса решения задачи при использовании ЭВМ в том, что, составляя алгоритм, мы разбиваем процесс решения задачи на такие операции, которые в состоянии выполнить ЭВМ.

Другое отличие в том, что составленный алгоритм решения задачи следует перевести на язык, понятный ЭВМ, аналогично тому, как алгоритм, записанный на русском языке, нужно перевести на французский, если исполнителем является француз.

Существует значительное число подобных языков — Бейсик, Фортран, Паскаль и др. Они называются *языками программирования*. Запись алгоритма на таком языке называется *программой*, а процесс перевода алгоритма на указанный язык — *программированием*.

Выводы. 1. Процесс решения задач на ЭВМ предполагает выполнение следующих основных этапов: формулировка задачи, выбор метода решения задачи, составление алгоритма, составление программы, решение задачи на ЭВМ по заданной программе.

Исходя из этого, основные понятия указанного процесса следующие: «метод» — «алгоритм» — «программа».

2. Рассмотренная трактовка понятия «алгоритм» показывает, что алгоритм — это не что-то отвлеченное, абстрактное и присущее лишь процессу использования ЭВМ, а неотъемлемая часть повседневной жизни. В частности, любые инструкции, любые распоряжения руководства должны быть сформулированы в виде алгоритма, чтобы они были однозначно поняты подчиненными. Причем правила формирования распоряжений-алгоритмов, инструкций-алгоритмов те же, что и для алгоритмов решения задач на ЭВМ.

В этом отношении предмет «Основы информатики» занимает особое положение в ряду прочих предметов, так как он учит составлять алгоритмы, что очень важно в жизни, а не только применительно к ЭВМ.

Возможности ЭВМ. Как отмечалось, при составлении алгоритма процесс решения задачи разбивают на этапы, ориентируясь на исполнителя, на операции, известные ему.

Так, в примере алгоритма заварки чая предполагалось, что исполнитель знаком с операцией «вскипятить воду». Если это не так, то п. 3 алгоритма следует разбить на три более мелких этапа, считая, что они ему известны:

```
3а. Зажечь газ.
3б. Поставить чайник на огонь.
3в. Дождаться бурного выделения пара и снять чайник
с огня.
```

Аналогично, если нужно составить программу для ЭВМ, то в алгоритме должны предусматриваться те операции, которые ЭВМ способна выполнять.

Что же может выполнять ЭВМ? Прежде всего рассмотрим, с какими величинами может она работать. Эти величины подразделяются на *числовые* и *текстовые*, с одной стороны, и на *постоянные* и *переменные* — с другой.

Постоянные числовые величины изображаются числами, например 31,4; 315; 10^8 и т.д. Эти величины не меняют своих значений. *Переменные числовые величины* могут в процессе решения задач принимать разные значения. В записи алгоритма они обозначаются буквами, как в математике, или словами: *X*, *Y*, *MAX*, РЕЗУЛЬТАТ. В ЭВМ для любой величины выделяется ячейка памяти. Если это числовая величина, то в ней хранится число, изображающее значение этой величины.

Текстовые величины могут быть переменными и постоянными. *Текстовые константы* — «ХОРОШО», «ВАСЯ», «$Y = 2X(A + B)$» и т.д., т.е. это любой текст в кавычках. Такие тексты обычно используются для пояснения результатов вычислений, выдаваемых ЭВМ. *Текстовые переменные*, как и числовые, обозначаются буквами или словами. В этом случае в ячейку памяти, выделенную для переменной, помещается не число, а некоторый текст. Например, если ДРУГ — текстовая переменная, то в ячейку, выделенную для нее, в один момент времени можно записать текст «ВАСЯ», в другой момент — «ТАНЯ» и т.д. Эти тексты и будут значениями переменной ДРУГ в разные моменты времени.

Теперь рассмотрим, какие операции может выполнять ЭВМ над этими величинами и как они записываются в алгоритме.

1. ЭВМ может считывать конкретные значения исходных величин с различных устройств ввода, например с клавиатуры, и помещать каждое из них в ячейку, выделенную для соответствующей переменной. В результате каждая исходная величина получает то или иное значение. С помощью такой операции, называемой «Ввод», обеспечи-

вается возможность решения задачи с разными значениями исходных данных. Записывается она, например, так:

Ввод X, Y, A.

2. ЭВМ может вычислять значения величины по заданной формуле, содержащей знаки любых арифметических операций, ряда элементарных функций типа $\ln x$, $|x|$, $\sin x$ и т.д., разные для разных языков программирования. Пример записи такой операции:

$$Y:=31\ln X+B^2.$$

Подобная операция называется «операция присваивания» и в общем виде записывается так:

x:=a,

где x — переменная; a — арифметическое выражение или текст. ЭВМ воспринимает эту запись как приказ — «вычислить значение выражения a и присвоить это значение переменной x», т.е. переслать его в ячейку памяти, отведенную для x. В частности, операция позволяет присваивать переменной конкретное значение.

Например:

```
Z:=5,1; R:="КОЛЯ".
```

Эти записи означают, что в ячейку памяти, выделенную для Z, ЭВМ должна записать число 5,1, а в ячейку для R — указанные четыре буквы.

3. ЭВМ может печатать на бумаге или выводить на экран монитора значения величин или любой текст. Эта операция называется «Вывод» или «Печать» и записывается, например, так:

```
Вывод X, Y, Z, "конец вычислений".
```

Данная операция, в частности, позволяет выводить из ЭВМ результаты решения задачи и пояснения к ним.

4. ЭВМ может переходить от одного этапа решения задачи к любому другому. Операция называется «Переход». В ней указывается номер этапа, к которому нужно перейти. Например:

```
Перейти к п. 6.
```

5. ЭВМ может сравнивать значения двух арифметических выражений (или двух текстовых величин) на предмет проверки условий: $<$, $>$, $=$ и т.д. и в зависимости от результатов проверки выбирать один из двух возможных вариантов дальнейших действий. Записывается это, например, так:

```
Если X>Y, то Y:=X^2,
иначе Y:=X^3.
```

Подобную операцию называют *Условный переход* (*Ветвление*).

В общем, это все, что умеет выполнять ЭВМ[1]. Поэтому сложность составления алгоритмов заключается в том, что процесс решения любой задачи нужно ухитриться представить в виде последовательности только таких операций.

10.1.2. Способы описания алгоритмов

В настоящее время используется несколько таких способов.

1. **Словесно-формульное описание алгоритма,** т.е. описание алгоритма с помощью слов и формул. Это наиболее простой способ. Для его понимания достаточно рассмотреть пример, приведенный ниже. Кстати, кулинарный рецепт — пример такого описания алгоритма.

Задача 10.1

Составить алгоритм начисления зарплаты согласно следующему правилу:

если стаж работы сотрудника менее 5 лет, то зарплата 130 тыс. руб., при стаже работы от 5 до 15 лет — 180 тыс. руб., при стаже свыше 15 лет зарплата повышается с каждым годом на 10 тыс. руб.

Сформулируем задачу в математическом виде: вычислить

$$ZP = \begin{cases} 130, & \text{если } ST < 5; \\ 180, & \text{если } 5 < ST < 15; \\ 180 + (ST - 15)10, & \text{если } 15 < ST, \end{cases}$$

где ZP — зарплата; ST — стаж работы.

Словесно-формульное описание алгоритма решения задачи 10.1:

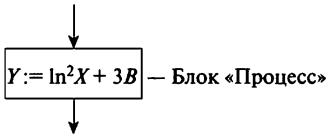
1. Ввести ST, перейти к п. 2.
2. Если $ST < 5$, то ZP:=130, перейти к п. 4, иначе — перейти к п. 3.
3. Если $ST \le 15$, то ZP:=180, перейти к п. 4, иначе ZP:=180 + (ST – – 15) 10, перейти к п. 4.
4. Вывести (отпечатать) значение ZP, перейти к п. 5.
5. Вычисления прекратить.

Алгоритм, очевидно, не нуждается в пояснении, поскольку форма записи его очень естественна. Рекомендуем читателю проследить, как реализованы в этом примере все правила записи алгоритма, приведенные в 10.1.1.

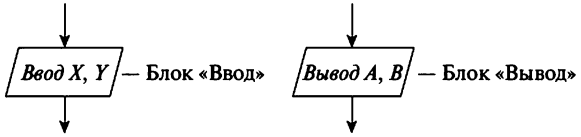
[1] В зависимости от вида используемого языка программирования ЭВМ может выполнять и иные операции, но приведенные здесь — основные, присущие большинству современных языков.

2. **Графическое описание алгоритма**, т.е. описание с помощью схем алгоритмов. *Схема алгоритма* представляет собой систему связанных геометрических фигур. Каждая фигура обозначает один этап процесса решения задачи и называется *блоком*. Порядок выполнения этапов указывается *стрелками*, соединяющими блоки. В схеме блоки стараются размещать сверху вниз, в порядке их выполнения. Для наглядности операции разного вида изображаются в схеме различными геометрическими фигурами.

Операция присваивания изображается прямоугольником, например:

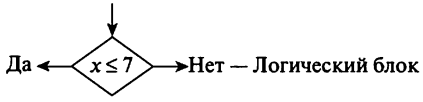


Операции *Ввод* и *Вывод* изображаются параллелограммом, например:



Каждый из трех указанных блоков имеет один вход и один выход.

Операция *Условный переход* изображается ромбом; блок имеет два выхода — *Да* и *Нет*, например:

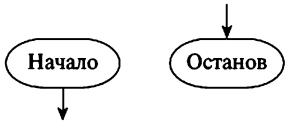


Если условие выполняется — выходим из блока по выходу *Да*, если не выполняется — по выходу *Нет*.

Начало процесса решения задачи обозначается блоком *Начало*.

Завершение процесса решения задачи обозначается блоком *Останов*.

Последние два блока изображаются так:



Пример: схема алгоритма решения задачи 10.1 (рис. 10.1):

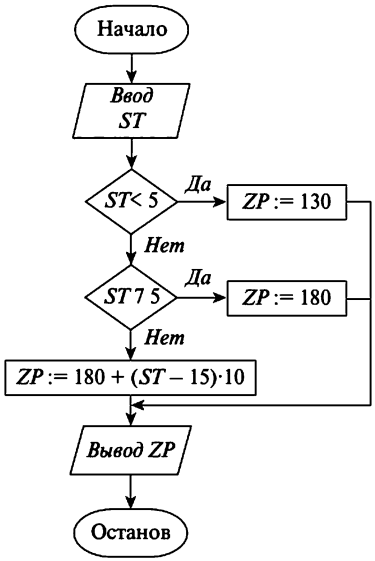


Рис. 10.1

3. **Описание алгоритма на алгоритмическом языке (алгоязыке).** *Алгоритмический язык* — это средство для записи алгоритмов в аналитическом виде, промежуточном между записью алгоритма на естественном (человеческом) языке и записью на языке ЭВМ (языке программирования).

Пример. Запись алгоритма решения задачи 10.1 на алгоритмическом языке:

```
алг зарплата (цел ST, вещ ZP)
  арг ST
  рез ZP
  нач
    если ST < 5
      то ZP:=150
      иначе
        если ST <= 15
          то ZP:=180
          иначе ZP:=180 + (ST - 15)10
        все
    все
  кон
```

Из примера видно, что запись алгоритма на алгоязыке весьма близка к его словесно-формульному описанию. Разница между ними со-

стоит в том, что в алгоязыке используется ограниченный набор терминов, более строгие правила записи операции и т.д. с целью обеспечения однозначности понимания алгоритма.

Наглядность такого способа описания алгоритмов невысока, особенно алгоритмов сложной структуры. Однако алгоязык может быть полезен как средство компактной записи алгоритмов.

Вывод. Сравнение разных способов описания алгоритмов на конкретном примере должно подтвердить мысль о том, что *наиболее наглядный способ — схемы алгоритмов*. Это и наиболее естественный способ, так как человек мыслит образами и, ожидая от него эффективной работы, следует предоставлять ему возможность работать с образами, в нашем случае — с образами алгоритмов, т.е. со схемами алгоритмов.

В соответствии с этим далее для изображения алгоритмов в основном используются схемы. Переход от схемы к любому другому способу описания алгоритма и даже к программе несложен.

10.1.3. Виды алгоритмов и основные принципы составления алгоритмов

Человеку в жизни и практической деятельности приходится решать множество различных задач. Решение каждой из них описывается своим алгоритмом, и разнообразие этих алгоритмов очень велико. Тем не менее можно выделить лишь три основных вида алгоритмов: *линейной* структуры, *разветвляющейся* структуры, *циклической* структуры (для краткости далее будем называть их просто: линейные, разветвляющиеся и циклические алгоритмы).

Разнообразие же алгоритмов определяется тем, что любой алгоритм распадается на части, фрагменты и каждый фрагмент представляет собой алгоритм одного из трех указанных видов. Поэтому важно знать структуру каждого такого вида алгоритмов и принципы его составления.

Далее эти виды алгоритмов будут рассмотрены подробно, сейчас лишь отметим, что алгоритм заварки чая, приведенный выше, является по своей структуре линейным, алгоритм задачи 10.1 — разветвляющимся. Примером циклического алгоритма может служить алгоритм начисления зарплаты согласно правилу задачи 10.1, но не для одного сотрудника, а для группы, например из 20 человек, т.е. алгоритм, в котором многократно выполняются одни и те же операции.

Цель настоящей главы — научить составлять алгоритмы указанных видов. Задача эта очень непростая из-за громадного разнообразия алгоритмов каждого вида, образно говоря «моря алгоритмов». Однако

существует «компас», помогающий достигать заданной цели в этом «море», — это «Основные принципы алгоритмизации», непосредственно вытекающие из приведенного определения алгоритма.

Рассмотрим суть упомянутых принципов на конкретной задаче.

Задача 10.2

Даны длины двух катетов прямоугольного треугольника. Определить периметр этого треугольника.

Требуется составить алгоритм решения задачи.

Решение. 1. Выделяем исходные данные и результаты (это первое, что должен содержать алгоритм согласно правилам его записи).

Исходные данные: A, B — длины катетов.

Результат: P — периметр треугольника.

2. Согласно определению, «алгоритм — это *метод* решения задачи...». Следовательно, далее нужно выбрать метод решения задачи — и это самое главное! («Нельзя сварить бульон из курицы, не имея курицы».) А зная метод, надо изложить его в соответствии с правилами записи алгоритмов, т.е. сначала разбить его на этапы.

В нашем случае метод решения задачи на ЭВМ такой:

1. Ввод данных.
2. Вычисление $Y = \sqrt{A^2 + B^2}$.
3. Вычисление $P = Y + A + B$.
4. Вывод P.

Решение задачи, как видим, распадается на четыре этапа.

Далее изображаем каждый из указанных этапов в виде определенной геометрической фигуры (блока) и соединяем их стрелками. В результате получаем схему алгоритма задачи (рис. 10.2).

Можно сформулировать общие правила построения схемы алгоритма задачи.

1. Выявить исходные данные, результаты, назначить им имена.
2. Выбрать метод (порядок) решения задачи.
3. Разбить метод решения задачи на этапы (с учетом возможностей ЭВМ).
4. Изобразить каждый этап в виде соответствующего блока-схемы алгоритма и указать стрелками порядок их выполнения.
5. В полученной схеме при любом варианте вычислений:

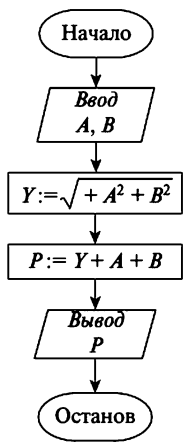


Рис. 10.2

а) предусмотреть выдачу результатов или сообщений об их отсутствии;
б) обеспечить возможность после выполнения любой операции так или иначе перейти к блоку Останов (*к выходу схемы*).

Эти правила и есть «Основные принципы алгоритмизации». Так их и будем называть. Мы считаем, что знание и применение настоящих «принципов» обязательно при составлении алгоритма любой задачи.

Примечание. В последующих задачах в отличие от задачи 10.2, разбивая метод решения на этапы, не будем упоминать об этапах ввода данных и вывода результатов, поскольку они являются обязательными в процессе решения любой задачи на ЭВМ и в большинстве случаев начинают и заканчивают этот процесс. Исключение будем делать лишь в случае нестандартного использования этих операций.

10.1.4. Исполнение алгоритмов

Как отмечалось, каждый алгоритм составляется для его исполнения человеком или ЭВМ. Исполнение алгоритма должно привести к решению соответствующей задачи, получению результата. Так, результатом исполнения алгоритма заварки чая должен быть хорошо заваренный чай, алгоритма на рис. 10.1 — размер зарплаты конкретного сотрудника.

Каждый учащийся должен уметь исполнить алгоритм, тем более что исполнить его много проще, чем составить.

В чем суть процесса исполнения алгоритма? В скрупулезном и педантичном выполнении операций алгоритма в требуемом порядке.

В частности, при исполнении алгоритма, заданного схемой, блоки выполняются строго в порядке, определяемом стрелками, соединяющими их, т.е. после выполнения операций некоторого блока переходим к выполнению операций следующего блока, связанного с первым стрелкой.

Как оформляется процесс исполнения алгоритма? При выполнении любой операции для каждой переменной — результата операции — указывается ее имя и полученное ею значение.

Далее рассмотрим этот процесс подробно для каждого вида алгоритмов, а сейчас покажем, как это делается на примере алгоритма, представленного на рис. 10.1. Исполним его при значении *ST* (стажа работы), равном 18:

```
1. Ввод: ST=18.
2. Проверить: условие — ST<5 выполняется?
```

```
    18<5 — Нет.
3. Проверить: условие — ST≤15 выполняется?
    18≤15 — Нет.
4. ZP=180+(18—15)·10 = 210.
5. Вывод: ZP=210.
6. Вычисления прекратить.
```

То же самое можно записать короче:

```
1. ST=18.
2. ST<5?
    18<5 — Нет.
3. ST<15?
    18<15 — Нет.
4. ZP=180+(18—15)·10=210.
5. Вывод: ZP=210.
6. Останов.
```

Именно эту форму записи процесса исполнения алгоритма будем использовать далее.

Заметим, что исполнение алгоритма может преследовать различные цели:

- научить читать и механически выполнять алгоритмы;
- проверить, правильно ли понимают учащиеся операции, включаемые в алгоритм;
- проверить правильность составленного алгоритма.

10.1.5. Отладка алгоритмов

Алгоритм можно использовать только в том случае, если он верен. Однако на практике в любом достаточно сложном алгоритме (если не принять специальных мер) обязательно содержатся ошибки. «Человеку свойственно ошибаться!» — говорили древние. Это свойственно и составителям алгоритмов. Поэтому любой вновь составленный алгоритм нужно «отладить».

Отладкой алгоритма называется процесс выявления и исправления ошибок в нем. Суть отладки алгоритма в том, что выбирается некоторый набор исходных данных, называемый *тестовым набором* (*тестом*), и задача с этим набором решается дважды[1]: один раз — исполнением алгоритма, второй раз — каким-либо иным способом, исходя из условия задачи, так сказать вручную. При совпадении результатов алгоритм считается верным. В качестве тестового набора можно выбрать любые данные, которые позволяют:

[1] Либо выбирают такой набор данных, при котором результаты решения заранее известны.

1) обеспечить проверку выполнения *всех* операций алгоритма;
2) свести количество вычислений к минимуму.

Примечание 1. Обычно, услышав об отладке, учащиеся удивляются, какой смысл решать задачу на ЭВМ, если предварительно нужно решить ее вручную? Поэтому подчеркнем, что алгоритм обычно составляется для того, чтобы использовать его многократно для решения задачи с разными значениями исходных данных.

Но прежде чем использовать его, убеждаются, что он верен, решая один раз подобную задачу вручную, с тем чтобы в дальнейшем в течение, может быть, многих лет использовать алгоритм без проверки, полностью ему доверяя.

Далее подробно рассмотрим отладку каждого вида алгоритмов, а сейчас проиллюстрируем этот процесс на примере отладки алгоритма задачи 10.1. В этой задаче результат вычисляется одним из трех возможных способов в зависимости от значения ST (стажа работы). В соответствии с этим в тестовый набор включим три варианта значений ST, например такие:

1) $ST = 3$ (для проверки варианта — $ST < 5$);
2) $ST = 8$ (для проверки варианта — $5 < ST \le 15$);
3) $ST = 18$ (для проверки варианта — $15 < ST$).

При каждом значении ST следует решить задачу вручную и исполнить алгоритм[1]. Исходя из условия задачи, несложно установить, что при этих данных получатся такие результаты:

1) $ZP = 130$ тыс. руб.; 2) $ZP = 180$ тыс. руб.; 3) $ZP = 210$ тыс. руб. Это и есть результаты ручного решения задачи.

Теперь исполним алгоритм при тех же значениях ST:

I
1. $ST = 3$.
2. $ST < 5$?
   $3 < 5$ — Да.
3. $ZP = 130$.
   $8 < 15$ — Да.
4. Вывод: $ZP = 130$.
5. Останов.

II
1. $ST = 8$.
2. $ST < 5$?
   $8 < 5$ — Нет.
3. 15?
   $8 < 15$ — Да.
4. $ST = 180$.
5. Вывод: $ZP = 180$.
6. Останов.

Исполнение алгоритма при $ST = 18$, приведенное выше, дало результат: $ZP = 210$. Теперь можно сделать вывод: алгоритм верен, так

[1] Согласно рекомендациям, приведенным в 10.3.3, в тест следовало бы включить и значения ST = 5 и ST = 15, но для иллюстрации принципа отладки достаточно приведенных трех значений.

как результаты решения задачи вручную и соответствующие результаты исполнения алгоритма совпали.

Примечание 2. В некоторых случаях для отладки алгоритма достаточно лишь исполнить его. Например, в тех случаях, когда для оценки результата (верен он или нет) достаточно сопоставить его с исходными данными задачи.

Примечание 3. Следует подчеркнуть, что указанная выше процедура отвечает в первую очередь на вопрос, верен алгоритм или нет. И если верен, то вопрос исчерпан, а если нет, то следует искать место ошибки в алгоритме. Но это уже иной, достаточно сложный процесс, выполняемый интуитивно, исходя из здравого смысла; успех его во многом зависит от опыта.

Контрольные вопросы

1. Что такое алгоритм? В чем отличие этого понятия от понятия «метод решения задачи»?
2. Каковы основные способы описания алгоритмов?
3. В чем суть исполнения алгоритма?
4. Отладка алгоритма — в чем ее цель и суть?
5. Каковы основные принципы составления алгоритмов? Из чего они вытекают?

10.2. Линейные алгоритмы

10.2.1. Простейшие линейные алгоритмы

Линейным называется алгоритм, в котором все этапы решения задачи выполняются строго последовательно. Структура такого алгоритма показана на рис. 10.3.

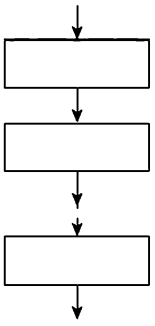


Рис. 10.3

Рассмотрим составление схем линейных алгоритмов на конкретных примерах.

Задача 10.3

Даны переменные A и B. Требуется обменять их значения, т.е. переменная A должна получить значение B, а B — значение A.

Решение.

1. Исходные данные: A, B. Результат: A, B.
2. Метод решения задачи: хотелось бы, чтобы читатель сам нашел его, поэтому сначала дадим ряд пояснений.

Пояснения:

1. Напомним, что в ЭВМ каждая величина хранится в отдельной ячейке. Поэтому задача фактически заключается в том, чтобы поменять местами содержимое двух ячеек.

2. Задача аналогична такой «жизненной» ситуации. Имеются две клетки — в одной находится волк, в другой — заяц. Требуется поменять их местами, т.е. пересадить их из одной клетки в другую.

Эта ситуация должна навеять читателю такой метод решения нашей задачи: введем в рассмотрение еще одну величину, например C, т.е. выделим третью ячейку (клетку), свободную; перенесем значение A в ячейку для C ($C := A$), затем перенесем значение B в ячейку для A и т.д. (рис. 10.4) (на рисунке ячейки изображены квадратиками, операции — стрелками, порядок выполнения операций — цифрами).

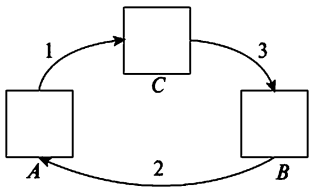


Рис. 10.4

Решение задачи распадается на три этапа. Соответствующие им блоки и порядок их выполнения изображены на схеме алгоритма (рис. 10.5). Для отладки исполним его, взяв в качестве теста такие данные: $A = 10$, $5 = 20$. Результаты исполнения приведены на том же рисунке. Сравнение их с исходными данными задачи свидетельствует о том, что алгоритм работает верно.

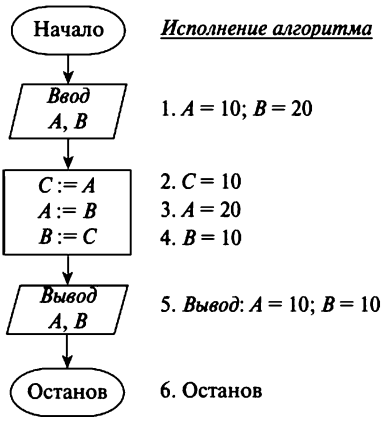


Рис. 10.5

Задача 10.4

Даны величины A, B, C, D. Требуется переместить значения величин: B должно получить значение A; C — значение B; D — значение C.

Решение.

1. Исходные данные A, B, C, D. Результат: A, B, C, D.

2. Метод решения задачи: обычно учащиеся предлагают такую последовательность операций:

$B := A$,

$C := B$,

$D := C$.

Для отладки исполним ее, взяв в качестве теста такие значения величин: $A = 5$; $B = 10$; $C = 20$; $D = 100$. Результаты исполнения:

1) $B = A = 5$; 2) $C = B = 5$; 3) $D = C = 5$.

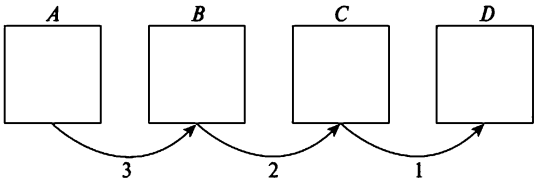


Рис. 10.6

Результаты говорят о том, что алгоритм неверен — перемещения значения величин не произошло. Следует, очевидно, те же операции выполнять в обратном порядке, как показано на рис. 10.6 (здесь обозначения те же, что на рис. 10.4).

В соответствии с рис. 10.6 изобразим схему алгоритма (рис. 10.7).

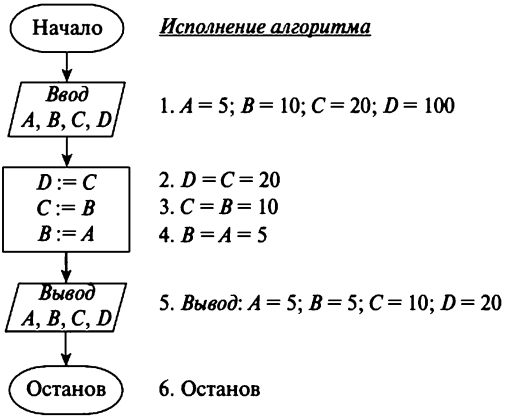


Рис. 10.7

Пояснение. В результате выполнения операции $D := C$ величина D получает новое значение, при этом прежнее значение D стирается.

После же операции $B := A$ величина B получает новое значение, при этом величина A свое значение сохраняет, поскольку ей не присваивалось другое значение — *любая величина сохраняет свое значение, пока ей не будет присвоено новое*, т.е. чтение числа из ячейки памяти не изменяет содержимого ячейки.

Исполнение алгоритма (см. рис. 10.7) при принятых выше значениях исходных величин подтверждает, что алгоритм верен.

10.2.2. Понятие массива

Часто в технике, науке и жизни используются не отдельные числа и величины, а *множества связанных однородных величин*. Так, дата — это совокупность трех чисел, например 19. 10. 88.

С листом бумаги связывается два числа — длина и ширина, например 250×170 мм; с чемоданом — три числа — длина, ширина, высота, например 500×300×150. Многочлен $Y = -7x^4 + 4x^3 + 4x^2 + 2x + 3$ однозначно определяется совокупностью из пяти чисел-коэффициентов: 3, 2, 5, 4, –7.

Несложно представить и другие множества связанных однородных величин с шестью, семью элементами и более. Такие множества широко используются и в информатике, где они называются *массивами*.

Дадим определение массива. Массивом называется упорядоченная совокупность *однородных* величин, обозначенных каждая одним и тем же именем с различными целочисленными *индексами*, изменяющимися по порядку. Индекс (индексы) определяет(ют) положение элемента в массиве. Раскроем смысл этого определения.

Каждому массиву обычно присваивается имя, что дает возможность различать массивы между собой и обращаться к ним по именам.

Различают разные виды массивов в зависимости от их внутреннего строения, взаимного расположения элементов. Так, элементы массива могут располагаться строго последовательно, например {3, 4, 2, 8}. Такие массивы называются *одномерными*.

Каждый подобный массив определяется *именем* и *числом элементов* и обозначается

T(1:n),

где **T** — имя массива; **n** — число элементов массива, например A (1:4).

Множество коэффициентов приведенного выше многочлена можно обозначить как массив K(1:5).

Каждый элемент массива также получает имя — он обозначается именем массива с индексом, равным порядковому номеру элемента. Например, первый элемент массива A будет обозначен как a_1, третий элемент массива K получит имя k_3 и т.д. Имена элементов массива дают возможность различать их между собой и обращаться к любому из них по имени.

Пусть, например, у нас два чемодана с размерами 600×250×700 мм и 500×250×650 мм. Можно представить размеры их — две тройки чисел, как два одномерных массива — $A(1:3)$ и $B(1:3)$.

$$A(1:3) = \{600, 250, 700\} = \{a_1, a_2, a_3\};$$
$$B(1:3) = \{500, 250, 650\} = \{b_1, b_2, b_3\};$$

где $a_1, a_2, a_3, b_1, b_2, b_3$ — имена элементов массива, т.е.

$$a_1 = 600; \quad a_2 = 250; \quad a_3 = 700;$$
$$b_1 = 500; \quad b_2 = 250; \quad b_3 = 650.$$

В рассмотренных примерах фигурировали числовые массивы, но массивы могут быть и текстовыми. Например, список дежурных в классе:

1) Иванов;
2) Петров;
3) Сидоров;
4) Голопупов

можно рассматривать как текстовой массив и обозначить так:

$$SP(1:4) = \{sp_1, sp_2, sp_3, sp_4\},$$

где sp_1 = «Иванов», sp_2 = «Петров» и т.д.

Номера фамилий в массив не включаем, так как они совпадают с индексами элементов.

Рассмотрим теперь ведомость, составленную роно города N (табл. 10.1):

Таблица 10.1

Номер школы	Число выпускников	Число медалистов
5	45	4
14	27	3
26	41	3
32	44	2

Ведомость представляет собой множество из 12 связанных между собой однородных величин — это тоже пример массива. Но элементы ее расположены в четыре строки по три элемента в каждой (три столбца).

Подобного вида таблицы из нескольких строк с равным числом элементов в каждой называют в информатике *двумерными массивами*. В математике подобные массивы называют *матрицами*. Каждый двумерный массив определяется *именем*, *числом строк* и *столбцов* и обозначается:

T(l:m, l:n),

где **T** — имя массива; **m(n)** — число строк (столбцов).

Нашу ведомость можно обозначить так: $B(1{:}4, 1{:}3)$, т.е. как массив B из четырех строк и трех столбцов.

Строки подобных массивов нумеруются по порядку сверху вниз, а столбцы — слева направо.

Каждый элемент двумерного массива определяется номером строки и столбца, на пересечении которых он находится, и в соответствии с этим обозначается именем массива с двумя индексами: первый — номер указанной строки, второй — номер столбца. Например, a_{15}, c_{26}.

Элементы нашей ведомости получат такие обозначения:

$$B(1:4,1:3)=\begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \\ b_{41} & b_{42} & b_{43} \end{pmatrix}=\begin{pmatrix} 5 & 45 & 4 \\ 14 & 27 & 3 \\ 26 & 41 & 3 \\ 32 & 44 & 2 \end{pmatrix},$$

т.е. $b_{11} = 5$, $b_{12} = 45$, $b_{13} = 4$ и т.д.

Как и одномерные, двумерные массивы могут быть не только числовыми, но и текстовыми. Например, список спортсменов класса (табл. 10.2) можно обозначить как текстовый массив $S(1{:}3, 1{:}2)$ т.е. s_{11} = «Петров», s_{12} = «Борьба» и т.д.

Таблица 10.2

Фамилия	Вид спорта
Петров	Борьба
Иванов	Плавание
Сидоров	Бег

Рассмотрим теперь меню школьной столовой (табл. 10.3).

Таблица 10.3

Блюдо	Цена
Борщ	0,35
Котлета	0,40
Каша	0,20
Чай	0,03

Меню является совокупностью из восьми связанных величин, но представить их как один массив нельзя, так как здесь объединены разнородные величины — текстовые и числовые.

Поэтому следует ввести два одномерных массива разного типа — один текстовый и один числовой:

$$B(1{:}4) = \{b_1, b_2, b_3, b_4\} = \{\text{«борщ»}, \text{«котлета»}, \text{«каша»}, \text{«чай»}\},$$
$$C(1{:}4) = \{c_1, c_2, c_3, c_4\} = \{0{,}35;\ 0{,}40;\ 0{,}20;\ 0{,}03\}.$$

Как представляется массив в ЭВМ? Как отмечалось, в ЭВМ для каждой величины выделяется ячейка памяти. Аналогично для каждого элемента массива также выделяется отдельная ячейка памяти, в которой хранится число (или текст), выражающее значение элемента.

Поэтому использование массивов большого размера связано с большим расходом памяти. Например, массив $D(1{:}10, 1{:}20)$ потребует 200 ячеек памяти; массив $P(1{:}100, 1{:}10)$ — 1000 ячеек.

Примечание. Далее массивы и элементы массивов будем обозначать заглавными и строчными буквами соответственно, но в тех задачах, решение которых заканчивается составлением программы, для обозначения указанных величин будем использовать лишь заглавные буквы, с тем чтобы сохранить преемственность в изображении одних и тех же величин в условии задачи, алгоритме и программе.

10.2.3. Линейные алгоритмы с массивами

Составление алгоритмов такого типа не имеет особой специфики. Порядок составления их определяется Основными принципами алгоритмизации. Эти принципы можно дополнить еще одним.

В сложных задачах выделяются в первую очередь наиболее крупные этапы решения задачи (подзадачи) и изображается порядок выполнения их в виде схемы, называемой *укрупненной*. Далее каждая подзадача рассматривается отдельно как самостоятельная задача и для каждой составляется своя схема алгоритма. Только после этого составляется *подробная* схема алгоритма всей задачи совмещением схем отдельных подзадач.

Этот подход взят из жизни и, наверное, не раз использовался учащимися, но только при решении иных задач. Вспомните план проведения мероприятий, план конспекта — что это такое? Это перечень основных наиболее крупных мероприятий, записанных с указанием порядка их выполнения; это список основных разделов конспекта, записанных в порядке их изложения. С этой точки зрения укрупненная схема алгоритма есть план алгоритма, изображенный в виде схемы. Именно так ее надо понимать и в соответствии с этим составлять.

Задача 10.5

Задан массив $B(1{:}4)$. Каждому элементу массива присвоить значение соседнего с ним справа. Последнему элементу присвоить значение первого.

К этой задаче сводится, например, следующая: в списке учащихся первую по списку фамилию поставить на последнее место. В результате второй элемент списка станет первым, третий элемент — вторым, четвертый — третьим и т.д. Здесь список можно рассматривать как одномерный текстовый массив.

Решение.

1. Исходные данные: массив $B(1{:}4) = \{b_1, b_2, b_3, b_4\}$.

Результат: массив $B(1{:}4)$.

Пояснение 1. Обычно учащиеся считают, что результатом должен быть другой массив, не совпадающий с исходным. Но это не так! Как отмечалось, для каждого элемента массива выделяется отдельная ячейка памяти ЭВМ. Новый массив потребует новых ячеек. В нашем случае новые ячейки не требуются, так как здесь меняется только содержимое исходных ячеек, поэтому результатом будет тот же массив $B(1{:}4)$.

2. Метод решения задачи. Для лучшего понимания его приведем пояснение.

Пояснение 2. Для нашей задачи, как и для задачи 10.3, можно подобрать аналогичную «жизненную» ситуацию. Например, имеются четыре клетки с кроликами, расположенные в ряд. В каждой клетке один кролик. Требуется пересадить каждое животное в соседнюю слева клетку, а из первой клетки пересадить в последнюю. Размеры клетки не позволяют помещать в одну клетку более одного кролика.

Порядок решения этой задачи, надо полагать, читателю ясен. По аналогии с ним можно сформулировать и метод решения нашей задачи со всеми подробностями, но предварительно запишем «план» ее решения.

1. Очевидно, потребуется свободная ячейка (клетка). Поэтому, как и в задаче 10.3, введем в рассмотрение некоторую величину D и первой операцией будет такая: $D := b_1$ т.е. значение b_1 перенесем в свободную ячейку.

2. Значения элементов $b_2 - b_4$ переносим на место элементов $b_1 - b_3$ соответственно (в детали этого процесса не вникаем, только выясняем — «что выполнить?», а «как это выполнить?» — нас пока не должно интересовать).

3. Последняя операция — перенесем значение b_1 в ячейку для b_4, т.е. $b_4 := D$.

Все сказанное схематично изображено на рис. 10.8. Обозначения здесь те же, что и на рис. 10.4.

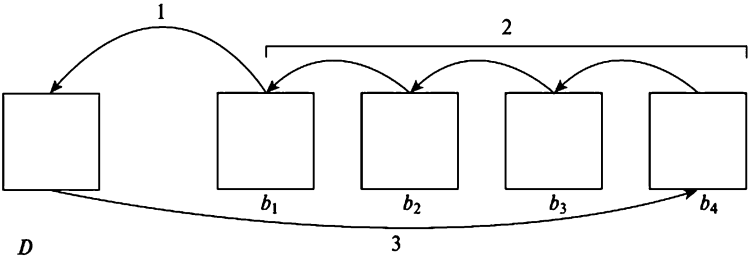


Рис. 10.8

Изобразим теперь план решения в виде схемы алгоритма (рис. 10.9). Это и будет укрупненная схема алгоритма задачи. Она отражает суть нашей задачи — составляющие ее подзадачи и порядок их решения.

Далее можем для каждой подзадачи отдельно составить схему алгоритма ее решения. Однако в этом нуждается лишь задача блока 4 (блоки, подобные блоку 4, будем называть *укрупненными*; они имеют особое обозначение на схеме алгоритма).

Процесс решения указанной задачи раскрыт на рис. 10.8, изобразим его в виде схемы алгоритма (рис. 10.10).

В блоке 4—1 (см. рис. 10.10) вместо слова *Начало* записано *Блок 4*. И впредь схему алгоритма, раскрывающую некоторый укрупненный блок, будем начинать с имени этого блока; последний блок такой схемы оставляем незаполненным, так как здесь он обозначает не прекращение вычислений, а завершение некоторого участка схемы алгоритма.

Далее объединяем схему алгоритма решения всей задачи (см. рис. 10.9) со схемой укрупненного блока 4 (см. рис. 10.10) — каким образом? Внутри изображения блока 4 на схеме рис. 10.9 помещаем подробную схему этого блока без

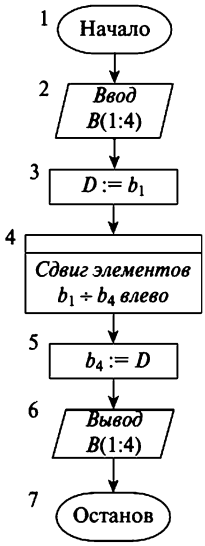


Рис. 10.9

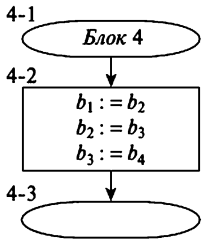


Рис. 10.10

каких-либо изменений (опуская лишь первыйблок (с названием схемы) и последний «пустой» блок).

В результате получим подробную схему алгоритма всей задачи (рис. 10.11).

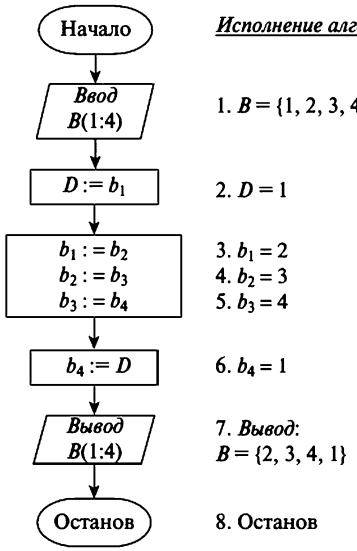


Рис. 10.11

Этот же принцип совмещения укрупненной схемы со схемами отдельных ее блоков будем использовать и в последующих задачах.

Для отладки алгоритма задачи 10.5 исполним его, приняв в качестве тестового набора такие значения элементов массива $B(1{:}4) = \{b_1, b_2, b_3, b_4\} = \{1, 2, 3, 4\}$.

Анализ результатов работы алгоритма (см. рис. 10.11) говорит о том, что алгоритм правильно решает задачу.

Задача 10.6

В обувной магазин поступила партия обуви различных размеров и разной цены. Данные представлены в табл. 10.4.

Таблица 10.4

Размеры обуви	Количество пар	Цена одной пары, тыс. руб.
38	120	3
39	50	4
41	93	9

Определить стоимость всей партии обуви после повышения цен на обувь в N раз и новые цены.

Решение

Очевидно, при работе с данными табл. 10.4 потребуются обозначения (имена) для ее элементов. Поэтому естественно представить их как двумерный массив (матрицу) $A(1{:}3,\ 1{:}3)$:

$$A(1:3,1:3)=\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}.$$

Результаты можно представить в виде массива $R(1{:}3)$ — новые цены, и переменной S — стоимость всей партии обуви.

Теперь задачу можно сформулировать математически: задана матрица $A(1{:}3,\ 1{:}3)$; умножить элементы третьего столбца этой матрицы на N (образовать массив $R(1{:}3)$) и вычислить значение S по формуле

$$S = a_{12} \cdot r_1 + a_{22} \cdot r_2 + a_3 \cdot r_3. \qquad (10.1)$$

Пояснение. Формулировка задачи неоднозначна (это типично для реальных задач) — результаты умножения на величину N допускают различные представления:

- можно записать их на место исходных элементов в массив A;
- можно организовать из них новый массив, как мы и поступили, руководствуясь соображением — *не следует без необходимости искажать исходные данные*.

Метод решения задачи (в общем виде) (план решения задачи):

1) умножение элементов третьего столбца матрицы A на N и образование массива $R(1{:}3)$;
2) вычисление S по формуле (10.1).

Соответствующая укрупненная схема алгоритма изображена на рис. 10.12.

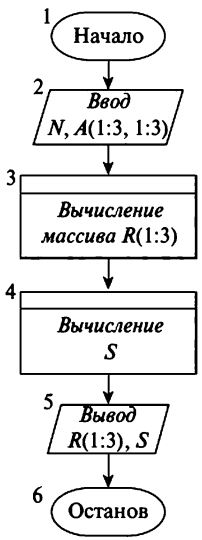


Рис. 10.12

Теперь по отдельности рассмотрим операции блоков 3 и 4.

Блок 3.

$$r_1 = a_{13}N; \quad r_2 = a_{23}N; \quad r_3 = a_{133}N.$$

Блок 4.

$$S := a_{12}r_1 + a_{22}r_2 + a_{33}r_3.$$

Включив указанные операции в блоки 3 и 4 схемы рис. 10.12, соответственно получим подробную схему алгоритма всей задачи (рис. 10.13).

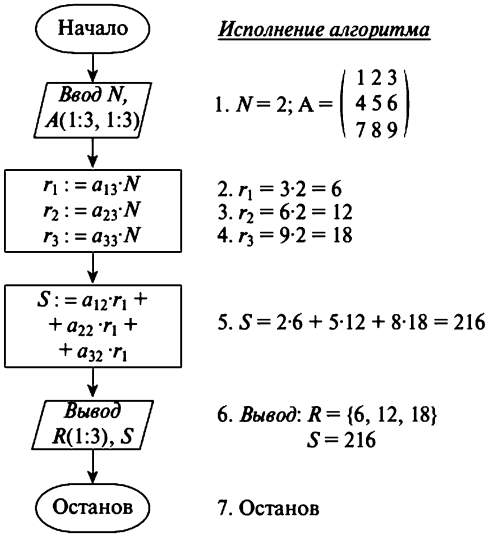


Рис. 10.13

10.2.4. Отладка линейных алгоритмов

Выше были рассмотрены примеры отладки алгоритмов нескольких задач (задачи 10.3—10.5). Однако с точки зрения процесса отладки эти задачи нетипичны, так как в них для оценки правильности решения достаточно сопоставить результаты решения с исходными данными задачи. А для этого в свою очередь достаточно было исполнить алгоритм, что весьма элементарно. Однако и в общем случае отладка линейных алгоритмов достаточно тривиальный процесс, не имеющий особой специфики по сравнению с процедурой, описанной в разделе 10.1.4 и выполняемый *на одном наборе данных*. Покажем это на примере отладки алгоритма задачи 10.6:

а) выберем для отладки этого алгоритма в качестве тестового набора, например, такие значения исходных данных:

$$N = 2;\ A(1:3, 1:3) = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix};$$

б) решим задачу при этих данных вручную. Получим результаты:

$$R(1{:}3) = \{6, 12, 8\}; \quad S = 216;$$

в) исполним алгоритм при тех же значениях исходных данных. Результаты исполнения приведены на рис. 10.13.

Сравнение их с результатами ручного решения задачи показывает, что они совпадают. Отсюда вывод: «алгоритм верен».

Теперь можно использовать алгоритм для решения задачи с любыми данными, в том числе и с данными табл. 10.4.

Задача 10.7

В матрице $B(1{:}3, 1{:}3)$ определить суммы элементов первой строки и последнего столбца.

Предоставляем возможность читателю самому решить задачу и выполнить отладку полученного алгоритма.

Контрольные вопросы

1. Какие алгоритмы называют линейными?
2. Что такое массив? Что такое одномерный и двумерный массивы?
3. Как обозначаются массив и элемент массива?
4. Что такое укрупненная и подробная схемы алгоритма? В чем их отличие?

Задачи для самостоятельного решения

1. В массиве $A(1{:}3, 1{:}3)$ элементы главной диагонали поставить на место соответствующих элементов третьей строки и определить сумму угловых элементов (a_{11}, a_{13}, a_{31}, a_{33}).

2. В массиве $D(1{:}3, 1{:}4)$ определить сумму R элементов строки с номером k и каждый элемент столбца с номером l умножить на R.

3. Используя данные табл. 10.1, определить общее число медалистов в районе и процентное отношение числа медалистов к общему числу выпускников в каждой школе района.

4. Используя данные табл. 10.3, определить общую стоимость обеда в столовой. Определить, во сколько раз возрастет стоимость обеда, если цена котлеты увеличится вдвое.

10.3. Разветвляющиеся алгоритмы

10.3.1. Понятия и определения

Разветвляющимся называется такой алгоритм, в котором выбирается один из нескольких возможных путей (вариантов) вычислительного

процесса. Каждый подобный путь называется *ветвью алгоритма*. Примером такого алгоритма является алгоритм решения задачи 10.1.

Логический блок. Признаком разветвляющегося алгоритма является наличие операций проверки условия. Обычно различают два вида условий — *простые* и *составные*.

Простым условием (*отношением*) называется выражение, составленное из двух арифметических выражений или двух текстовых величин, связанных одним из знаков: $<$, $>$, $\le$, $\ge$, $=$, $\ne$.

Примеры отношений:

$$X + Y < 7; \quad Y \ge \ln Z; \quad R \le X^2 + \sqrt{D}; \quad 2 \ne 3.$$

«МАМА» ≠ «ПАПА»; B = «ДА» (B — текстовая переменная).

Определение и использование составных условий рассмотрено в главе 1.

В схеме алгоритма операцию проверки условия выполняет, как отмечалось, *логический блок*. Он изображается ромбом, внутри которого указывается проверяемое условие (отношение), и имеет два выхода: *Да* и *Нет*.

Если условие (отношение) истинно (выполняется), то выходим из блока по выходу *Да*; если ложно (не выполняется) — по выходу *Нет*.

Примечания. 1. Условие (отношение), содержащееся в логическом блоке, может быть заменено на противоположное:

$$X < 10 \to X \ge 10 : K = 15 \to K \ne 15.$$

При этом выходы *Да* и *Нет* меняются местами.

2. Логический блок может содержать лишь одно условие.

В соответствии с этим блок, представленный на рис. 10.14, записан неверно, так как двойное неравенство $7 < X < 9$ содержит фактически два простых условия: $7 < X$ и $X < 9$.

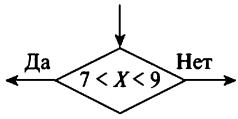


Рис. 10.14

Типовая схема разветвляющегося алгоритма. Несмотря на большое разнообразие разветвляющихся алгоритмов, желательно все же добиваться некоторого однообразия их структуры, для чего следует приводить их к виду схемы, представленной на рис. 10.15.

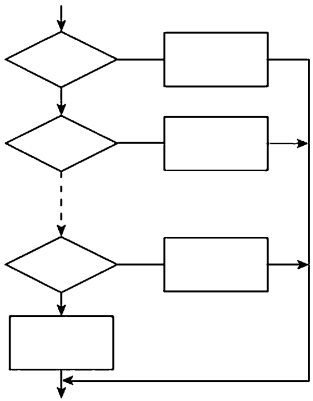


Рис. 10.15

Хотя это и не всегда возможно, но к этому нужно стремиться.

Подобный подход позволяет строить простые и однообразные по структуре программы, а значит и легко понимаемые. Схему, изображенную на рис. 10.15, будем считать типовой.

10.3.2. Составление разветвляющихся алгоритмов

Составление таких алгоритмов выполняется в соответствии с Основными принципами алгоритмизации, здесь работает та же схема: «формулировка задачи — метод — алгоритм».

Рассмотрим несколько классов задач, решение которых требует составления разветвляющихся алгоритмов.

I. Существуют задачи, связанные с вычислением функций, заданных несколькими арифметическими выражениями (формулами). Это очень распространенные задачи, особенно в инженерных и экономических расчетах. В общем случае для их решения требуется составление не простых алгоритмов, они рассмотрены в главе 9. В частном случае, например при вычислении одного значения указанной функции, алгоритм решения задачи является чисто разветвляющимся. Приведем пример такой задачи.

Задача 10.8

Вычислить значение Y по одной из формул:

$$Y=\begin{cases} X+a, & \text{если } X<10; \quad (1)\\ X-b_1, & \text{если } 10\le X\le 20; \quad (2)\\ c+b_2, & \text{если } 20<X<30; \quad (3)\\ c-b_3, & \text{если } 30<X, \quad (4)\end{cases}$$

где $X=\sqrt[3]{a+b_1}$.

Здесь b_1, b_2, b_3 будем рассматривать как элементы массива $B(1:3)$.

Пояснение 1. Условие задачи означает, что Y зависит от X, а X может принимать любое значение из интервала $(-\infty, +\infty)$. Этот интервал разбит на четыре области и в пределах каждой области Y вычисляется по одной из формул. Это хорошо видно на графике функции Y (рис. 10.16). Здесь значение $x = 10$ является границей для первой и второй областей; $x = 20$ — для второй и третьей, а $x = 30$ — для третьей и четвертой областей.

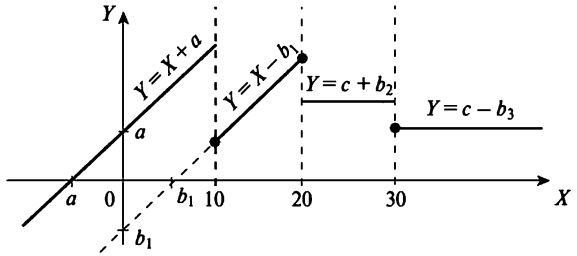


Рис. 10.16

Решение

1. Исходные данные: c, a, $B(1:3)$. Результаты: X, Y.
2. Метод решения задачи (в общем виде):
   - ввод c, a, $B(1:3)$;
   - вычисление X;
   - вычисление Y;
   - вывод X, Y.
3. Укрупненная схема алгоритма (фрагмент схемы) (рис. 10.17).

Рассмотрим подробно блок 2 этой схемы.

Метод решения задачи: он должен обеспечить выявление области, которой принадлежит значение X, для чего достаточно проверить заданные условия по порядку, например сверху вниз:

Проверяем $X < 10$,	если условие истинно (*Да*), вычисляем $Y = X + a$;
	если условие ложно (*Нет*), проверяем:
$10 \le X \le 20$,	если *Да*, вычисляем $Y = X - b_1$;
	если *Нет*, проверяем:
$20 < X < 30$,	если *Да*, вычисляем $Y = c + b_2$;
	если *Нет*, вычисляем $Y = c - b_3$.

Процесс решения задачи распадается на этапы (операции):

«проверка истинности условия», «вычисление по формуле». Согласно Основным принципам алгоритмизации, заменяем каждую такую операцию соответствующим блоком — ромбом, прямоуголь-

ником и соединяем согласно порядку их выполнения линиями. В итоге получаем схему блока 2 (рис. 10.18). Объединив ее с блоками 1 и 3, получим схему алгоритма Блок 2 всей задачи.

Пояснение 2. Блок, общий для нескольких ветвей, можно изображать в схеме один раз, например блок *Вывод* в схеме рис. 10.18.

Пояснение 3. Изображая в схеме логический блок, следует выяснить и указать в явном виде, какому условию отвечает проверяемая переменная (в нашем случае — X) на выходе *Нет* этого блока (см. рис. 10.18). Полученную информацию следует использовать при реализации последующих условий. Так, для проверки условия (2) нашей задачи ($10 \le X \le 20$) достаточно проверить отношение $X \le 20$, поскольку на входе второго логического блока (рис. 10.18) отношение $10 \le X$ всегда истинно. Аналогично для проверки условия (3) достаточно проверить одно отношение: $X \le 30$; информация на выходе *Нет* третьего логического блока (условие $30 \le X$ — истинно) позволяет исключить отдельный блок для проверки условия (4).

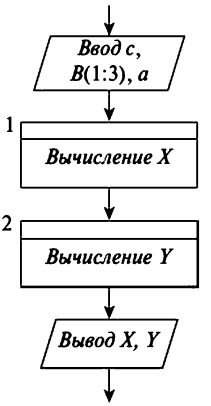


Рис. 10.17

Задача 10.9

Вычислить значение функции Z при одном значении X: $Z = Y^2 + X^2$, где

$$Y = \begin{cases} X + a, & \text{если } X < 10; \\ |X| + 2_1, & \text{если } X < 3; \\ X^2 - a_1, & \text{если } X = 3; \\ a_2^2, & \text{если } 10 \le X. \end{cases}$$

Решение

Исходные данные: $A(1:2)$, X, C.

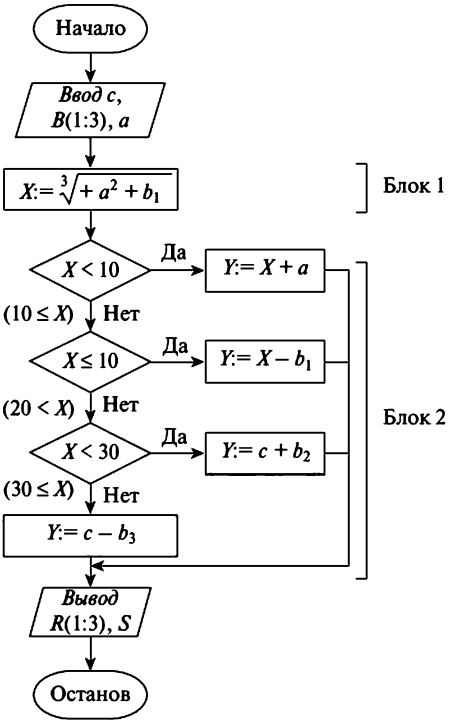


Рис. 10.18

Результат: Y, Z.

Метод решения задачи (в общем виде):

- ввод C, X, $A(1:2)$;
- вычисление Y;
- вычисление Z;
- вывод Y, Z.

Составляем укрупненную схему алгоритма (рис. 10.19).

Далее рекомендуем читателю самому сформулировать метод решения задачи блока 3 и составить для него схему алгоритма.

Результат можно сравнить со схемой на рис. 10.20.

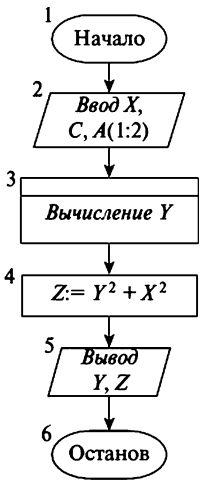


Рис. 10.19

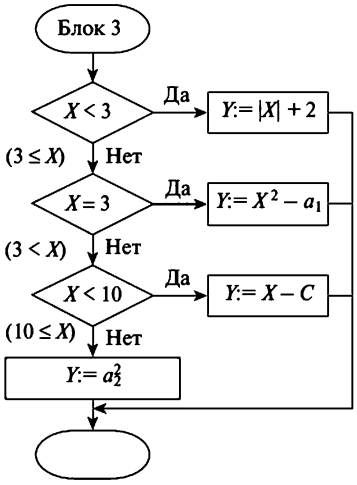


Рис. 10.20

Следующую задачу рекомендуем решить читателю самостоятельно.

Задача 10.10

Вычислить значение функции Z по одной из формул:

$$Z = \begin{cases} X - b_1, & \text{если } X < 6; \\ X^2 + b_2, & \text{если } X = 15 \text{ или } 20; \\ b_3 + X & \text{в остальных случаях.} \end{cases}$$

II. Еще один распространенный вид задач — логические (так их иногда называют). К ним относятся задачи определения минимума, максимума некоторого числа величин, задачи упорядочивания, сортировки данных, многие задачи обработки массивов. В таких задачах доля арифметических операций обычно мала. Это достаточно

сложные задачи, однако в простейших случаях при небольшом числе данных они приводят к построению несложных алгоритмов разветвляющейся структуры.

Рассмотрим примеры подобных задач.

Задача 10.11

Определить $Y = \max\{X, Z\}$.

Решение

Исходные данные: X, Z. Результат: Y.

Метод решения задачи очевиден и предполагает выполнение трех основных операций. Применяя Основные принципы алгоритмизации, получим схему алгоритма. Фрагмент ее приведен на рис. 10.21.

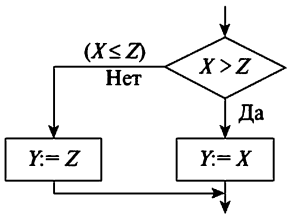


Рис. 10.21

Задача 10.12

Определить значение наибольшего элемента главной диагонали матрицы $A(1:3, 1:3)$.

Исходные данные:

$$A(1:3,1:3) = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}.$$

Результат: Y, т.е. $Y = \max\{a_{11}, a_{22}, a_{33}\}$.

Рассмотрим два варианта решения задачи.

Первый вариант. Метод решения задачи:

Проверяем $a_{11} > a_{22}$,	
если *Да*, проверяем $a_{11} > a_{33}$,	если *Да* — $Y = a_{11}$;
	если *Нет* — $Y = a_{33}$;
если *Нет*, проверяем $a_{22} > a_{33}$,	если *Да* — $Y = a_{22}$;
	если *Нет* — $Y = a_{33}$.

Изображая каждую операцию этого процесса соответствующим блоком, согласно Основным принципам алгоритмизации, получим схему алгоритма, фрагмент которой приведен на рис. 10.22.

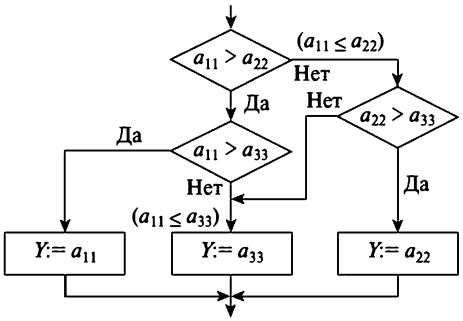


Рис. 10.22

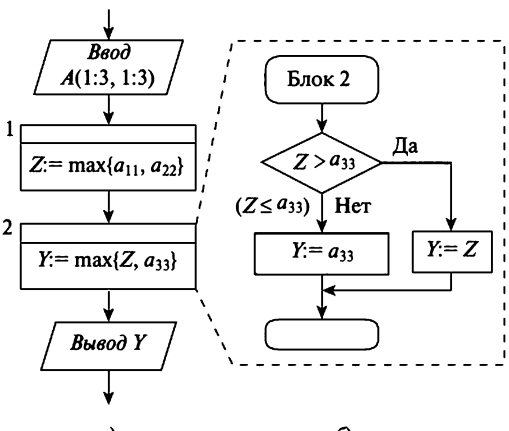
Второй вариант.

Метод решения задачи (в общем виде) (план решения):

- ввод $A(1{:}3,\ 1{:}3)$;
- определение $Z = \max\{a_{11}, a_{22}\}$;
- определение $Y = \max\{Z, a_{33}\}$;
- вывод Y.

Укрупненная схема алгоритма, отражающая этот процесс (точнее ее фрагмент), приведена на рис. 10.23, *а*.

Теперь составим схемы алгоритмов каждого из блоков 1 и 2 в отдельности (см. рис. 10.23, *а*). Они аналогичны схеме алгоритма задачи 10.11, поэтому изобразим лишь схему блока 2 (рис. 10.23, *б*). Объединяя эти схемы, получим подробную схему алгоритма всей задачи (рис. 10.24).



а) *б*)

Рис. 10.23

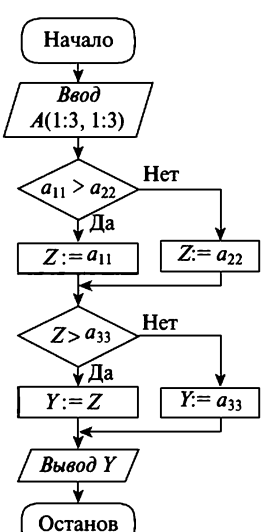


Рис. 10.24

Вывод. Сравнение двух схем одной и той же задачи (рис. 10.22 и 10.24) показывает, что в данном случае подход, основанный на разбиении задачи на независимые подзадачи и составлении укрупненной схемы, более эффективен.

Этот вывод, как показывает опыт, справедлив и в общем случае.

III. Часто встречаются задачи, в которых используются не только простые условия, но и составные. Что такое составное условие, использование таких условий с операциями **и**, **или**, **нет** рассмотрено в главе 1.

Например, контролер ОТК завода решает задачу — отобрать из числа изготовленных болты длиной l, допустим 18 мм, и диаметром d не менее 7,98 мм и не более 8,01 мм.

Другой пример: школьник выяснил, что сможет попасть в кино при условии, если в кассе есть билеты по цене 40 или 50 руб.

В первом примере мы имеем дело с тремя отношениями, связанными между собой союзом «и» и частицей «не», во втором — с двумя отношениями, связанными союзом «или».

Условия наших примеров в алгоритме могут выглядеть в виде составных условий таким образом:

первое — ($l = 18$) **и** (**не** ($d \le 7{,}98$)) **и** (**не** ($d > 8{,}01$)) или,

что то же: ($l = 18$ **и** $d \le 7{,}98$ **и** $d \le 8{,}01$);

второе — $C = 40$ **или** $C = 50$.

Рассмотрим еще примеры составных условий.

Пример 1. Дана система координат на плоскости. Условие — точка $M(x, y)$ лежит в первой четверти — является составным и может быть записано в таком виде:

$$x > 0 \quad \textbf{и} \quad y > 0.$$

Пример 2. Условие — точка $R(x, y)$ лежит на одной из осей координат — является составным и может быть записано так:

$$x = 0 \quad \textbf{или} \quad y = 0.$$

Реализация логических выражений (составных условий). Возможны два подхода к составлению алгоритмов задач, содержащих составные условия.

1. Составное условие рассматривается как «единое и неделимое» и в схеме изображается одним логическим блоком. Такой подход допустим, если в языке программирования, на который мы ориентируемся, составные условия разрешены.

2. Для каждого составного условия изображается схема алгоритма, реализующая ее. В такой схеме каждое отношение составного условия изображается отдельным логическим блоком.

Подобная схема имеет два выхода (*Да* и *Нет*) и обеспечивает «передачу управления» (так принято говорить) на один выход, если составное условие выполняется, и на другой — если оно не выполняется.

Этот подход неизбежен, если используемый язык программирования не допускает применения составных условий.

Далее будем ориентироваться на такие версии Бейсика, в которых составные условия допустимы, и будем использовать первый подход, но для начала рассмотрим решения задач 10.13 и 10.14 с использованием второго подхода, что должно помочь читателю «прочувствовать» смысл условий **«А и В»**, **«А или В»**.

Задача 10.13

Вычислить значение Y по одной из формул:

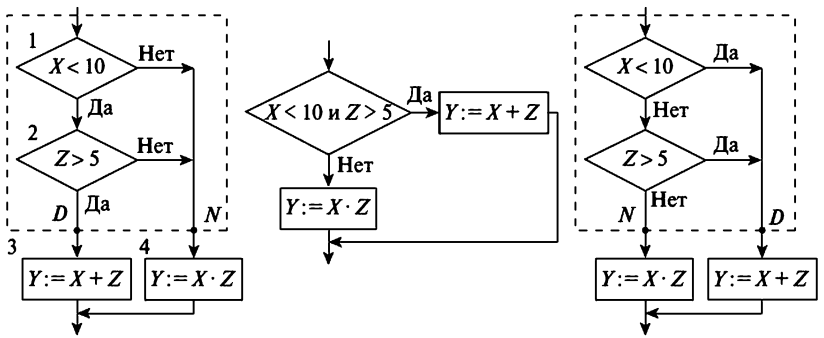
$$Y = \begin{cases} X + Z, & \text{если } X < 10 \text{ и } Z > 5; \\ X \cdot Z & \text{в остальных случаях.} \end{cases}$$

Решение

Составим схему алгоритма задачи, используя второй подход. Метод решения задачи вытекает из определения составного условия **«А и В»**.

Проверяем $X < 10$,	
если *Да*, проверяем $Z > 5$,	если *Да* — $Y = X + Z$;
	если *Нет* — $Y = X \cdot Z$.
если *Нет* — $Y = X \cdot Z$;	

Руководствуясь Основными принципами алгоритмизации, составляем схему алгоритма задачи (рис. 10.25, *а*).



а) *б*) *в*)

Рис. 10.25

Здесь блоки 1 и 2 образуют схему, реализующую составное условие **«A и B»**. Точки *D* и *N* — выходы *Да* и *Нет* соответственно схемы, реализующей указанное условие.

Отметим, что при использовании первого подхода алгоритм решения задачи 10.13 описывается простейшей схемой (рис. 10.25, *б*).

Задача 10.14

Изменить формулировку задачи 10.13, заменив союз «и» на «или». Для полученной задачи самостоятельно составить схему алгоритма и сравнить ее со схемой рис. 10.25, *в*.

Примечание. Далее и простые и составные условия будем называть одним термином — *логические выражения*.

10.3.3. Отладка разветвляющихся алгоритмов

Особенность отладки разветвляющихся алгоритмов состоит в следующем: для проверки правильности всех ветвей алгоритма *тест должен включать несколько наборов исходных данных* — их число должно быть не менее числа ветвей алгоритма.

В случае вычислительных задач, когда диапазон изменения аргумента разбивается на области (интервалы, отрезки и т.д.), целесообразно проверять правильность работы алгоритма в одной из внутренних точек и в граничных точках каждой области.

Тест в подобных случаях удобно записывать в виде таблицы, включая в нее и результаты решения нашей задачи вручную, и результаты исполнения алгоритма.

Продемонстрируем процесс отладки разветвляющегося алгоритма.

Задача 10.15

Вычислить значение *Y* по одной из формул:

$$Y = \begin{cases} X + A, & \text{если } X < 10; \\ X + B, & \text{если } 10 \le X \le 23; \\ A^2, & \text{если } 23 < X. \end{cases}$$

Здесь диапазон значений *X* разбит на три области: $(-\infty; 10)$; $[10, 23]$; $(23, +\infty)$.

Схема алгоритма задачи приведена на рис. 10.26.

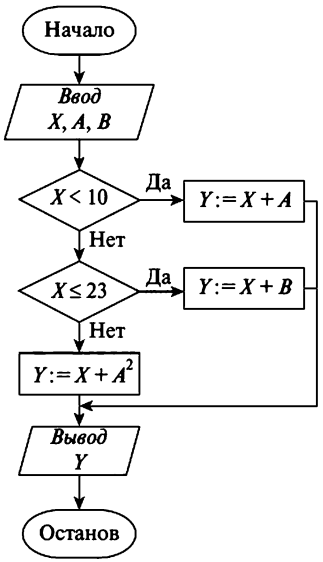


Рис. 10.26

Отладка алгоритма

А. *Выбор теста*. Выберем, например, такие данные с учетом приведенных выше рекомендаций: $x = 5$, $x = 15$, $x = 25$ — внутренние точки областей; $x = 10$; $x = 23$ — границы областей. Значения A и B для всех наборов выбираем постоянными и такими, чтобы максимально упростить вычисления. Все данные сведем в табл. 10.5.

Таблица 10.5

Номер набора	X	A	B	$Y_{ручн}$	$Y_{алг}$
1	5	2	3	7	7
2	10	2	3	13	13
3	15	2	3	18	18
4	23	2	3	26	26
5	25	2	3	29	29

Б. *Ручное решение*. Его выполняем исходя лишь из условия задачи и записываем в графу «$Y_{ручн}$» табл. 10.5.

В. *Исполнение алгоритма*, представленного на рис. 10.26.

Первый набор данных: $X = 5$, $A = 2$, $B = 3$.

1. $X < 10? - 5 < 10 -$ Да.
2. $Y = X + A = 5 + 2 = 7$.
3. Вывод: $Y = 7$.
4. Останов.

Второй набор данных: $X = 10$, $A = 2$, $B = 3$.

1. $X < 10? - 10 < 10 -$ Нет.
2. $X < 23? - 10 < 23 -$ Да.
3. $Y = X + B = 10 + 3 = 13$.
4. Вывод: $Y = 13$.
5. Останов.

Третий набор данных: $X = 15$, $A = 2$,? $= 3$.

1. $X < 10? - 15 < 10 -$ Нет.
2. $X < 23? - 15 < 23 -$ Да.
3. $Y = X +? = 15 + 3 - 18$.
4. Вывод: $Y = 18$.
5. Останов.

Исполнить алгоритм на остальных тестовых наборах данных предоставляем читателю. Результаты исполнения запишем в последний столбец табл. 10.5.

Г. *Сравнение результатов*. Результаты решения задачи вручную и исполнения алгоритма совпали. Вывод: алгоритм верен.

Далее рекомендуем читателю выполнить отладку алгоритмов задач 10.10, 10.14, выделенных для самостоятельного решения.

Контрольные вопросы

1. Какой алгоритм называется разветвляющимся? Что такое ветвь алгоритма?
2. Какой блок называется логическим?
3. Какое условие называется составным?
4. В чем особенность отладки разветвляющихся алгоритмов?

Задачи для самостоятельного решения

Для каждой из приведенных ниже задач составить схему алгоритма.

В задачах 2—4 требуется вычислить значение y по одной из заданных формул.

Приведенные в условиях задач величины с индексами следует рассматривать как элементы массивов (одномерных или двумерных).

1. Для каждого приведенного ниже условия задачи записать соответствующее логическое выражение (составное условие), связывающее данные задачи:

   а) три величины A, B, C записаны в порядке возрастания ($A < B < C$);

   б) массивы $A(1:3)$ и $B(1:3)$ равны;

   в) массивы $A(1:2)$ и $B(1:2)$ не равны;

   г) все элементы матрицы $A(1:2, 1:2)$ равны.

2. $$y = \begin{cases} \ln|x + a^2|, & \text{если } x < 7; \\ \sqrt{|b_{11} + b_{21}|} + b_{22}, & \text{если } x = 10; \\ \dfrac{0{,}35 b_{12}}{0{,}5|b_{11} \cdot b_{22}| + 2} & \text{в остальных случаях.} \end{cases}$$

3. $$y = \begin{cases} \sqrt[3]{\dfrac{\ln x + 5{,}5|a_1|}{1{,}5x}}, & \text{если } 15 < x < 21; \\ a_1 x^2 + a_2 x + a_3 & \text{в остальных случаях.} \end{cases}$$

4. $$y = \begin{cases} 1{,}5x - \sqrt{|c_1| + e^{c_2}}\,|, & \text{если } x < 10; \\ x\left(\ln|c_1| + 0{,}5c_3\right)^2, & \text{если } 15 < x \le 25; \\ \text{вывести значение «}y\text{ не определено»} & \text{в остальных случаях.} \end{cases}$$

5. $w = \begin{cases} \dfrac{c_1 x^3 + c_3 x + \sqrt{|c_1|}}{4x - 3{,}0b} & \text{если знаменатель не равен нулю или } x < 5; \\ \ln|c_1 x + 2{,}5b|, & \text{в остальных случаях;} \end{cases}$

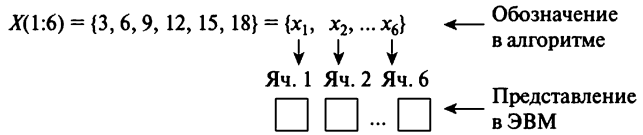
   $\upsilon = w^2 + c_1 x$.
6. Массив $B(1{:}3)$ — целый. Какому элементу массива равна величина D? $D = a + a^2 + a^3$ (a — целая величина).
7. Лежит ли в первой четверти точка P с координатами a, b?

   Здесь $a = \sqrt{3{,}3|d_1 - d_3|} - k$; $b = d_2^2 - \ln(d_1 + 3)$ и $d_1 > 0$; $d_2 > 0$.
8. Вывести значения x_1 x_2, x_3 в порядке возрастания.

10.4. Циклические алгоритмы с одним циклом

10.4.1. Понятия и определения

Представление переменной величины в ЭВМ и обозначение ее в алгоритме. Пусть величина X принимает значения: 3, 6, 9, 12, 15, 18. Их можно представить как массив $X(1{:}6)$. В этом случае каждое значение X в схеме алгоритма обозначается именем массива с некоторым индексом, а в ЭВМ каждое значение X будет храниться в отдельной ячейке:



Это *первый* способ представления переменной.

Его преимущество — возможность доступа к любому значению величины в любой момент времени. Недостаток — большой расход памяти при большом числе значений переменной.

Внимание! Наличие в алгоритме переменной с индексом означает, что ее значения представлены в виде массива и каждое такое значение будет храниться в ЭВМ в отдельной ячейке памяти. С учетом сказанного следует очень осторожно переносить математические выражения, содержащие индексы, в алгоритм, поскольку в математике индексы играют иную роль, чем в программировании.

Допустим далее, что значения величины X используются последовательно по мере их возрастания. Тогда все они, от 3 до 18 могут быть вычислены последовательно, если включить в алгоритм формулу[1]

$$X_{i=1} := X + 3. \qquad (10.2)$$

[1] Подобного вида формулы, выражающие каждый член последовательности через p предшествующих, называются в математике рекуррентными. В данном случае p = 1.

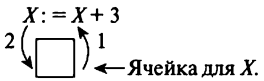
Здесь значения X будут восприниматься как элементы массива, так как переменная X — с индексом. Чтобы избежать этого, выражение (10.2) записывают в алгоритме так:

$$X := X + 3.$$

В общем случае при последовательном использовании значений переменной любое выражение вида $X_{i+1} = f(x_i)$ можно преобразовать в алгоритме к такому:

$$X := f_{(X)}.$$

Здесь X — простая переменная. При этом символ «X», стоящий справа от знака «:=», означает предшествующее, а тот же символ слева от знака «:=» — последующее значение переменной X. В этом случае для переменной X выделяется в ЭВМ только одна ячейка и все значениях последовательно помещаются в нее, причем каждое новое значение будет стирать предшествующее:



При этом должно быть задано начальное значение переменной, а также либо конечное ее значение, либо число ее значений.

Это *второй способ* представления переменной. При этом способе требуемый объем памяти много меньше, чем в первом случае, однако второй способ применим лишь тогда, когда существует закономерность изменения значений переменной и эти значения используются строго последовательно.

Итак, рассмотрен очень важный вопрос, поскольку в любой задаче в первую очередь нужно для каждой используемой величины осознанно выбирать один из двух способов представления ее, наиболее целесообразный.

Теперь уточним термин «исходные данные». С учетом сказанного в этом параграфе будем четко различать два понятия:

данные задачи — все величины, необходимые для решения задачи, указанные в ее формулировке;

исходные данные для алгоритма (*программы*) — величины, значения которых должны быть заданы перечислением их. В дальнейшем ради краткости термин «исходные данные» будем использовать именно в этом смысле.

Так, в задаче: «вычислить произведение целых четных чисел от M до N» данными задачи будут числа M, $M + 2$, $M + 4$, ..., N. Однако в список исходных данных для алгоритма (программы) следует вклю-

чать лишь M и N, так как все прочие значения можно вычислить, зная M и N, в процессе решения задачи.

В частном случае данные задачи и данные для алгоритма могут совпадать.

Понятие циклического алгоритма. Проиллюстрируем это понятие конкретным примером.

Задача 10.16

Вычислить значения функции $Y = X^3 + BX - C$ при $X = 2, 4, 6$.

Решение

В соответствии с изложенным выше все значения X можно вычислить по формуле

$$X := X + 2,$$

возложив эту обязанность на ЭВМ.

В таком случае исходными данными будут только величины[1] B и C. Решая задачу, получим три значения величины Y, выдаваемые последовательно.

Порядок решения задачи можно описать следующим линейным алгоритмом:

1) $X := 2$;
2) $Y := X^3 + BX - C$;
3) Вывод Y;
4) $X := X + 2$;
5) $Y := X^3 + BX - C$;
6) Вывод Y;
7) $X := X + 2$;
8) $Y := X^3 + BX - C$;
9) Вывод Y;
10) $X := X + 2$;
11) Останов.

Однако эта запись весьма громоздка, а если представить, что X изменяется до 2000 (или до 20 000!), то и вообще окажется неприемлемой (описать 1000 (или 10 000) этапов!).

Можно заметить, что в алгоритме операции этапов 2—4 фактически повторяются три раза без каких-либо изменений, поэтому запишем алгоритм в таком виде:

1) $X := 2$;
2) $Y := X^3 + BX - C$;
3) Вывод Y;

[1] Здесь данными задачи являются величины X, B, C, но исходными данными для алгоритма будут лишь B, C.

4) $X := X + 2$;

5) Перейти к п. 2.

В этом алгоритме этапы 2—5 выполняются многократно благодаря операции пятого этапа. Это и есть описание алгоритма, называемого *циклическим*. Недостаток этого описания — алгоритм будет выполняться бесконечно, хотя по условию задачи он должен выполняться до тех пор, пока $X < 6$. Проверкой такого условия и дополним его. Тогда получим:

1) $X := 2$.

2) $Y := X^3 + BX - C$.

3) Вывод Y.

4) Проверить $X < 6$ → *Да* — перейти к п. 5; *Нет* — перейти к п. 7.

5) $X := X + 2$.

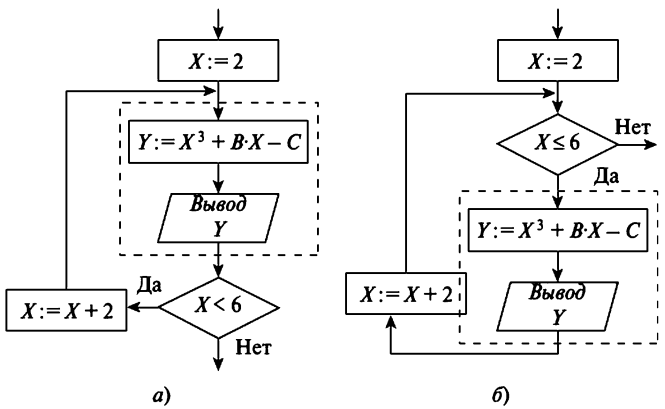
6) Перейти к п. 2.

7) Останов.

Мы получили пример циклического алгоритма в законченном виде. Его схема изображена на рис. 10.27, *а*.

Проверку условия $X \le 6$ можно поместить и перед п. 2 алгоритма. Соответствующая такому случаю схема алгоритма изображена на рис. 10.27, *б*.

Схемы рис. 10.27, *а*, *б* содержат по четыре полностью аналогичных блока (различаются лишь условия в логическом блоке).



а) *б*)

Рис. 10.27

Подобные четыре блока содержатся практически в любом циклическом алгоритме.

Определение и типовая схема циклического алгоритма. *Циклическим* называют алгоритм, в котором получение результата обеспечивается многократным выполнением одних и тех же операций.

Структуру любого циклического алгоритма можно описать схемами, которые будем называть типовыми (рис. 10.28).

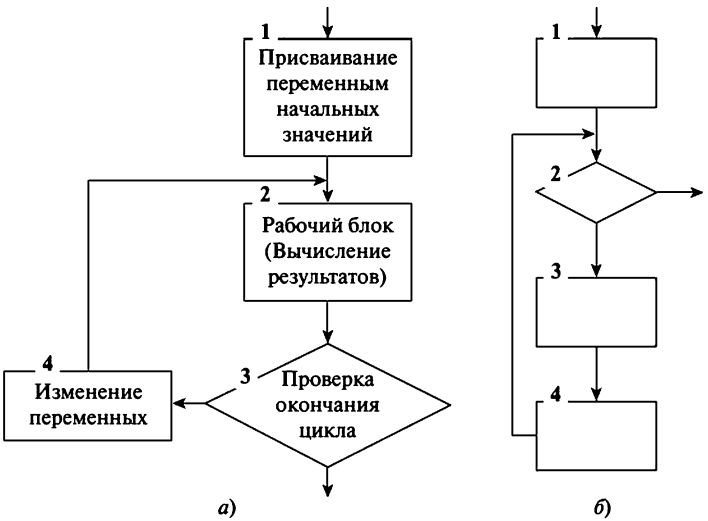


Рис. 10.28

В принципе возможны еще два варианта подобных схем. Они содержат те же четыре блока, что и схемы на рис. 10.28, расположенные, правда, в ином порядке.

С точки зрения принятого ниже подхода к составлению циклического алгоритма выбор типовой схемы совершенно не принципиален. Если вид схемы роли не играет, то целесообразнее использовать схему вида рис. 10.28, *а* и только потому, что она обеспечивает бóльшую наглядность схем алгоритмов сложных задач (алгоритмов с несколькими вложенными циклами), рассматриваемых ниже.

Назначение блоков циклического алгоритма. Вернемся к определению циклического алгоритма. Из него следует, что основой такого алгоритма являются операции, многократное выполнение которых дает искомый результат. Такие операции будем называть *рабочими*. Так же будем называть и описывающие эти операции формулы, т.е. формулы, которые выражают зависимость результатов рабочих операций от некоторых переменных (аргументов). Эти формулы содержатся в рабочем блоке циклического алгоритма (блок 2), а блок 4 содержит законы изменения аргументов.

Так, в задаче 10.16 рабочая формула

$$Y := X^3 + BX - C.$$

Закон изменения аргумента

$$X := X + 2.$$

Как видно из схем рис. 10.28, *а*, *б*, выполнение циклического алгоритма распадается на этапы — их называют *циклами*, на каждом из которых выполняются операции блоков 2, 3, 4, т.е. на каждом цикле вычисляются новые значения аргументов (в блоке 4) и соответствующие им новые значения результатов рабочих операций (в блоке 2) (результатов промежуточных или окончательных).

В этом суть циклического алгоритма!

Блоки 1 и 3 (рис. 10.28, *а*) обычно задают диапазон изменения аргументов. Так, в задаче 10.16 подобные блоки обеспечивают изменение X от 3 до 6.

Блок 1 присваивает всем аргументам начальные значения, обеспечивая тем самым выполнение рабочих операций на первом цикле.

В блоке 3 проверяется условие окончания (повторения) цикла, чаще всего предполагающее сравнение текущего значения некоторой переменной с ее предельным значением. Как, например, в блоке 3 схемы рис. 10.27.

В общем случае условие окончания (повторения) цикла может быть явно не связано ни с аргументами, ни с их конечными значениями (как в солдатской шутке — распоряжение — «копать траншею от забора и до обеда!»).

Вывод. Любой циклический алгоритм определяется тремя наборами формул (содержащихся в блоках 1, 2, 4) и одним условием (отношением, логическим выражением).

Так, алгоритм вычисления суммы элементов массива $P(1{:}n)$ определяется такими формулами:

рабочая формула (аргументы S и i) — $S = S + p_i$;

начальные значения аргументов — $S = 0$; $i = 1$;

закон изменения аргумента i — $i = i + 1$.

Условие повторения цикла:

$i < n$ — для схемы типа рис. 10.28, *а*;

$i \le n$ — для схемы типа рис. 10.28, *б*.

Из сказанного вытекает суть процесса построения циклического алгоритма (с одним циклом).

1. Вывести три набора формул и условие окончания (повторения) цикла, в том числе:
   - рабочие формулы;
   - законы изменения аргументов;
   - формулы для вычисления начальных значений аргументов.
2. Изобразить типовую схему алгоритма с одним циклом и разместить в ней все указанные формулы и условие.

Основная сложность в этом процессе — вывод формул.

В частном случае все эти формулы и условие могут быть заданы в формулировке задачи, тогда сразу начинаем с п. 2.

10.4.2. Вывод алгоритмов с одним циклом. Методика

Вывод формул для циклического алгоритма

Рассмотрим этот процесс на примере решения конкретной задачи.

Задача 10.17

Задан массив $A(1{:}n)$ (значение n — четное число). Вычислить сумму элементов этого массива с четными номерами.

Исходные данные: $A(1{:}n) = \{a_1, a_2, a_3, ..., a_n\}$.

Результат: S.

Опишем последовательность этапов решения задачи, причем тех и только тех, которые выполнялись бы при решении ее вручную.

Этап 1. Вычислим $S_1 = a_2 + a_4$. (10.3)

Этап 2. Вычислим $S_2 = S_1 + a_6$. (10.4)

Этап 3. Вычислим $S_3 = S_2 + a_8$. (10.5)

Этап 4. Вычислим $S_4 = S_3 + a_{10}$. (10.6)

.....

Этап r. Вычислим $S_r = S_{r-1} + a_n$, (10.7)

где r — номер последнего этапа, конкретное его значение пока не известно.

Далее введем переменную i, выражающую в общем случае номер этапа, т.е. $i = 1, 2,.., r$.

Анализируя формулы (10.3)—(10.6), несложно подметить связь значений переменных величин на каждом этапе с номером этого этапа. Выразим в общем случае зависимость их от величины i как операцию на этапе с номером i в таком виде:

Этап i. Вычислим $S_i = S_{i-1} + a_{2+2i}$. (10.8)

Приведем формулы всех этапов к одному виду, виду формул i-го этапа. Рассматривая формулы (10.3)—(10.7), убеждаемся в том, что

в изменении нуждается только выражение (10.3). В нем первое слагаемое отличается от первого слагаемого прочих формул не только значением индекса.

Заменим (10.3) формулой (10.9), полученной из (10.8) подстановкой вместо i единицы:

$$S_1 = S_0 + a_4. \tag{10.9}$$

Такая замена возможна, если до начала вычислений принять $S_0 = a_2$. Тем самым мы нашли начальное значение (a_2) величины S.

Теперь можно утверждать, что вместо всего множества формул: (10.3), (10.4), (10.5), ... — для решения задачи достаточно использовать многократно одну — (10.8), последовательно изменяя в ней значения i от 1 до r с шагом 1 и производя вычисления по этой формуле при каждом значении i. Таким образом, (10.8) и *есть рабочая* формула циклического алгоритма решения задачи 10.17 при начальных значениях переменных[1]

$$S_0 = \underline{a_2} \quad \text{и} \quad i = 1.$$

Определим число этапов, т.е. значение r. Из формул (10.3)—(10.7) следует, что с ростом номера этапа i индекс величины a монотонно возрастает, не превышая значения n [см. (10.7)], т.е. справедливо выражение

$$2 + 2i \le n, \quad \text{т.е.} \quad i \le (n - 2)/2. \tag{10.10}$$

Выражение (10.10) и есть условие повторения цикла.

Формулу (10.8) следует упростить, опустив индексы при S, так как иначе величина S в алгоритме (программе) будет рассматриваться как массив, в чем нет необходимости. Действительно, все значения S используются строго последовательно — на каждом этапе нужно знать лишь одно значение, полученное на предшествующем этапе, и нет нужды сохранять в ЭВМ все значения S, т.е. следует выбрать 2-й способ представления переменной.

Итак, все составляющие циклического алгоритма имеются.

Разместим эти величины в типовой схеме алгоритма с одним циклом в соответствии со стрелками (рис. 10.29), запишем в блок 2-ю рабочую формулу. Схема готова!

[1] Далее можно без особого труда, используя известный принцип математической индукции, строго доказать, что формула (10.8) верна при всех $i > 1$. Предоставляем такую возможность читателю.

Рис. 10.29

Приведенные рассуждения служат некоторым обоснованием излагаемой ниже методики.

Методика составления алгоритмов с одним циклом

1. Формулируем условие задачи.
2. Выявляем данные задачи, исходные данные для алгоритма. При этом руководствуемся правилом: если данные задачи содержат множество значений некоторой величины и закон изменения их неизвестен, то представляем их в виде массива и включаем в список исходных данных. В противном случае массив можно не создавать и величину в исходные данные не включать.
3. Выделяем результаты, т.е.
   - выявляем все вычисляемые величины и назначаем им имена;
   - определяем количество значений каждой такой величины;
   - при необходимости сохранить в ЭВМ значения величины (запомнить их) образуем массив.
4. Выбираем метод решения задачи и разбиваем его на этапы с равным числом аналогичных операций на каждом.
5. Выводим рабочие формулы, для чего:
   а) записываем операции первых n этапов решения задачи ($n = 2, 3, 4, ...$) в виде последовательности равного числа подобных формул на каждом этапе. Значение n выбирается таким, которое позволит подметить закономерность изменения всех переменных.

   Результат выполнения одной и той же операции на разных этапах обозначаем одним именем с индексом, равным номеру этапа;

б) записываем операции, выполняемые на этапе с номером i, как результат обобщения операции п. 5 *а*, т.е. для каждой переменной величины выводим формулу, выражающую ее зависимость *либо от номера этапа i, либо от ее предшествующего значения* (*рекуррентную формулу*);

в) записываем операции последнего этапа, если они известны.

6. Выводим начальные значения переменных, для чего операции всех этапов приводим к виду формул i-го этапа.
7. Выявляем условие окончания (повторения) цикла (если оно не задано в явном виде) как логическое выражение, связывающее обычно текущее значение одной или нескольких переменных с их предельными значениями (достигаемыми на последнем этапе решения задачи).
8. Для каждой переменной, используемой на i-м этапе, выписываем начальное и конечное значения, а также закон ее изменения[1]. Для каждого массива, используемого в задаче, должна быть введена переменная, изображающая индекс его элементов.
9. Проверяем правильность вывода формул i-то этапа. Для этого подставляем в указанные формулы значения $i = 1, 2, 3$, а также последнее (предельное) значение i, и убеждаемся, что полученные формулы соответствуют операциям первых трех и последнего этапов.
10. Изображаем типовую схему циклического алгоритма и заполняем каждый блок в соответствии с его назначением.

В частности, в блок 2 записываем все формулы, полученные на этапе с номером i либо заданные в условии задачи формулы для вычисления результатов; в блок 4 — законы изменения переменных, не включенные в блок 2, и т.д.

Внимание! При заполнении схемы алгоритма индексы сохраняются лишь для величин, представляемых в виде массива.

11. Организуем вывод результатов: если результатом является массив, элементы которого вычисляются последовательно (например, по одному элементу, по одному столбцу и т.д. в каждом цикле), то можно выводить (печатать) в каждом цикле элементы, вычисляемые в этом цикле. Но можно выводить (печатать) весь массив после окончания работы циклического алгоритма (программы). Первый подход позволяет создавать более компактные алгоритмы (программы).

[1] Эти три объекта далее для краткости будем называть (условно) одним термином — «параметры переменной».

Примечания

1. Невозможность выполнения операции пункта 5 б означает, что тот метод (порядок) решения задачи, который был намечен, не позволяет нам построить циклический алгоритм, следует вернуться к п. 4 и искать иной метод решения нашей задачи либо строить алгоритм иного вида.

2. Некоторые составляющие циклического алгоритма могут быть заданы в условии задачи в явном виде, тогда соответствующие пункты методики пропускаются. Так, если в условии задачи заданы рабочие формулы, то п. 5 можно опустить.

10.4.3. Вывод алгоритмов с одним циклом. Примеры

Задача 10.18

Задана последовательность чисел: 3, 5, 6, 8, 9, 11, 15, 16, 20, 21.

Вычислить суммы пар чисел: первого и десятого, второго и девятого, ..., пятого и шестого.

Решение. 1.

Здесь данные задачи — указанные числа. Закономерность в изменении их отсутствует, поэтому, согласно п. 2 методики, представим эти числа в виде массива $X(1{:}10)$ (здесь и далее имена для величин выбираем произвольно).

В результате решения задачи получим пять чисел, которые также представим как массив с именем, например, S, т.е.

исходные данные: $X(1{:}10)$,

результат: $5(1{:}5)$.

2. Метод решения задачи. Суммируем попарно элементы массива X в порядке, указанном в условии задачи: x_1 и x_{10}, x_2 и x_9, и т.д.

3. В соответствии с п. 5 методики запишем операции, выполняемые на первом, втором и третьем этапах решения задачи, и, исходя из них, выявим операцию i-го этапа. Затем запишем операцию последнего этапа.

Этап 1.	$s_1 = x_1 + x_{10}$.	
Этап 2.	$s_2 = x_2 + x_9$.	
Этап 3.	$s_3 = c_3 + x_8$.	
Этап i.	$s_i = x_i + x_{11-I}$	(8.11)
Этап 5.	$S_5 = x_5 + x_6$.	
	(*последний*)	

4. Формула (10.11) и есть рабочая формула. В ней единственная независимая переменная — величина i. Согласно требованиям п. 10 методики выявим все параметры этой переменной: начальное значе-

ние — 1, конечное значение — 5, закон изменения — $i := i + 1$. Для компактности запишем их в таком виде:

$$i:=\begin{vmatrix}1 — \text{начальное значение.}\\ i+1 — \text{закон изменения,}\\ 5 — \text{конечное значение.}\end{vmatrix}$$

Так же будем записывать параметры переменных и в дальнейшем.

5. Изобразим типовую схему алгоритма (см. рис. 10.28, *а*) и заполним ее согласно п. 10 методики, т.е. в типовой схеме размещаем формулу i-го этапа и все параметры переменной i.

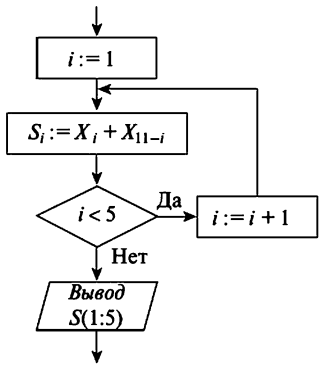


Рис. 10.30

Полученная схема приведена на рис. 10.30.

В этой схеме блоки *Начало* и *Останов* не изображены. Не будем изображать их (а иногда и блоки *Ввод* и *Вывод*) и в прочих схемах этого параграфа.

Задача 10.19

Вычислить произведение целых нечетных чисел от n до m ($m < n$).

Решение

1. Здесь данными задачи являются величины m, $m + 2$, $m + 4$, ..., n. Их можно вычислить, зная m и n. Поэтому исходные данные (для алгоритма) — m и n, результат — R.

2. Метод решения задачи. Последовательно перемножаем m на $m + 2$, полученный результат — на $m + 4$ и т.д.

3. В соответствии с п. 5 методики разобьем указанный процесс на этапы, запишем операции первого, второго и третьего этапов и, исходя из них, выведем и представим операцию, выполняемую на i-м этапе. Запишем операцию последнего этапа.

Этап 1. $p_1 = m(m + 2)$. (10.12)

Этап 2. $p_2 = p_1(m + 4)$.

Этап 3. $p_3 = p_2(m + 6)$.

Этап 4. $p_i = p_{i-1}(m + 2i)$. (10.13)

Этап 5. $p_r = p_{r-1}n$.
(*последний*)

Формула (10.13) есть рабочая формула создаваемого циклического алгоритма, в котором задействованы переменные i и p.

4. Найдем начальное значение переменной p, для чего, согласно п. 6 методики, «формулы, используемые на каждом этапе, приводим к виду формул i-го этапа».

Выполним эту процедуру.

Формула, используемая на любом этапе нашего циклического вычислительного процесса, должна получаться подстановкой в (10.13) номера этого этапа в качестве значения i. От этого правила отклоняется лишь формула первого этапа (10.12), поэтому заменим ее такой:

$$p = p_0(m + 2). \qquad (10.14)$$

При этой замене не должно измениться значение p_1 а для этого в свою очередь до начала вычислений следует принять $p_0 = m$. Это и есть начальное значение переменной p_0 — то, к чему мы стремились!

Должно быть понятно, что формула (10.14) получена подстановкой в (10.13) значения $i = 1$.

В последующих задачах не будем так подробно описывать этот процесс, будем лишь сообщать, какие формулы чем заменяются на первом этапе, а также начальные значения переменных.

5. Выполним требования п. 7 методики — сопоставим операции i-го и r-го этапов, из чего следует условие повторения цикла: $m + 2i \le n$, т.е. $i \le (n - m)/2$.

6. Выделим и запишем параметры переменныхр и i:

$$p := \begin{vmatrix} m \\ p(m+2i); \\ ? \end{vmatrix} \qquad i := \begin{vmatrix} 1 \\ i+1 \\ (n-m)/2. \end{vmatrix}$$

Здесь индексы при p мы опустили, так как для вычисления любого последующего значения переменной p достаточно знать лишь ее предшествующее значение и представлять все значения p в виде массива нет необходимости (см. 10.4.1).

7. Изобразим типовую схему алгоритма с одним циклом (см. рис. 10.28, *а*) и заполним ее согласно п. 12 методики. Получим схему, представленную на рис. 10.31.

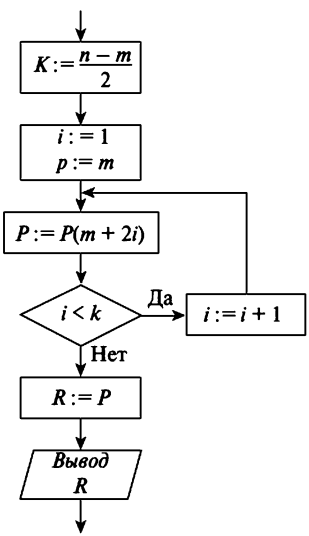


Рис. 10.31

Задача 10.20

Вычислить значение многочлена $z = 3x^5 - x^4 + 6x^3 - 2x^2 + 7x + 5$ при одном значении x.

Решение

1. Исходными данными будут x и коэффициенты многочлена. Последние представим в виде массива $A(0:5)$: $A(0:5)$ $(a_0, a_1, a_2, \ldots a_5)$.

Результат: z, т.е.

$$z = a_5 + a_4x + a_3x^2 + a_2x^3 + a_1x^4 + a_0x^5. \qquad (10.15)$$

2. Метод решения задачи[1]. Последовательно вычисляем слагаемые формулы (10.15) в порядке возрастания степени x и по мере вычисления суммируем каждое из них с предшествующей суммой.

Пояснение. Решение задачи заключается в поочередном выполнении операций — возведения в степень, умножения и сложения. Поэтому разбиение этого процесса на этапы (в отличие от задач 10.18 и 10.19) неоднозначно. «Делить» на этапы следует с таким расчетом, чтобы на каждом из них выполнялся один и тот же набор операций в одном и том же порядке. Весь набор операций на каждом этапе должен выражаться равным числом подобных формул.

В нашем случае на каждом этапе следует выполнять три указанные выше операции в том же порядке. Число формул, описывающих эти операции, может быть 3 и 2 и 1. Ограничимся двумя.

3. Опишем этапы решения задачи:

Этап 1.		$b_1 = a_4x + a_5.$
Этап 2.	$r_2 = x \cdot x;$	$b_2 = a_3r_2 + b_1$
Этап 3.	$r_3 = r_2x;$	$b_3 = a_2r_3 + b_2.$
Этап 4.	$r_4 = r_3x;$	$b_4 = a_1r_4 + b_3.$
Этап i.	$r_i = r_{i-1}x;$	$b_i = a_{5-i}r_i + b_{i-1}.$
Этап k. (*последний*)	$r_k = r_{k-1}x;$	$b_k = a_0r_k + b_{k-1}.$

4. Приведем полученные формулы к виду формул этапа i, т.е. первую операцию этапа 2 запишем так: $r_2 = r_2x$; операции этапа 1 дополним еще одной и запишем в таком виде:

$$r_1 = r_0x; \quad b_1 = a_4r_1 + b_0.$$

Из этого следует — $r_0 = 1$; $b_0 = a_5$ (т. е. выявлены начальные значения величин r и b).

Из сравнения индексов при переменной а на этапах i и k вытекает условие повторения цикла: $5 - i \geq 0$, т.е. $i \leq 5$.

[1] Здесь следовало бы использовать схему Горнера. В этом случае процесс построения алгоритма был бы более простым, но менее поучительным.

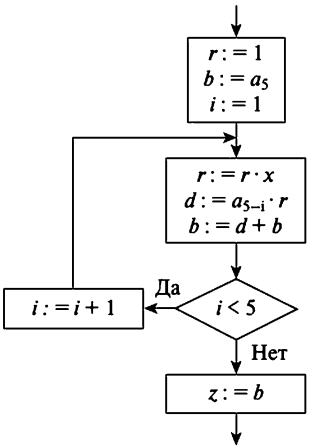


Рис. 10.32

5. Запишем параметры переменных i, r, b (индексы r и b опускаем — здесь тот же случай, что и с переменной p в задаче 10.19):

$$i := \begin{vmatrix} 1 \\ i+1; \\ 5 \end{vmatrix} \quad r := \begin{vmatrix} 1 \\ rx; \\ ? \end{vmatrix} \quad b := \begin{vmatrix} a_5 \\ a_5 r + b. \\ ? \end{vmatrix}$$

6. Изобразим типовую схему алгоритма, представленную на рис. 10.28, *а*, и заполним ее. Получим схему, представленную на рис. 10.32.

Задача 10.21.

Дана матрица $A(1{:}n,\ 1{:}n)$. Найти значение наибольшего элемента побочной диагонали.

Решение

1. Исходные данные: n, матрица $A(1{:}n,\ 1{:}n)$.

Результат: Y.

2. Метод решения задачи. Оперируем с элементами диагонали $a_{1,n}$; $a_{2,n-1}$; $a_{3,n-2}$; $a_{n,1}$. Сравниваем первые два элемента и находим наибольший из них, результат сравниваем с третьим элементом и находим наибольший, и т.д.

3. Выделим и опишем этапы решения задачи.

Этап 1. Проверить условие $a_{1,n} > a_{2,n-1}$, если оно выполняется (*Да*), то результат (обозначим его b_1) равен $a_{1,n}$, иначе (*Нет*) $b_1 = a_{2,n-1}$.

Этап 2. Проверить $b_1 > a_{3,n-2}$,
Да — $b_2 = b_1$;
Нет — $b_2 = a_{3,n-2}$.

Этап 3. Проверить $b_2 > a_{4,n-3}$,
Да — $b_3 = b_2$;
Нет — $b_3 = a_{4,n-3}$.

Этап i. Проверить $b_{i-1} > a_{i+1,n-i}$,
Да — $b_i = b_{i-1}$;
Нет — $b_i = a_{i+1,n-i}$.

Этап r. Проверить $b_{r-1} > a_{n,1}$,
Да — $b_r = b_{r-1}$;
Нет — $b_r = a_{n,1}$.

(последний)

4. Операции первого этапа приведем к виду операций этапа i, т.е. к такому виду:

проверить $b_0 > a_{2,n-1}$
Да — $b_1 = b_0$
Нет — $b_1 = a_{2,n-1}$.

Из этого следует, что значение b_0 должно быть равно $a_{1,n}$.

5. Из сравнения индексов при переменной a на i-м и r-м этапах вытекает условие повторения цикла: $i+1 \le n$, т.е. $i \le n-1$.

6. Запишем параметры переменной i. Для переменной b приведем лишь начальное значение (так же будем поступать при выполнении этого пункта методики в последующих задачах с переменными, формулы вычисления которых являются рабочими), т.е.

$$i := \begin{vmatrix} 1 \\ i+1; b = a_{1,n} \text{ (начальное значение переменной } b) \\ n-1 \end{vmatrix}.$$

Индексы при b мы опустили — здесь опять тот же случай, что и с переменной p в задаче 10.19.

7. Изобразим типовую схему циклического алгоритма рис. 10.28, *а* и заполним ее. В рабочий блок включим все три операции i-го этапа, в том числе и проверку условия $b_{i-1} > a_{i+1,\,n-i}$, которую изобразим в виде отдельного логического блока, как того требуют правила построения схем алгоритмов.

Операцию $b_i = b_{i-1}$ опустим.

Полученная схема алгоритма представлена на рис. 10.33.

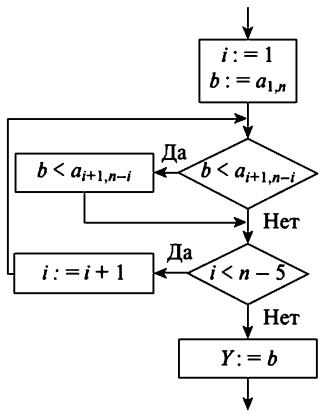


Рис. 10.33

Задача 10.22

Задана последовательность из n чисел:

$$3, 4, 7, 8, -9, 12, \ldots .$$

Определить сумму положительных элементов этой последовательности.

Решение

1. Здесь данные задачи — все n чисел. Закономерность изменения их отсутствует, поэтому представим числа в виде массива, т.е. исходные данные: n, $R(1{:}\,n)$; результат: s.

2. Метод решения задачи. Каждый элемент массива, начиная с первого, сравниваем с нулем, и если он больше нуля, то суммируем его с суммой предшествующих элементов, иначе — переходим к следующему элементу.

3. Выделим и опишем этапы решения задачи. Исходя из опыта решения предшествующих задач, выберем для переменной s — текущего значения суммы — второй способ представления, т.е. индексы при s опускаем:

Этап 1. Проверить $r_1 > 0$	*Да* — $s := r_1$;
	Нет — перейти к следующему этапу.
Этап 2. Проверить $r_2 > 0$,	*Да* — $s := r_2 + s$;
	Нет — перейти к следующему этапу.
Этап 3. Проверить $r_i > 0$,	*Да* — $s := r_3 + s$;
	Нет — перейти к следующему этапу.
Этап i. Проверить $r_1 > 0_i$,	*Да* — $s := r_i + s$;
	Нет — перейти к следующему этапу.

4. Операции первого этапа запишем таким образом:

Проверить $r_1 > 0$	*Да* — $s := r_1 + s$;
	Нет — перейти к следующему этапу.

Отсюда следует — начальное значение s должно быть равно нулю.

5. Запишем параметры переменных i и s:

$$i := \begin{vmatrix} 1 \\ i+1; s = 0 \text{(начальное значение переменной } s\text{)} \\ n \end{vmatrix}.$$

6. Разместим все формулы i-го этапа и параметры переменных в типовой схеме циклического алгоритма, представленной на рис. 10.28, *а*. Получим схему алгоритма задачи (рис. 10.34).

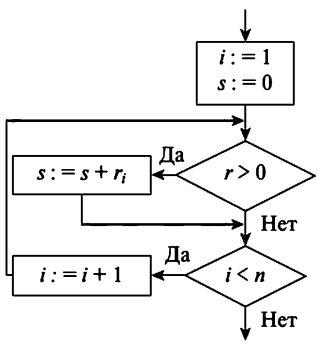


Рис. 10.34

Задача 10.23.

Вычислить приближенное значение функции $y = a^x$ при $x < 0$ и $|x| \le 1$ с помощью разложения ее в ряд:

$$y = 1 + \frac{\ln a}{1}x + \frac{\ln^2 a}{2!}x^2 + \frac{\ln^3 a}{3!}x^3 + \ldots \qquad (10.16)$$

Допустимая погрешность вычисления — ε.

Ряд (10.16) сходящийся. При $x < 0$ он становится знакочередующимся. В таком случае, как известно, за приближенное значение y можно принять сумму n первых членов ряда. Погрешность такого приближения δ не превышает абсолютного значения $(n+1)$-го (а тем более n-го) члена ряда.

Решение.

1. Исходные данные: x, a, ε.

Результат: y.

2. Метод решения задачи (в общем виде). Проверить условие: $x < 0$, если *Да* — вычислить y по формуле (10.16); вывод y; останов, если *Нет* — останов.

Этот процесс изобразим в виде укрупненной схемы (рис. 10.35).

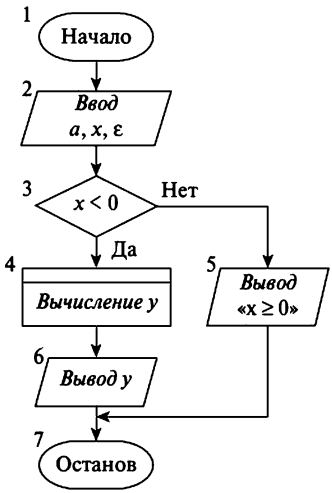


Рис. 10.35

Пояснение. Схема рис. 10.35 содержит блок 5, не предусмотренный условием задачи. Мы включили его, руководствуясь правилом: «при любом варианте решения задачи алгоритм (программа) должен выдавать сообщение о результатах решения, даже если их нет» (см. 10.1.3).

Далее рассмотрим решение задачи блока 4 схемы, представленной на рис. 10.35.

3. Метод решения задачи блока 4. Последовательно вычислять значение каждого члена ряда, начиная с первого, и суммировать его с предшествующей суммой, пока не выполнится условие — $\delta < \varepsilon$.

Введем обозначение: $c = \ln a$.

4. Выделим и опишем этапы решения задачи.

Этап 1.	$b_1 = cx;$	$y_1 = 1 + b_1.$
Этап 2. $r_2 = 1 \cdot 2;$	$b_2 = b_1 cx;$	$y_2 = y_1 + b_2/r_2.$
Этап 3. $r_3 = r_2 \cdot 3;$	$b_3 = b_2 cx;$	$y_3 = y_2 + b_3/r_3.$
Этап i. $r_i = r_{i-1} \cdot i;$	$b_i = b_{i-1} cx;$	$y_i = y_{i-1} + b_i/r_i.$

Поскольку условие окончания цикла известно: $\delta \le \varepsilon$, операции последнего этапа не записываем. Очевидно, $\delta \le |b_i/r_i|$.

5. Операции первого и второго этапов приведем к виду операций i-го этапа и запишем в таком виде:

$$r_2 = r_1 \cdot 2; \quad r_i = r_0 \cdot 1; \quad b_1 = b_0 cx; \quad y_1 = y_0 + b_1/r_1.$$

Отсюда следует, что начальные значения переменных r, b и y равны 1. Индексы при r, b и y опускаем.

6. Запишем параметры всех переменных:

$$i := \begin{vmatrix} 1 \\ i+1; \\ ? \end{vmatrix} \quad b := \begin{vmatrix} 1 \\ bcx; \\ ? \end{vmatrix} \quad r = 1, y = 1 \text{ (начальные значения переменных).}$$

7. Составим схему алгоритма блока 4 (рис. 10.36). Подробную схему алгоритма всей задачи, надеемся, читатель сможет получить сам, объединив схемы, представленные на рис. 10.35 и 10.36.

Довольно часто встречаются в циклических алгоритмах *переменные, значения которых не зависят от номера этапа.*

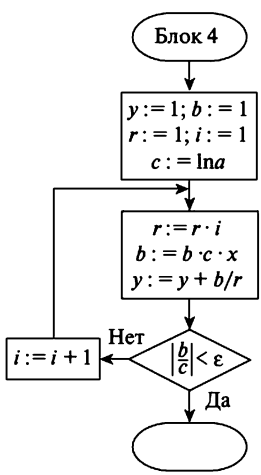


Рис. 10.36

Рассмотрим на примере задачи 10.24 составление алгоритмов в таком случае.

Задача 10.24

Из массива $A(1{:}n)$ выбрать положительные элементы.

Решение.

1. Исходные данные: n, $A(1{:}n)$.

Результат: массив $B(1{:}n)$. В массив B будем помещать указанные элементы массива A.

Пояснение. Число положительных элементов массива A до решения задачи неизвестно, поэтому «заказан» массив B максимального размера.

2. Метод решения задачи. Каждый элемент массива A по порядку, начиная с первого, сравниваем с нулем, и если его значение

больше нуля, то заносим это значение в массив B, «заполняя» элементы последнего строго по порядку.

3. Опишем порядок решения задачи так, как это выполнило бы большинство учащихся.

Этап 1. Проверить $a_1 > 0$ *Да* — $b_1 := a_1$;
Нет — перейти к следующему этапу.
Этап 2. Проверить $a_2 > 0$, *Да* — $b_2 := a_2$;
Нет — перейти к следующему этапу.
Этап 3. Проверить $a_3 > 0$, *Да* — $b_3 := a_3$;
Нет — перейти к следующему этапу.

Внимательный читатель, наверное, заметил ошибку в этих формулах — значение индекса элемента массива B в действительности не совпадает с номером этапа и, более того, никак не связано с ним.

Так, на этапе 2 индекс может принимать значения 1 или 2, на этапе 3—1, 2 или 3, на этапе 4—1, 2, 3 или 4, и т.д. Поэтому обозначим индекс именем, как переменную величину, например, f.

Далее в соответствии с п. 5 б методики для этой величины нужно вывести рекуррентную формулу, т.е. найти зависимость величины от ее предшествующего значения. В нашем случае подобная формула такова:

$$f := f + 1. \tag{10.17}$$

Очевидно, очередное значение f по этой формуле следует вычислять каждый раз, когда в массиве A выявляется очередной положительный элемент. С учетом сказанного правильный порядок решения задачи можно описать так.

Этап 1. Проверить $a_1 > 0$ *Да* — $f := f + 1$. $b_f := a_1$;
Нет — перейти к следующему этапу.
Этап 2. Проверить $a_2 > 0$, *Да* — $f := f + 1$. $b_f := a_2$;
Нет — перейти к следующему этапу.
Этап 3. Проверить $a_i > 0$, *Да* — $f := f + 1$. $b_f := a_i$;
Нет — перейти к следующему этапу.

Формулу (10.17) мы сразу включили в описание операций первого этапа и тем самым привели их к виду формул i-го этапа. Из анализа операций первого этапа следует, что до начала вычислений нужно принять $f = 0$.

Теперь изобразим схему алгоритма (рис. 10.37).

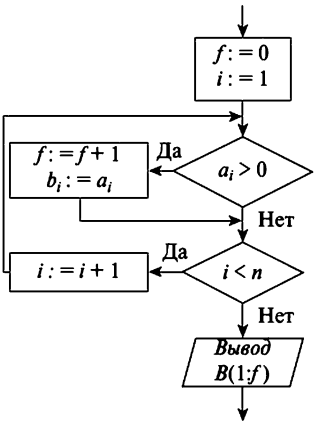


Рис. 10.37

Обобщим сказанное. Если при составлении циклического алгоритма выявляется переменная, значение которой не зависит от номера этапа, но зависит от предшествующего значения этой величины, то следует выполнить несколько дополнительных операций (после описания этапов решения задачи):

1) присвоить указанной переменной имя (например, f);
2) вывести рекуррентную формулу для вычисления этой величины (например, $f := f + 1$);
3) включить полученную формулу в описание процесса решения задачи.

Дальнейшие действия выполняют в соответствии с методикой.

Этот же подход можно использовать и в том случае, когда переменная зависит от номера этапа, но эту зависимость сложно выразить аналитически.

Внимание! В задаче 10.24 выводить (печатать) следует не весь массив, а только те его элементы, которые получили значения в процессе решения задачи, т.е. первые f элементов.

Заметим, что если подходить строго, то выводить массив B можно лишь при $f > 0$, включив в алгоритм проверку этого условия.

10.4.4. Дополнительные сведения по выводу рабочих формул циклического алгоритма

Суть рассматриваемого процесса вывода указанных формул — описание операций 1-го, 2-го, 3-го и т.д. этапов решения задачи, а затем вывод зависимости каждой переменной величины от номера этапа. В приведенных выше примерах вывод формул мы выполняли интуитивно, хотя в некоторых случаях это было сделать весьма не просто.

Покажем некоторый математический подход к решению этой задачи.

1. Пусть некоторая величина X на 1-м, 2-м, 3-м и т.д. i-м этапах решения задачи принимает значения $C_1, C_2, C_3, ..., C_i, ...$. Вывести формулу, выражающую зависимость X от i.

Для нас наиболее существенны два случая: переменная X изменяется с постоянным шагом и с переменным шагом.

Рассмотрим оба случая.

А. Переменная X изменяется с постоянным шагом d, т.е.

$$C_2 = C_1 + d, \quad C_3 = C_2 + d, \quad \text{и т.д.} \quad C_{j+1} = C_i + d,$$

где C_i — константа, $i = 1, 2, 3, \ldots$.

В этом случае переменная X линейно зависит от i и зависимость имеет вид

$$\mathbf{Ai + B},$$

значения коэффициентов A и B вычисляются по формулам

$$A = d, \quad B = C_1 - d.$$

Пример. В задаче 10.18 индекс при втором слагаемом имеет значения 10, 9, 8, 7, ... Т. е. $C_1 = 10$, $C_2 = 9$ и т.д., шаг $d = -1$.

Решение

$$A = d = -1; B = C_1 - d = 10 - (-1) = 11.$$

Выражение для индекса при втором слагаемом должно иметь вид

$$Ai + B = (-1)i + 11 = 11 - i.$$

Б. Переменная X изменяется с переменным шагом, но шаг изменения шага — постоянный.

Например, величина Y при $i = 1, 2, 3, 4$ и т.д. принимает значения 2, 4, 7, 11 и т.д. При этом шаг $d_1 = 4 - 2 = 2$; шаг $d_2 = 7 - 4 = 3$; шаг $d_3 = 11 - 7 = 4$ и т.д., т.е. значение шага — величина переменная и шаг изменения шага $dd = 1$.

Рассмотрим общий случай. Пусть X принимает значения C_1 при $i = 1$, C_2 при $i = 2$, C_3 при $i = 3$ и т.д., и $d_i = C_{i+1} - C_i$, а $dd = d_{i+1} - d_i$ (т. е. шаг изменения шага) есть константа.

В этом случае зависимость переменной X от i имеет вид

$$Ai^2 + Bi + C,$$

где коэффициенты A, B, C вычисляются по формулам

$$A = (C_1 - 2C_2 + C_3)/2,$$
$$B = C_2 - C_1 - 3A,$$
$$C = 4A + C_1 + C_2 - C_3.$$

Приведенные формулы получены на основе интерполяционного полинома Лагранжа (Мантуров О.В. и др. Толковый словарь математических терминов. М., 1965).

Для переменной Y нашего примера:

$$A = (2 - 2 \cdot 4 + 7)/2 = 0{,}5,$$
$$B = 4 - 2 - 3 \cdot 0{,}5 = 0{,}5,$$
$$C = 4 \cdot 0{,}5 + 2 + 4 - 7 = 1,$$

т.е. зависимость Y от i выражается формулой

$$0{,}5i_2 + 0{,}5i + 1.$$

В правильности формулы несложно убедиться, подставив вместо i значения 1, 2, 3, 4 и т.д.

Отметим, что случай Б не редкость при решении реальных задач.

2. Возможен иной подход к выводу формул, основанный на использовании рекуррентных выражений, т.е. выражений вида

$$X_{i+1} = F(X_i).$$

В этом случае на этапе i каждой переменной даем свое имя, выводим соответствующую *рекуррентную формулу* и *начальное значение*. Первые включаем в формулы i-го этапа, вторые помещаем в блок изменения переменных, а третьи — в блок начальных значений схемы алгоритма. Все!

Продемонстрируем этот подход на примере решения следующей задачи: в массиве $X(1{:}n)$ вычислить суммы троек элементов: 1—3, 4—6, 7—9 и т.д., n — кратно 3. Результат: $S(1{:}n/3)$.

Этап 1. $S_1 = X_1 + X_2 + X_3.$
Этап 2. $S_2 = X_4 + X_5 + X_6.$
Этап 3. $S_3 = X_7 + X_8 + X_9.$
Этап i. $S_i = X_k + X_l, + X_m.$

Т.е. переменные величины — индексы при X, обозначаем именами, не связывая их значения с номером этапа i. Далее выводим для всех переменных рекуррентные выражения, предельно простые:

$$i := \begin{vmatrix} 1 \\ i+1 \\ n/3 \end{vmatrix} \quad k := \begin{vmatrix} 1 \\ k+3; \\ ? \end{vmatrix} \quad l := \begin{vmatrix} 2 \\ l+3; \\ ? \end{vmatrix} \quad m := \begin{vmatrix} 3 \\ m+3. \\ ? \end{vmatrix}$$

Схема алгоритма приведена на рис. 10.38.

Реализация цикла на Бейсике в этом случае потребует девять операторов.

Если бы мы выводили зависимость переменных от номера этапа, то на этапе i получили бы такую формулу:

$$S_i = X_{3i-2} + X_{3i-1} + X_{3i}$$

с единственным аргументом i. Для построения программы цикла на Бейсике в этом случае потребуется лишь три оператора.

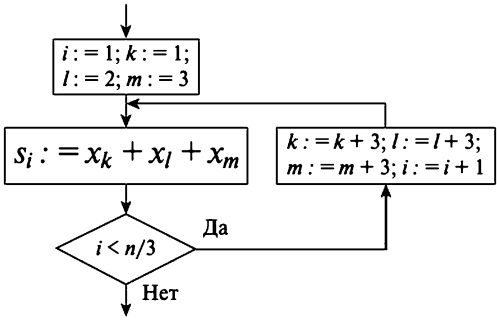


Рис. 10.38

10.4.5. Отладка циклических алгоритмов

Как отмечалось, отладка алгоритма выполняется для проверки его правильности, выявления и исправления ошибок в нем.

Отладка циклического алгоритма не имеет особой специфики (по сравнению с отладкой рассмотренных выше алгоритмов). В этом случае *тест может содержать один набор* данных и выбирается таким образом, чтобы обеспечить проверку выполнения всех блоков и ветвей алгоритма при небольшом числе повторения цикла (три, четыре).

Рассмотрим отладку алгоритма на примере схемы, представленной на рис. 10.33.

А. Выберем в качестве тестового набора такие, например, значения исходных данных: $n = 4$,

$$A = \begin{pmatrix} 1 & 1 & 2 & 4 \\ 1 & 1 & 5 & 6 \\ 1 & 6 & 1 & 1 \\ 3 & 1 & 1 & 1 \end{pmatrix}.$$

Значения элементов побочной диагонали выбраны с таким расчетом, чтобы обеспечить проверку правильности работы обеих ветвей алгоритма в рабочем блоке. Значения остальных элементов роли не играют, и мы выбрали их простейшими.

Б. Выполним ручное решение задачи на тестовом наборе данных — результат очевиден: $Y = 6$.

В. Исполним алгоритм:

Цикл 1	Цикл 2	Цикл 3
1. $i = 1$	7. $b < a_{32}$? $5 < 6$ — Да	11. $b < a_{41}$? $6 < 3$ — Нет
2. $b = a_{14} = 4$		
3. $b < a_{23}$? $4 < 5$ — Да	8. $b = a_{32} = 6$	12. $i < n - 1$? $3 < 3$ — Нет
4. $b = a_{23} = 5$	9. $i < n - 1$? $2 < 3$ — Да	13. $Y = 6$
5. $i < n - 1$? $1 < 3$ — Да	10. $i = 2 + 1 = 3$	14. Останов.
6. $i = 1 + 1 = 2$		

Г. Сравним результаты. Результаты ручного решения задачи и исполнения алгоритма совпали.

Вывод: алгоритм верен.

Контрольные вопросы

1. Какой алгоритм называется циклическим?
2. Какова типовая схема циклического алгоритма? Назначение основных блоков этой схемы?
3. В чем суть методики составления алгоритма с одним циклом?
4. В чем заключаются основные правила выявления исходных данных и результатов при составлении циклического алгоритма?

Задачи для самостоятельного решения

Для каждой из приведенных задач составить схему алгоритма циклической структуры.

1. Поставить элементы главной диагонали, меньшие k, матрицы $A(1{:}n, 1{:}n)$ на место соответствующих элементов l-го столбца.
2. Перемножить соответствующие элементы i-й строки и p-то столбца матрицы $A(1{:}n, 1{:}n)$.
3. Заменить элементы последнего столбца матрицы $A(1{:}4, 1{:}4)$ по порядку, начиная с 1-го, числами 1, 3, 5, 9.
4. Вычислить

$$y = 1 + \frac{x}{2} + \frac{x^2}{4} + \frac{x^3}{6} + \frac{x^4}{8} + \frac{x^5}{10}.$$

5. Вычислить произведения целых чисел от m до n на число k.
6. В массиве $A(1{:}10)$ элементы с номерами 2, 4, 5, 9 разделить на r. Результаты запомнить.

7. Вычислить

$$p = 2\ln|x| \quad \text{и} \quad z = \begin{cases} (b_1 + b_2 - b_3)\dfrac{1}{\sqrt{x}}, & \text{если } p < d; \\ (b_1 - b_3)x, & \text{если } p \geq d, \end{cases}$$

где $x = 3, 9, 12, 13, 15$. Результаты запомнить.

8. Элементы массива $B(1{:}n)$ (n — четное) с четными номерами заменить на 0 (стереть). Вывести весь массив.
9. Переставить значения соответствующих элементов главной и побочной диагоналей матрицы $D(1{:}n,\ 1{:}n)$, сумма которых больше нуля.
10. Сдвинуть элементы массива $B = \{b_1, b_2, ..., b_n\}$, начиная с k-го, на два разряда вправо. Элементам, равным нулю, присвоить значение r.
11. Вычислить скалярное произведение векторов $A = \{a_1, a_2, ..., a_n\}$ и $B = \{b_1, b_2, ..., b_n\}$, т.е. вычислить

$$c = a_1b_1 + a_2b_2 + ..., a_nb_n.$$

12. Заменить в матрице $A(1{:}n,\ 1{:}m)$ элементы k-й строки, равные нулю, элементами массива $B(1{:}m)$ по порядку.

ГЛАВА 11

СТРУКТУРНЫЙ ПОДХОД К ПРОГРАММИРОВАНИЮ

Цель этой главы — дать подход к составлению алгоритмов достаточно сложных задач. Прежде чем перейти к его изложению, необходимо сделать небольшое введение.

Процесс обучения алгоритмизации весьма сложен, в общем случае успех дела во многом зависит от подготовки учащегося и его природных данных. Достаточно сказать, что одна и та же задача для одного начинающего программиста может показаться элементарной, а для другого — непосильной.

Соответственно и материал этой главы кому-то может показаться слишком простым — этим учащимся рекомендуется сразу перейти к задачам для самостоятельного решения.

Другим учащимся материал может показаться слишком сложным, и это может быть отнесено на счет методики как недостаточно удачной. Однако другой методики, достаточно четкой, с такой же степенью универсальности автор не знает и не уверен, что она существует. Взамен предложенной методики можно рекомендовать лишь такой подход, широко распространенный на практике, — рассмотреть три-четыре задачи, достаточно сложные, и ознакомиться со схемой алгоритма для каждой из них. Затем взять условие следующей задачи и нарисовать схему ее алгоритма.

Для тех, кто решил освоить предлагаемую методику составления алгоритмов, подчеркнем, что знание методики составления алгоритмов с одним циклом (см. 10.4.2) совершенно необходимо.

11.1. Основные положения и методика составления алгоритмов

В главе 10 мы рассмотрели построение основных видов алгоритмов: линейных, разветвляющихся, циклических с одним циклом.

В настоящей главе переходим к составлению более сложных алгоритмов. Особенностью таких алгоритмов кроме наличия боль-

шего числа блоков является то, что каждый из них можно разбить на фрагменты, части и эти части могут быть любого из трех рассмотренных выше видов.

При составлении сложных алгоритмов будем использовать подход, являющийся развитием и углублением известного в литературе подхода к программированию — *структурного*. Понять его совершенно необходимо для овладения материалом настоящей главы.

Основные составляющие структурного подхода к программированию:

- нисходящее пошаговое проектирование;
- структурное программирование;
- модульное программирование; структурный контроль.

Основополагающим здесь является первый из указанных принципов. Рассмотрим его наиболее основательно. Сразу отметим — принцип нисходящего пошагового проектирования не является каким-то отвлеченным, искусственным приемом. Человечество и на производстве, и в быту для решения сложных задач многие века и часто бессознательно использует тот же подход.

В главе 10 на примере нескольких задач рассмотрена основная идея этого подхода — использование укрупненных схем алгоритмов (составление плана решения задач). Однако эти задачи были простыми и подход мог показаться искусственным и необязательным.

Далее этот подход рассмотрен обстоятельнее и на более сложных задачах.

Вначале рассмотрим типичную производственную ситуацию, хотя и несколько условную.

Некоторому проектному институту поручено спроектировать жилой район с именем *N*. Главный инженер института как руководитель проекта изучает задачу, уясняет исходные данные и результаты ее и выделяет в ней наиболее крупные подзадачи, части, т.е. решает, например, что район *N* должен состоять из трех микрорайонов (МКР) с именами *A*, *B*, *C*, и намечает порядок их проектирования. Описав этот процесс графически, получим первую схему алгоритма проектирования района *N*.

Далее МКР распределяются между отделами института с номерами 1, 2, 3. Начальник, например отдела № 2, получив задание на проектирование МКР, например *B*, выполняет аналогичные операции со своей задачей и выделяет в ней основные, наиболее крупные составляющие ее. Допустим, этот МКР будет состоять из улиц с именами *У*1, *У*2, *У*3, *У*4. Описав графически порядок проектирования их, получим схему алгоритма проектирования микрорайона *B* района *N*. Пусть теперь каждый блок этой схемы (одна улица МКР) закрепляется

за одной из проектных групп отдела № 2 (2.1, 2.2, 2.3, 2.4). И здесь руководителем каждой группы выделяются наиболее крупные подзадачи в своей задаче — проектирование отдельных кварталов (допустим так) и порядок их выполнения. Каждая группа составляет схему алгоритма проектирования кварталов одной улицы микрорайона *B* района *N*. Аналогично можно составить схему алгоритма проектирования каждого МКР, каждой из улиц каждого МКР, каждого квартала каждой улицы каждого МКР и т.д.

Все полученные схемы в совокупности будут описывать процесс проектирования всего района *N*.

Схем будет много, но они простые, наглядные и удобные для использования.

Если создать сразу общую схему проектирования всего района с детализацией ее, например до уровня отдельных домов, то получим схему, практически необозримую и, главное, возможно, непригодную для работы из-за большого числа ошибок — правильно отразить большое число связей большого числа объектов очень сложно.

Если позволяет время, на каждом этапе проектирования может работать один и тот же коллектив (или даже один человек). Сначала он работает в качестве главного инженера, затем в качестве начальника отделов *1*, *2* и *3* последовательно. Затем в качестве начальника групп каждого отдела и т.д., выдавая на каждом этапе схему алгоритма проектирования определенного объекта. В итоге будет получен тот же результат.

Аналогично должен работать и программист, составляя схему алгоритма решения сложной задачи.

Сформулируем основные особенности рассмотренного процесса.

1. Алгоритм решения задачи составляется за ряд шагов.
2. На первом шаге рассматриваемая задача разбивается на составные части, причем выделяются наиболее крупные ее части и составляется алгоритм, указывающий порядок выполнения этих частей.
3. На втором и последующем шагах в качестве задачи рассматривается некоторая часть исходной задачи, выделенная на каком-то предшествующем шаге, т.е. один из блоков полученных схем алгоритмов, причем рассматривается как самостоятельная задача, изолированно от всех прочих.

Эти положения и составляют суть процесса *нисходящего пошагового проектирования*.

Структурное программирование предполагает составление алгоритма задачи из конструкций строго определенного вида. Основное положение структурного программирования следующее:

— *любой алгоритм может быть представлен комбинацией базовых алгоритмических структур трех видов*: линейной (см. рис. 10.3), разветвляющейся[1], типа изображенной на рис. 10.15, циклической — типа структур, приведенных на рис. 10.28, *а*, *б*.

Причем, подчеркиваем особо, подобная структура (схема) должна быть с одним входом и одним выходом.

Это положение означает, что на *каждом шаге процесса нисходящего проектирования следует составлять алгоритм одного из трех указанных видов*.

Ряд прочих требований структурного программирования определяет непосредственно процесс программирования. Их нет смысла рассматривать на этапе алгоритмизации. С этим этапом не связаны и понятия модульного программирования и структурного контроля. Поэтому в настоящей главе их не будем касаться.

Далее будут использоваться понятия укрупненной и подробной схем алгоритма. Мы их уже использовали в главе 10, но сейчас дадим более четкие их определения. Предварительно напомним, что после составления алгоритма задачи его необходимо перевести на язык ЭВМ, т.е. записать в виде программы на языке программирования. При таком переводе каждый блок схемы алгоритма будет описываться некоторым фрагментом программы более или менее сложным. Сложность его зависит от вида используемого языка.

Элементарным будем называть некоторый блок схемы алгоритма, если выполняемая им операция описывается одним оператором языка программирования.

Далее будем ориентироваться на язык Бейсик и считать элементарным блок, выполняющий следующие операции:

а) вычисление по формуле, в которой указаны действия и величины, допустимые в языке Бейсик. В частности, допустимы все арифметические операции над целыми и действительными величинами, а также элементарные функции — $\sin x$, $\cos x$ и т.д. (см. главы 11 и 13);
б) проверка истинности логического выражения;
в) ввод любого числа величин;
г) вывод (печать) любого количества величин;
д) любая группа операций, если в программе она реализуется в виде подпрограммы.

Теперь можно дать следующие определения.

[1] В литературе в качестве базовой разветвляющейся структуры принят фрагмент, содержащий лишь один логический блок, т.е. частный случай структуры, приведенной на рис. 10.15.

Подробной будем называть схему алгоритма, которая содержит только элементарные блоки.

Укрупненной будем называть схему алгоритма, хотя бы один блок которой не является элементарным, такие блоки назовем *укрупненными*. Каждый из них описывается собственной схемой алгоритма.

Очевидно, схема алгоритма проектирования района, полученная на первом шаге, является укрупненной.

Рассмотрим теперь применение всех изложенных принципов проектирования к нашим задачам, заодно проиллюстрируем на примерах введенные выше понятия — укрупненная, подробная схемы и пр.

Для начала вернемся к более детальному рассмотрению задачи 10.12, 2-го варианта ее решения, базирующегося на излагаемом подходе.

Решение задачи выполнено за три шага.

Шаг 1. Метод решения всей задачи сформулирован в самом общем виде, и выделены наиболее крупные этапы ее решения:

а) определение $Z = \max\{a_{11}, a_{22}\}$;

б) определение $Y = \max\{Z, a_{33}\}$.

В соответствии с этим составлена укрупненная схема алгоритма задачи, указывающая связь этих этапов, блоков 1 и 2 (см. рис. 10.23, *а*).

Каждая из операций этой схемы выполняется только один раз. Такие операции (этапы) будем называть *однократными*.

Шаг 2. Составлена схема алгоритма блока 2 (см. рис. 10.23, *б*) независимо от прочих блоков.

Это *подробная* схема, так как все ее блоки элементарные.

Шаг 3. Составлена схема алгоритма блока 1 (также независимо от прочих блоков).

Объединив схемы блоков 1 и 2 в соответствии с укрупненной схемой, получим *подробную* схему алгоритма всей задачи (см. рис. 10.24). Все операции блоков 1 и 2 также *однократные*.

Решение задачи 10.12 иллюстрирует тот факт, что линейные и разветвляющиеся алгоритмы описывают совокупности однократных операций и наоборот.

Задача 11.1

Вычислить сумму n первых членов ряда:

$$y = 1 = \frac{x}{2} + \frac{x}{3} + \frac{x}{4} + \ldots$$

при $x = 2, 5, 6, 9, 10$.

Шаг 1. Рассматриваем решение всей задачи в целом.

1. Исходные данные: n, массив $X(1:5)$.

Результат: массив $Y(1:5)$.

2. Метод решения задачи (в самом общем виде).

Этап 1. Вычислить первое значение $y/(y_1)$ при первом значении $x(x_1)$.

Этап 2. Вычислить y_2 при $x = x_2$.

Этап i. Вычислить y_i при $x = x_i$.

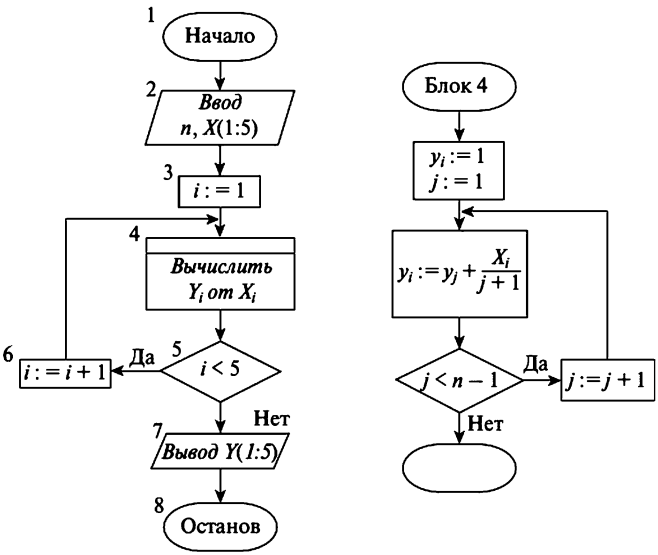
Этап 5. Вычислить y_5 при $x = x_5$

Процесс решения этой задачи заключается фактически в выполнении одной операции 5 раз — вычисление величины y одним и тем же способом при различных значениях одной и той же переменной x.

Подобные операции (этапы) решения задачи будем называть *многократными. Наличие такой операции (этапа) есть признак циклического процесса, описываемого циклическим алгоритмом.*

Соответственно с этим на первом шаге решения задачи получим *укрупненную схему* алгоритма всей задачи циклической структуры (рис. 11.1, *а*). На втором шаге — *подробную схему* той же структуры блока 4 (рис. 11.1, *б*), учитывая, что для этого блока исходные данные: x_i; результат: y_i.

Объединив далее схемы рис. 11.1, *а* и 11.1, *б*, получим подробную схему алгоритма всей задачи (рис. 11.2). Подобного вида алгоритмы называют *алгоритмами с вложенными циклами.*



а) *б)*

Рис. 11.1

Методика составления алгоритмов

1. Четко формулируем условие задачи.

2. Выявляем исходные данные и результаты в соответствии с правилами, сформулированными в п. 2 и 3 методики составления алгоритмов с одним циклом (см. 10.4.2).

3. Формулируем метод решения задачи в наиболее общем виде.

4. Выделяем наиболее крупные этапы решения задачи, выполняющиеся *однократно* и охватывающие в совокупности процесс решения всей задачи:

- при отсутствии таких этапов переходим к п. 5;
- наличии — последовательно описываем их произвольным образом, составляем укрупненную (подробную) линейную или разветвляющуюся схему алгоритма задачи с одним выходом, руководствуясь Основными принципами алгоритмизации, и переходим к п. 7.

5. Выделяем наиболее крупную операцию, выполняющуюся *многократно*, т.е. операцию, многократное выполнение которой обеспечивает решение всей задачи.

6. Составляем укрупненную (подробную) схему алгоритма с одним циклом по известной методике (см. 10.4.2). При этом, описывая процесс решения задачи в терминах операции п. 5, на каждом из этапов (1, 2, 3, ..., *i*) указываем множества исходных данных и результатов заданием, в общем случае их граничных значений (или граничных значений индексов).

7. Все сказанное есть описание процесса выполнения одного шага нисходящего проектирования. Далее все то же выполняем на каждом последующем шаге, т.е. для каждого укрупненного блока полученной схемы составляем свою схему алгоритма, рассматривая задачу каждого блока как самостоятельную и применяя каждый раз настоящую методику, начиная с п. 1.

Внимание! Очередность рассмотрения блоков в укрупненной схеме — от выхода к входу.

Операции этого пункта выполняем до тех пор, пока не получим подробные схемы для всех укрупненных блоков всех укрупненных схем.

8. Схемы отдельных укрупненных блоков механически объединяем в соответствии с иерархической структурой укрупненных схем и получаем подробную схему алгоритма всей задачи.

Примечание. Отметим, что кроме реализации основных принципов структурного подхода методика позволяет определить вид алгоритма, строящегося на каждом шаге (линейный, разветвляющийся или ци-

клический), а также обеспечивает вывод рабочих формул, начальных и конечных значений всех переменных для каждого вложенного цикла.

Дополнение. При применении излагаемого подхода наибольшая трудность для учащихся, особенно на первых порах, — суметь разбить задачу на подзадачи (выделить наиболее крупные этапы решения задачи). Поэтому приведем некоторые рекомендации по выполнению этой операции, не являющиеся, конечно, универсальными:

а) анализируем множество исходных данных алгоритма и выделяем наиболее крупные подмножества их, обрабатываемые каждое независимо от других. Тогда за наиболее крупные этапы принимаем операции по обработке каждого такого подмножества;

б) аналогично можно проанализировать множество результатов и выделить наиболее крупные подмножества их, вычисляемые независимо друг от друга. Операции по вычислению каждого такого подмножества можно рассматривать как наиболее крупные этапы решения задачи.

Пример. Массивы $A(1{:}n)$ и $B(1{:}m)$ умножить на k.

Исходные данные задачи содержат два массива, которые могут обрабатываться независимо друг от друга, соответственно можно выделить две крупные однократные операции — умножение массива A на k, умножение массива B на k. К этому же придем, проанализировав результаты.

Далее рассмотрим применение изложенной методики к решению двух классов задач:

- вычисление значений функций нескольких переменных (вычисления по заданным формулам);
- обработка массивов (таблиц).

Это наиболее простые и в то же время наиболее распространенные классы задач — большинство инженерных и экономических расчетов сводятся к решению таких задач. Да и в жизни нам, если и приходится сталкиваться с какими-либо вычислениями, то это, как правило, вычисления по готовым формулам либо работа с таблицами. К тому же наиболее доступные и распространенные языки программирования (Бейсик, Паскаль и др.) предназначены в первую очередь для решения именно таких задач.

Контрольные вопросы

1. Какой блок схемы алгоритма называется «элементарным»? Что такое «укрупненный» блок, «укрупненная» схема алгоритма?
2. Какая схема алгоритма называется подробной? В чем заключается подход к построению таких схем за ряд шагов?

3. В чем суть методики составления алгоритмов?
4. В чем особенность составления укрупненной схемы циклического алгоритма?

11.2. Алгоритмы вычисления функций нескольких переменных

Рассмотрим типичную задачу — функция Y от нескольких переменных задана *аналитически*, т.е. формулой, содержащей имена нескольких переменных (аргументов) — x, z, w ... . Задано множество возможных значений каждого аргумента.

Требуется вычислить значения функции Y при изменении значений различного числа ее аргументов в заданных диапазонах.

Пример. Формула вычисления объема усеченной пирамиды имеет вид

$$V = \frac{1}{3}h(q + r + \sqrt{qr}), \qquad (11.1)$$

где V — объем пирамиды; h — высота; $q(r)$ — площадь нижнего (верхнего) основания, т.е. V — функция трех переменных и в математике обозначается так: $V = f_{(h, q, r)}$.

При использовании формулы (11.1) могут возникнуть задачи различной сложности:

а) вычислить значение V для одного набора значений аргумента. Например, для $h = h_1$, $q = q_1$, $r = r_1$. Решение этой задачи описывается линейным алгоритмом;

б) вычислить значение V при различных значениях одного аргумента, например $h = h_1, h_2, ..., h_n$ и $q = q_1$, $r = r_1$. В этом случае мы имеем дело фактически с функцией одного аргумента ($V = f_{(h)}$) и решение задачи описывается алгоритмом с одним циклом;

в) вычислить значение V при различных значениях двух или трех аргументов. Например, вычислить значения V при:

$$\begin{gathered} h = h_1, \quad h_2, ..., h_n; \\ q = q_1, \quad q_2, ..., q_i; \\ r = r_1, \quad \text{т.е.} \quad V = f_{(h, q)} \end{gathered}$$

или все то же, но $r = r_1, r_2, ..., r_m$; т.е. $V = f_{(h, q, r)}$.

В этих случаях мы приходим к задаче составления алгоритма с несколькими вложенными циклами. Именно эту задачу и будем рассматривать далее.

Для начала сформулируем задачу более строго: функция $Y = f_{(x, z)}$ или $Y = f_{(x, z, w)}$ (ограничимся этими двумя случаями) задана аналитиче-

ски. Задано множество возможных значений каждого аргумента (перечислением их значений либо способом их вычисления).

Требуется вычислить значения Y при всех возможных (по условию задачи) сочетаниях значений аргументов x и z (или x, z, и w).

Результатом решения задачи должна быть таблица, содержащая для каждого значения функции соответствующие ему значения аргументов. Впредь, говоря о выводе значений функции, будем подразумевать вывод таблицы ее значений.

Суть нашей задачи — перебор значений аргументов.

Метод решения ее должен обеспечить такой порядок перебора значений аргументов, чтобы каждый возможный набор значений x и z (или x, z и w) появлялся один и только один раз.

Рассмотрим, как подобный перебор можно осуществить.

Возьмем такую жизненную ситуацию.

Студентка в магазине «Одежда» желает подобрать для себя наилучший ансамбль (сочетание) «пальто — шляпка». Известно, что в магазине пять моделей пальто и шесть моделей шляпок ее размера. Ей потребуется перебрать все возможные сочетания пальто и шляпок. Как она поступит? Сначала она надевает первое пальто с первой, второй, ..., с шестой шляпкой, оценивая качество каждого сочетания. Затем надевает второе пальто и опять примеривает его поочередно с каждой шляпкой — первой, второй, шестой.

Аналогично поступает она с остальными пальто. Очевидно, ей придется выполнить $5 \times 6 = 30$ примерок.

Если бы студентке потребовалось подобрать ансамбль «пальто — шляпка — шарф» при числе подходящих видов шарфов, например четыре, то количество примерок было бы равно $5 \times 6 \times 4 = 120$, так как в этом случае пришлось бы оценивать сочетание первого шарфа с каждой возможной парой — «пальто — шляпка», второго шарфа с каждой такой парой и т.д.

Таким образом обеспечивается полный перебор вариантов.

Вернемся к нашей задаче и рассмотрим первый ее вариант.

Вычисление функций вида $Y = f_{(x, z)}$

Задача 11.2

Вычислить значения функции

$$V = \frac{1}{3}h\left(q + r + \sqrt{qr}\right)$$

при всех возможных сочетаниях значений h и q (в заданном диапазоне значений) и одном значении r. Примем для конкретности: $h = 3, 5, 6, 9, 12$; $q = 3, 6, 9, 12$.

Решение

Шаг 1. Рассматриваем решение задачи в целом.

1. Исходные данные: r, массив $H(1:5) = \{3, 5, 6, 9, 12\}$. Результат: V (20 значений).

2. Метод решения задачи. (Вспомним о пальто и шляпках.)

Примем сначала $h = h_1$ и будем изменять q, заставив его «пробегать» все значения от 3 до 12. Затем примем $h = h_2$ и вновь заставим q пробегать все значения и т.д., пока h не достигнет значения h_5.

На каждом наборе значений h и q вычисляем значение V. Очевидно, в этом случае каждое сочетание значений h и q появляется один и только один раз.

Количество значений функции V — k_V:

$$k_V = k_h k_q = 5 \times 4 = 20,$$

где k_h, k_q — количество значений h и q соответственно.

Очевидно, приведенный метод пригоден для вычисления значений любой подобной функции двух переменных.

3. Рассмотренный метод требует *многократного* выполнения одной крупной операции — «вычислить значения V при одном значении h и четырех значениях q». Соответственно порядок решения задачи (в общем виде) будет таков.

Этап 1. Вычислить значения V при $h = h_1$ и всех значениях q.

Этап 2. Вычислить значения V при $h = h_2$ и всех значениях q.

Этап i. Вычислить значения V при $h = h_i$ и всех значениях q,

$$i := \begin{vmatrix} 1 \\ i+1. \\ 5 \end{vmatrix}$$

4. Рисуем типовую схему циклического алгоритма и заполняем ее. Получим укрупненную схему алгоритма всей задачи (рис. 11.3.). *Она указывает порядок изменения величины h.* В ней блок 4 — укрупненный.

Шаг 2. Рассматриваем задачу блока 4 (см. рис. 11.3) как самостоятельную:

«Вычислить значения функции V при всех значениях q и одном значении $h = h_i$».

1. Исходные данные: r, h_i. Результат: V (четыре значения).

2. Мы пришли к задаче вычисления функции одной переменной

$$V = f_{(q)},$$

где $q := \begin{vmatrix} 3 \\ q+3. \\ 12 \end{vmatrix}$

В данном случае все составляющие циклического алгоритма известны, поэтому, изобразив типовую схему алгоритма и заполнив ее, получим в результате подробную схему алгоритма блока 4 (рис. 11.4). *Она указывает порядок изменения величины q.*

Объединив схемы на рис. 11.3. и 11.4, получим подробную схему алгоритма всей задачи (рис. 11.5).

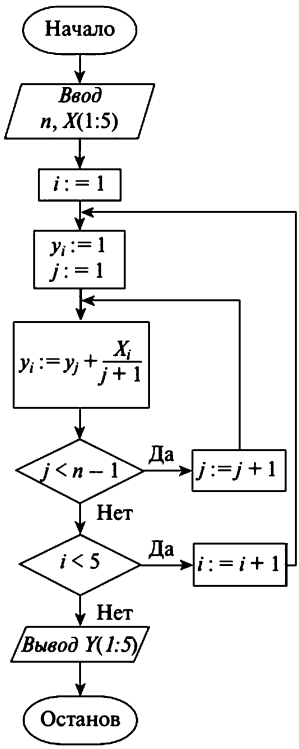


Рис. 11.2

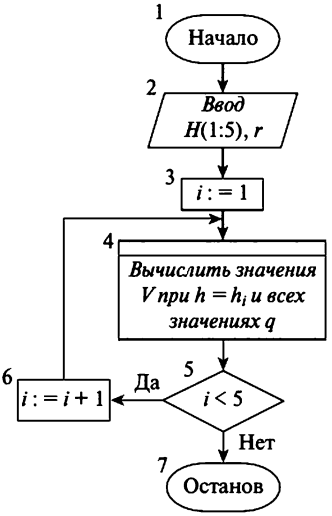


Рис. 11.3

Допустим теперь, что по условиям задачи требуется запомнить и сохранить в ЭВМ все значения некоторой переменной, например *V*. Наш алгоритм этого не предусматривает.

Как запомнить все значения величины, вычисляемые последовательно? С этой целью организуем массив, например *P*(1:20), и каждому его элементу присвоим одно значение *V* (первое значение присвоим P_1, второе — P_2, ..., *k*-е значение — P_k), т.е. каждое значение *V* заносим в отдельную ячейку памяти ЭВМ. Индекс *k*, очевидно, должен меняться по формуле $k = k + 1$ после вычисления каждого значе-

ния V, т.е. в схему, представленную на рис. 11.5, нужно добавить блок вычисления P_k и k:

$$P_k := V, \quad k := k + 1,$$

а также блок присваивания начального значения $k = 1$.

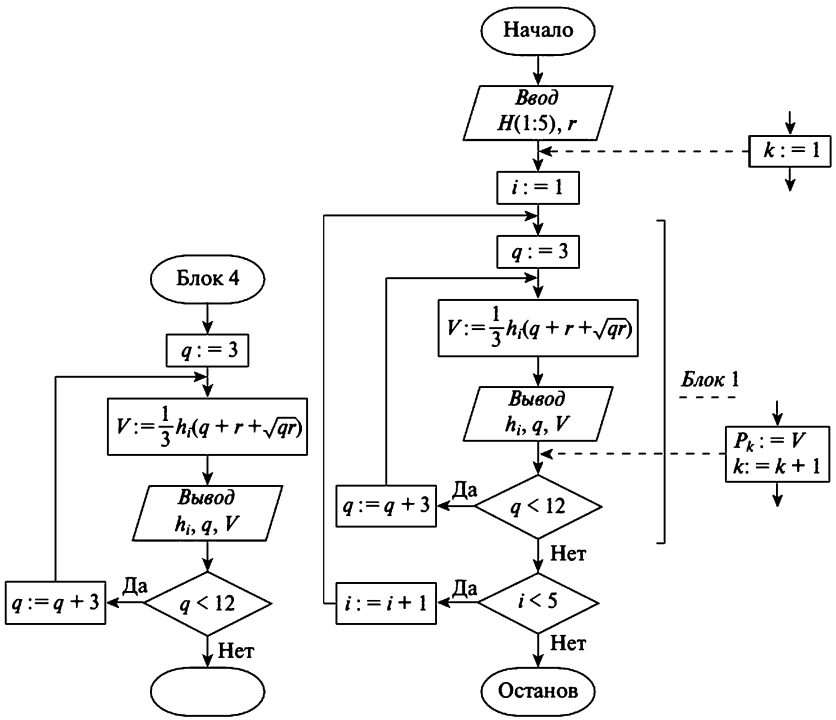


Рис. 11.4

Рис. 11.5

На рис. 11.5 стрелками показано положение этих блоков в схеме алгоритма.

Пояснение. Указанный прием, обеспечивающий запоминание значений переменной, применим в любой задаче рассматриваемого вида, т.е. в любую схему достаточно добавить два указанных блока (даже не меняя имен переменных P и k, если они не повторяются в схеме).

Теперь рассмотрим 2-й вариант нашей задачи.

Вычисление функции вида $Y = f_{(x, z, w)}$

Задача 11.3

Вычислить все возможные значения функции V [см. (11.1)] при условии, что r, h, q принимают, например, такие значения:

$$r = 1, 2, 4, 9\ (k = 4);$$
$$h = 3, 5, 6, 9, 12\ (k_h = 5);$$
$$q = 3, 6, 9, 12\ (k_q = 4).$$

Решение

Шаг 1. Рассматриваем решение задачи в целом.

1. Исходные данные: массивы Д(1:4), //(1:5). Результат: V (kV значений).

2. Метод решения задачи. (Вспомним о пальто, шляпке и шарфике.) Сначала зафиксируем значение $r = r_1$ и будем перебирать все возможные пары значений h и q так, как это выполнено выше в первом случае (для $V = f_{(h,\ q)}$. На каждом наборе значений r, h, q вычисляем значение функции V. Далее принимаем $r = r_2$ и опять перебираем все возможные пары значений h и q и т.д., пока r не достигнет конечного значения r_4, h — значения h_5, q — значения 12. При этом каждое сочетание значений r, h, q появляется один и только один раз.

Количество значений функции V:

$$k_V = k_r k_h k_q = 4 \times 5 \times 4 = 80.$$

Обращаем внимание, что указанный порядок перебора значений аргументов пригоден для вычисления значений любых подобных функций трех переменных.

3. Решение задачи заключается в *многократном* выполнении одной крупной операции — «вычислить значения V при одном значении r и всех сочетаниях значений h и q».

При этом на этапе 1 будут вычисляться все значения V при $r = r_1$, на этапе 2 — при $r = r_2$ и т.д., на этапе j — при $r = r_j$ где

$$j := \begin{vmatrix} 3 \\ j+1. \\ 4 \end{vmatrix}$$

4. Составим соответствующую этому процессу укрупненную схему алгоритма (рис. 11.6). В ней блок 4 — укрупненный.

Шаг 2. Рассматриваем задачу блока 4 (рис. 11.6) как самостоятельную.

Исходные данные: r_j, H(1:5).

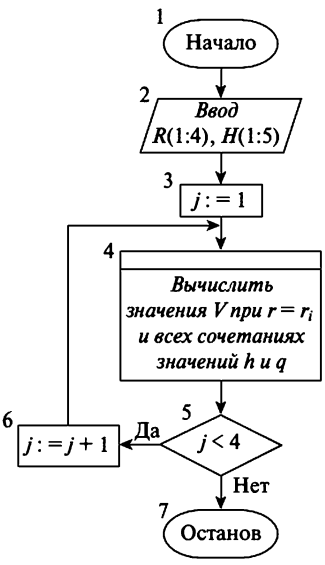


Рис. 11.6

Результат: V (20 значений). Поскольку величина r_j в пределах блока 4 постоянна, то вычисляем фактически значения функции двух переменных $V = f_{(h, q)}$ при всех сочетаниях переменных h и q, т.е. при $h = 3, 5, 9, 12$; $q = 3, 6, 9, 12$. Но это и есть, очевидно, задача 11.2, схема алгоритма которой изображена на рис. 11.5.

Единственное различие этих задач — исходная величина, в нашем случае r_j, а не r, как в задаче 11.2.

Объединяя схемы рис. 11.5 и 11.6, получим подробную схему алгоритма задачи 11.3 (рис. 11.7).

Если необходимо запомнить результаты, то следует добавить те же два блока, что и в алгоритме задачи 11.2.

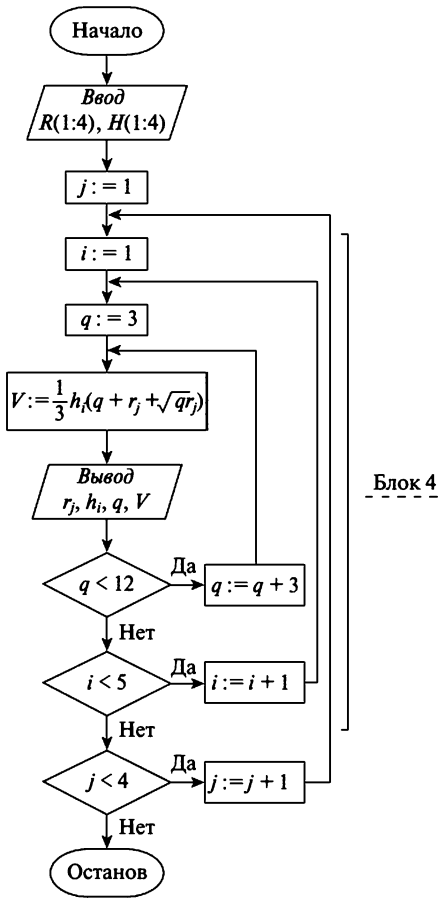


Рис. 11.7

Отметим, что схема алгоритма задачи 11.3 (вычисление $V = f_{(n, q, r)}$) отличается от схемы алгоритма задачи 11.2 (вычисление $V = f_{(h, q)}$) только дополнительным внешним циклом, в котором изменяется величина j от 1 до 4 с шагом 1.

Итак, мы рассмотрели построение алгоритмов вычисления значений функций двух и трех переменных. Усвоив решение этих задач, несложно перейти к вычислению значений функций с бóльшим числом переменных.

Как это сделать, покажем на следующем примере.

Допустим (чисто условно), что в формуле (11.1) вместо константы «1/3» участвует некоторая переменная t, изменяющаяся от 3 до 18 по формуле $t_{i+1} = 2t_i - 2$ (т.е. $t_1 = 3$), и требуется обеспечить вычисление функции $V = f_{(h, q, r, t)}$

при всех возможных по условию задачи сочетаниях четырех переменных.

Чем же будет отличаться схема алгоритма в этом случае от представленной на рис. 11.7 схемы, вычисляющей функцию V от трех переменных?

Очевидно, тем же, чем схема на рис. 11.7 отличается от схемы на рис. 11.5, т.е. еще одним циклом, внешним по отношению к исходной схеме, в которой будет изменяться новая переменная t. В трех блоках этого цикла будет размещена информация:

$$t := \begin{vmatrix} 3 \\ 2t-2. \\ 18 \end{vmatrix}$$

Таким же образом можно ввести и 5-ю и 6-ю и т.д. переменные.

В заключение отметим некоторые особенности алгоритмов, вычисляющих функции нескольких переменных, и процесса их составления.

1. Число вложенных циклов алгоритма равно числу аргументов функции.

2. При вычислении значений любой подобной функции с двумя (тремя) аргументами можно использовать один и тот же метод решения задачи (метод перебора вариантов), приведенный в описании решения задачи 11.2 (11.3).

3. В подобных задачах обычно совмещается процесс вычисления результатов с выводом их, а именно: после вычисления каждого очередного значения функции выводится (печатается) одна строка таблицы значений этой функции.

При необходимости сохранить в ЭВМ указанную таблицу (запомнить ее) можно организовать матрицу размера $(1{:}k,\ 1{:}n+1)$, где k — количество значений функции, а n— число ее аргументов, и заполнять в каждом цикле по одной строке этой матрицы.

4. В указанных задачах число значений функции очень быстро растет с ростом числа аргументов и числа значений аргументов функции. Настолько быстро, что для функции, например $Y = f_{(x_1,\ldots,x_{20})}$ при условии, что x_i принимает не менее 10—20 значений, вычислить все значения Y практически невозможно ни на одной ЭВМ (тем более ПЭВМ)! Желающие могут сами в этом убедиться (хотя бы теоретически).

Это, в частности, к вопросу о том, действительно ли ЭВМ может решать любые задачи? Заметим к тому же, что функции от двадцати аргументов далеко не самые сложные, которые встречаются в науке и технике.

Задачи для самостоятельного решения

1. Вычислить значения функции

$$y = (5x^2 + b) + r,$$

где $x = 2, 3, 4, 6, 7$; $b = 7, 8, 10, 12, 25, 30$. Результаты представить в виде массива.

2. Вычислить значения функции y:

$$y = \begin{cases} |x| - c^2, & \text{если } x < 8; \\ \sqrt{x} + 2cr, & \text{если } 8 \le x, \end{cases}$$

где $x = 2, 4, 6, 8, 10$; $r = 1, 2, 3, ..., 12$.

3. Условие то же, что в задаче 2. Определить сумму всех значений функции y.

4. Условие то же, что в задаче 8. Вывести значения функции y, отвечающие условию: $5 \le y \le 20$.

5. Условие то же, что в задаче 1, и, кроме того, $r = 2, 3, 4, 5, 6$.

6. Вычислить значения функции p:

$$p = (6\ln^2 x + \sqrt{d})^n + 3a^2,$$

где $x = 2, 4, 6, 8, 10, 12, 14$; d — принимает значения из отрезка [3, 7] с шагом $\Delta d = 0{,}5$; $a = 1, 2, 3, 7, 9, 10$. Результаты представить в виде массива.

7. Условие то же, что в задаче 6, но $p = \begin{cases} 2\ln x + d, & \text{если } d < 6; \\ 3d^2 + a, & \text{если } d = 6; \\ \sqrt{d} + 3ax, & \text{если } d > 6. \end{cases}$

11.3. Алгоритмы решения задач обработки массивов (матриц)

Рассмотрим на конкретном примере содержательный смысл изучаемой методики применительно к решению таких задач.

Задача 11.4

Матрица $A(1{:}\,n,\ 1{:}m)$ (m — кратно четырем) разделена по вертикали на две половины. Определить сумму элементов каждого столбца левой половины и сумму элементов каждого четного столбца правой половины матрицы A.

Решение

Шаг 1. Рассматриваем решение всей задачи в целом. Она заключается в однократном решении двух подзадач.

Поэтому в первую очередь нужно, не вникая в тонкости, выяснить, в каком порядке следует решать эти подзадачи.

На схеме, представленной на рис. 11.8, показан этот порядок.

Рис. 11.8

Шаг 2. Рассматриваем задачу укрупненного блока 2 как самостоятельную — «определить сумму элементов каждого четного столбца правой половины матрицы *A*».

Исходные данные: матрица $A(1{:}n,\ 1{:}m)$.

Результат: массив $S(1{:}\ m/4)$.

Описывая теперь порядок решения этой задачи, указываем (пока) только последовательность обработки столбцов, умалчивая и даже вообще не вникая в то, каким образом будем суммировать элементы в пределах столбца, т.е. описываем порядок решения задачи в самом общем виде.

Этап 1. Суммировать элементы $\left(\frac{m}{2}+2\right)$-го столбца и образовать элемент s_1.

Этап 2. Суммировать элементы $\left(\frac{m}{2}+4\right)$-го столбца и образовать элемент s_2.

Этап *i*. Суммировать элементы $\left(\frac{m}{2}+2i\right)$-го столбца и образовать элемент s_i.

Схема алгоритма рис. 11.9 отражает именно этот порядок обработки столбцов. Здесь блок 2—3 — укрупненный.

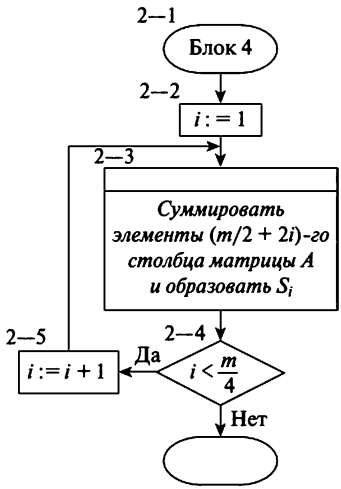


Рис. 11.9

Шаг 3. Рассматриваем задачу блока 2—3 (см. рис. 11.9) как самостоятельную. Текст, содержащийся в блоке 2—3 схемы, и есть формулировка этой задачи. Только на этом шаге, рассматривая задачу указанного блока, следует задуматься о том, в каком порядке суммировать элементы в пределах каждого столбца. Но сначала отвечаем на вопрос: какие

величины являются исходными данными и результатами задачи блока 2—3? Обычно учащиеся считают, что исходные данные — элементы первой строки, а результат — s_1. На самом деле такими величинами являются величины, указанные в формулировке задачи.

Следовательно, исходные данные: элементы k-го столбца матрицы A, где $k = \frac{m}{2} + 2i$, т.е.

$a_{1,k}$; $a_{2,k}$; $a_{3,k}$; $a_{n,k}$ — их мы и должны суммировать.

Результат: s_i (одно значение).

Теперь описываем порядок суммирования элементов одного, k-го столбца, общий для всех столбцов:

Этап 1. $p = a_{1,k} + a_{2,k}$.
Этап 2. $p = p + a_{3,k}$.
Этап j. $p = p + a_{j+1,k}$,

где $j := \begin{vmatrix} 1 \\ j+1; \\ n-1 \end{vmatrix}$ начальное значение $p = a_{1,k}$.

Этот процесс и описывает схема рис. 11.10.

Объединив схемы рис. 11.9 и рис. 11.10, получим подробную схему алгоритма блока 2, где будет указан и порядок обработки столбцов, и порядок суммирования элементов столбца.

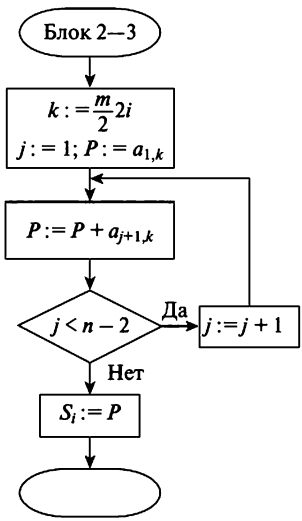


Рис. 11.10

Схему блока 1 (см. рис. 11.8) рекомендуем читателю построить самостоятельно, она подобна схеме блока 2, но несколько проще.

Рассмотрим еще ряд задач, но уже не так подробно.

Задача 11.5

Записать элементы массива D(1: 20) в виде матрицы C(1:4, 1:5) и вычислить сумму угловых элементов матрицы.

Решение

Шаг 1. Рассматриваем решение всей задачи в целом.

1. Исходные данные: массив D(1:20).

Результаты: матрица C(1:4, 1:5), S.

2. Метод решения задачи в наиболее общем виде заключается в однократном решении двух подзадач:

- образование матрицы $C(1{:}4, 1{:}5)$;
- вычисление суммы S.

3. Рисуем укрупненную схему алгоритма задачи (рис. 11.11).

Рис. 11.11

Шаг 2. Рассматриваем задачу укрупненного блока 3 (см. рис. 11.11).

1. *Исходные данные*: массив $D(1{:}20)$.

Результат: матрица $C(1{:}4, 1{:}5)$.

2. Метод решения задачи. Первые пять элементов массива D записываем в первую строку матрицы C, следующие пять элементов — во вторую строку, следующие пять элементов — в третью, и т.д.

3. Описываем этапы решения задачи.

Этап 1. Из элементов с d_1 по d_5 образуем первую строку матрицы C.

Этап 2. Из элементов с d_6 по d_{10} образуем вторую строку матрицы C.

4. Записываем параметры переменной i:

$$i := \begin{vmatrix} 1 \\ i+1. \\ 4 \end{vmatrix}$$

5. Рисуем укрупненную схему алгоритма блока 3 (рис. 11.12).

Шаг 3. Рассматриваем задачу укрупненного блока 3—3 (рис. 11.12) — «образуем i-ю строку матрицы C».

1. Исходные данные:

$$d_l, d_{l+1}, d_{l+2}, \ldots, d_{5i},$$

где $l = (i-1)5 + 1 = 5i - 4$.

Результат: $c_{i,1}, c_{i,2}, c_{i,3}, \ldots, c_{i,5}$.

2. Метод решения задачи очевиден.

3. Описываем этапы решения задачи.

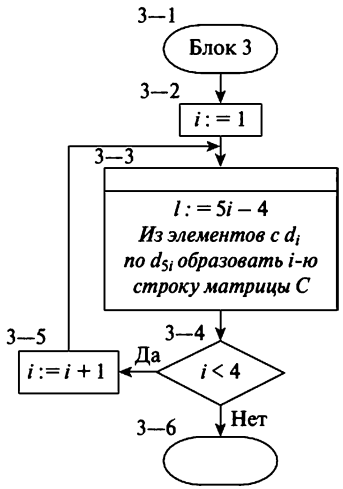


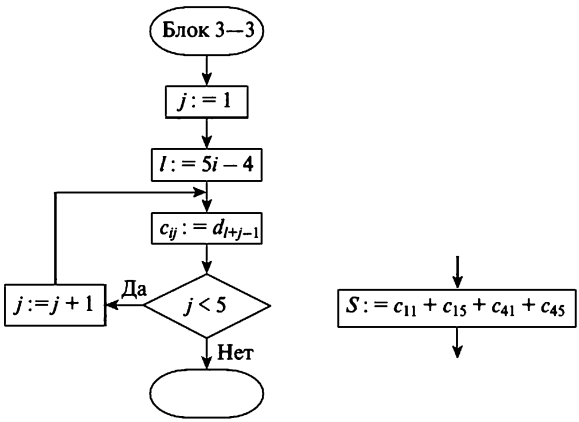
Рис. 11.12

Этап 1. $c_{i,1} = d_l$.
Этап 2. $c_{i,2} = d_{l+1}$.
Этап i. $c_{i,j} = d_{l+j-1}$.

4. Записываем параметры переменной j:

$$j := \begin{vmatrix} 1 \\ j+1. \\ 5 \end{vmatrix}$$

5. Рисуем подробную схему алгоритма блока 3—3 (рис. 11.13, *а*).



а) *б*)

Рис. 11.13

Шаг 4. Составляем схему блока 4. Она тривиальна (рис. 11.13, *б*).

Объединяя теперь схемы блоков 3 и 3—3, получим подробную схему алгоритма блока 3. Объединяя ее и схему блока 4, получим подробную схему алгоритма всей задачи (рис. 11.14).

Задача 11.6

В массиве $A(1{:}n)$ каждую группу из k последовательно расположенных элементов, совпадающую с массивом $B(1{:}k)$, заменить элементами массива $C(1{:}k)$ соответственно ($k < n$).

Решение

Шаг 1. Рассматриваем решение всей задачи в целом.

1. Исходные данные: k, n, массивы $A(1{:}n)$, $B(1{:}k)$, $C(1{:}k)$.

Результат: массив $A(1{:}n)$.

2. Метод решения задачи (в наиболее общем виде). Сравниваем каждую группу из k элементов массива A, начинающуюся с первого, вто-

рого, третьего и т.д. элемента, с массивом B и в случае равенства группы массиву B заменяем ее массивом C.

3. Описываем этапы решения задачи.

Этап 1. Проверяем, равна ли группа элементов $a_1 \div a_k$ массиву $B(1{:}k)$, — если условие выполняется (*Да*), — элементам этой группы присваиваем значения соответствующих элементов массива C, иначе (*Нет*) — переходим к следующему этапу.

Этап 2. То же, для элементов $a_1 \div a_{k+1}$ массива A.

Этап 3. То же, для элементов $a_3 \div a_{k+2}$ массива A.

Этап i. Проверяем, равна ли группа элементов $a_i \div a_{k+i-1}$ массиву $B(1{:}k)$ (назовем это условие *условием A*), если *Да* — элементам этой группы присваиваем значения соответствующих элементов массива $C(1{:}k)$, если *Нет* — переходим к следующему этапу.

Этап d (последний). То же, для элементов $a_{n-k+1} \div a_n$.

4. Выявляем отношение, связывающее индексы при первых элементах группы на i-м и d-м этапах:

$$i \le n - k + 1.$$

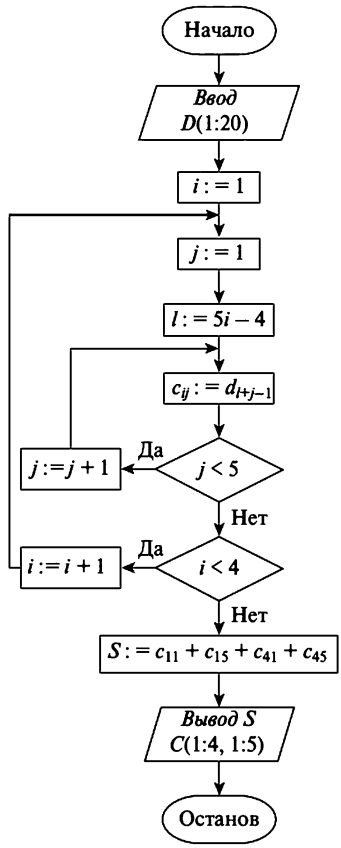


Рис. 11.14

Это и есть условие повторения цикла.

5. Записываем параметры переменной i:

$$i := \begin{vmatrix} 1 \\ i+1 \\ n-k+1 \end{vmatrix}.$$

6. Рисуем укрупненную схему алгоритма всей задачи (рис. 11.15). В ней блоки 4 и 5 — укрупненные.

Шаг 2. Рассматриваем задачу блока 5 (см. рис. 11.15): «элементам массива A, указанным в блоке 5, присвоить значения соответствующих элементов массива $C(1{:}k)$».

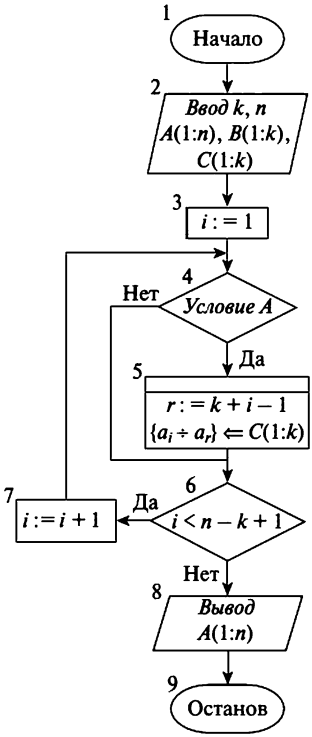


Рис. 11.15

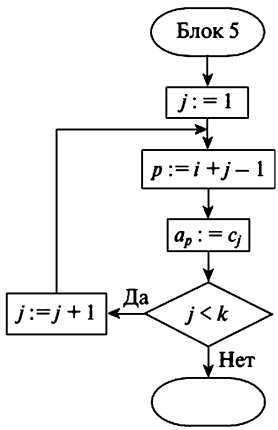


Рис. 11.16

1. *Исходные данные*: $c_1, c_2, c_3, \ldots, c_k$.
Результат: $a_i, a_{i+1}, a_{i+2}, \ldots, a_{k+i-1}$.

2. Метод решения задачи очевиден.

3. Описываем этапы решения задачи.

Этап 1. $a_i = c_1$.
Этап 2. $a_{i+1} = c_2$.
Этап j. $a_{i+j-1} = c_j$.

4. Записываем параметры переменной j:

$$j := \begin{vmatrix} 1 \\ j+1. \\ k \end{vmatrix}$$

5. Рисуем подробную схему алгоритма блока 5 (рис. 11.16).

Шаг 3. Рассматриваем задачу укрупненного блока 4 (см. рис. 11.15): «проверить — условие A выполняется?»

1. *Исходные данные*: $a_i, a_{i+1}, a_{i+2}, \ldots, a_{k+i-1}$; $b_1, b_2, \ldots, b_k$.
Результат: «*Да*» или «*Нет*».

2. Метод решения задачи. Сравниваем попарно элементы: и: a_i, b_1, a_{i+1}, и b_2 и т.д.; если сравниваемые элементы равны (*Да*), то переходим к следующей паре, если не равны (*Нет*), то рассматриваемая группа элементов массива A не равна массиву B и выходим из блока 4 по выходу *Нет*.

3. Описываем этапы решения задачи.

Этап 1. Проверяем $a_i = b_1$, *Да* — переходим к следующему этапу;
Нет — переходим к выходу *Нет* блока 4.

Этап 2. Проверяем $a_{i+1} = b_2$ *Да* — переходим к следующему этапу;
Нет — переходим к выходу *Нет* блока 4.

Этап 3. Проверяем $a_{i+j-1} = b_j$ *Да* — переходим к следующему этапу;
Нет — переходим к выходу *Нет* блока 4.

4. Записываем параметры переменной j:

$$j := \begin{vmatrix} 1 \\ j+1. \\ k \end{vmatrix}$$

5. Рисуем подробную схему блока 4 (рис. 11.17).

Объединяя теперь подробные схемы блоков 4 и 5 в соответствии с укрупненной схемой задачи (см. рис. 11.15), получаем подробную схему алгоритма всей задачи (рис. 11.18).

Следует отметить недостаток этого алгоритма — блок 4 имеет 2 выхода, что с точки зрения структурного программирования нежелательно.

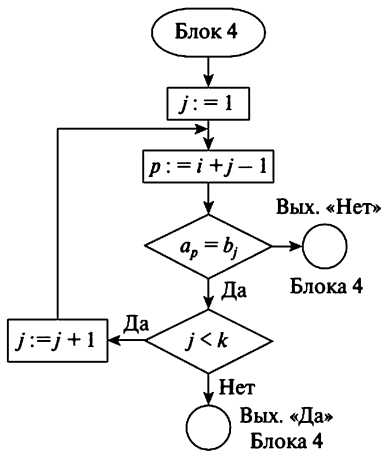


Рис. 11.17

Рекомендации по решению задач обработки массивов (матриц)

1. При решении задач, оперирующих с матрицами большого или переменного размера, часто бывает затруднительно уяснить метод (порядок) решения задачи ввиду плохой обозримости таких матриц.

Поэтому рекомендуется процесс решения задачи прослеживать вначале на небольших матрицах конкретного размера и составлять алгоритм именно для таких матриц с последующим обобщением его на случай матриц заданного размера.

2. Специфика рассматриваемых задач: в них в качестве подзадач выступают обычно операции по обработке всей матрицы в целом, части матрицы, столбца, строки и т.д., т.е. набор их невелик. Из этого набора и следует выбирать наиболее крупные операции на каждом шаге решения задачи.

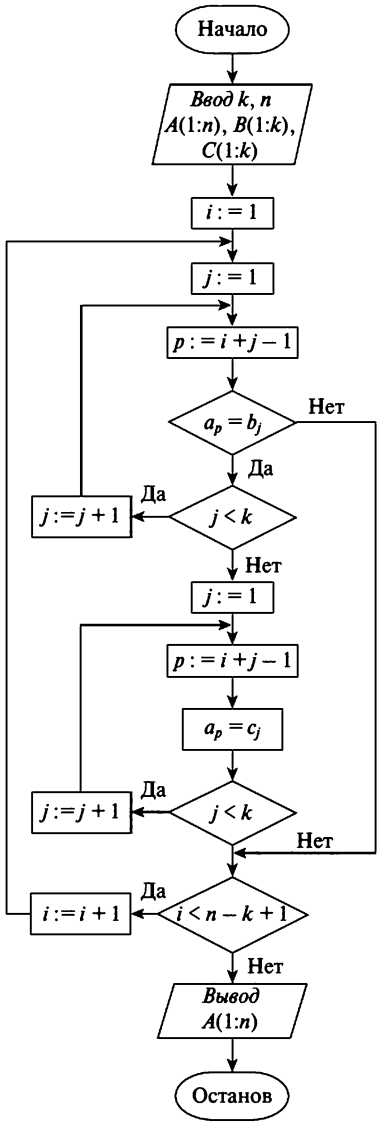


Рис. 11.18

3. Для решения матричной задачи часто главное — определить последовательность вложенных подзадач убывающей сложности.

В этом случае полезно бывает начинать с выбора не самой крупной операции, а, наоборот, с самой мелкой и искать операции возрастающей сложности.

Рассмотрим на конкретной задаче применение рекомендации п. 3.

Задача 11.7

Проверить, есть ли в матрице $P(1{:}k, 1{:}n)$ столбец, равный какому-либо столбцу матрицы $Q(1{:}k, 1{:}m)$.

Здесь очевидно, самая мелкая операция — сравнение одного элемента матрицы P с одним элементом матрицы Q (назовем ее — операция A).

Следующая по сложности операция — сравнение одного столбца матрицы P с одним столбцом матрицы Q (операция B); она заключается в повторении k раз операции A.

Более крупная операция — сравнение одного столбца матрицы P со всеми столбцами матрицы Q (т.е. со всей матрицей Q) — назовем ее операцией C. Она требует, чтобы операция B была выполнена m раз.

Еще более сложная операция — сравнение каждого столбца матрицы P с матрицей Q (операция D). Она заключается в выполнении n раз операции C и приводит к решению всей задачи.

Вот мы и получили множество задач возрастающей сложности.

Описывая теперь процесс выполнения каждой операции по этапам (причем рассматривая операции в таком порядке — D, C, B, A), получим четыре схемы, четыре вложенных цикла. Надеемся, что читатель сам сможет решить эту задачу.

Задача 11.8

Найти сумму элементов матрицы $A(1{:}/r, 1{:}m)$.

Решение

Исходные данные: n, m, $A(1{:}n, 1{:}m)$; результат: S (одно значение).

Шаг 1. Рассматриваем решение задачи в целом.

1. Исходные данные и результаты — см. выше.

2. Метод решения задачи в наиболее общем виде представляет собой следующую последовательность этапов.

Этап 1.

Вычислить $Y =$ «сумма элементов 1-й строки»

$$\left(\text{т.е. } \sum_{j=1}^{m} a_{1,j}\right).$$

Этап 2. Вычислить:

$$Y = Y + \text{«сумма элементов 2-й строки»} \left(\text{т.е. } \sum_{j=1}^{m} a_{2,j}\right).$$

Этап 3. Вычислить:

$$Y = Y + \text{«сумма элементов 3-й строки»} \left(\text{т.е. } \sum_{j=1}^{m} a_{3,j}\right).$$

Этап i. Вычислить:

$$Y = Y + \text{«сумма элементов } i\text{-й строки»} \left(\text{т.е. } \sum_{j=1}^{m} a_{i,j}\right),$$

где $i := \begin{vmatrix} 1 \\ i+1. \\ n \end{vmatrix}$

Операцию 1-го этапа следует привести к такому виду:

$$Y = Y + \sum_{j=1}^{m} a_{1,j}.$$

Отсюда следует: должно быть начальное значение $Y = 0$. Условие повторения цикла: $i \le n$.

Шаг 2. Рассматриваем задачу i-го этапа шага 1: «Определить сумму переменной Y и элементов i-й строки матрицы A».

1. *Исходные данные*: $Y, a_{i,1}, a_{i,2}, \ldots, a_{i,m}$; результат: Y.

2. Метод решения задачи определяется такой последовательностью операций.

Этап 1. $Y = Y + a_{i,1}$

Этап 2. $Y = Y + a_{i,2}$

Этап j. $Y = Y + a_{i,j}$

где $j := \begin{vmatrix} 1 \\ j+1. \\ m \end{vmatrix}$

Условие повторения цикла: $j \le m$.

Далее, надеемся, читатель сам сможет изобразить схему алгоритма каждого шага и объединить их в общую схему алгоритма всей задачи.

Отметим, что совершенно аналогичным образом можно описать процесс построения алгоритма следующих задач:

— вычислить произведение элементов матрицы A;

— определить максимальный (минимальный) элемент матрицы A и многих подобных им задач.

Задачи для самостоятельного решения

1. Матрицу $A(1{:}n, 1{:}m)$ умножить на k и найти максимальный элемент 2-го столбца.

2. Найти сумму матриц $A(1{:}n, 1{:}m)$ и $B(1{:}n, 1{:}m)$. Умножить каждый элемент 1-й строки матрицы A на соответствующий элемент последней строки матрицы B.

3. Найти сумму элементов каждой строки матрицы $A(1{:}n, 1{:}m)$ и максимальный элемент 2-й строки.

4. Определить наибольший элемент в каждом столбце матрицы $B(1{:}l, 1{:}k)$. Вычислить сумму элементов 2-го столбца.

5. Записать элементы матрицы $A(1{:}n, 1{:}m)$ в виде массива $B(1{:}n*m)$. Найти минимальный элемент 2-й строки.

6. В матрице $B(1{:}l, 1{:}k)$ сместить каждую строку, начиная со второй, на одну вверх. Первую строку поставить на место последней. Найти разность максимального элемента 3-й строки и минимального элемента 2-го столбца.

7. Переставить строки матрицы $C(1{:}l, 1{:}k)$: первую с последней, вторую с предпоследней и т.д. Умножить 3-ю строку матрицы на сумму элементов 5-го столбца.

8. Элементы матрицы $B(1{:}n, 1{:}m)$, большие 5 и меньшие 20, записать в массив $C(1{:}n*m)$. Вычислить разность соответствующих элементов первой и последней строк.

9. Образовать матрицу $B(1{:}d, 1{:}d)$, все элементы главной диагонали которой равны единице. Угловые элементы матрицы принять равными k, все остальные элементы принять равными десяти.

ГЛАВА 12

ОСНОВЫ ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ БЕЙСИК

Бейсик — ***язык программирования, т.е. средство для записи алгоритма решения задачи в виде, понятном для ЭВМ, в виде программы для ЭВМ.***

Бейсик — один из наиболее простых языков и очень подходит для начального знакомства с ЭВМ. Он ориентирован на работу с ЭВМ в режиме диалога, т.е. обеспечивает непосредственное общение человека с ЭВМ в процессе решения задачи. Это и определило его широкое использование в ПЭВМ.

В настоящее время Бейсик относится к наиболее распространенным языкам — вряд ли в нашей стране найдется ПЭВМ, которая бы «не понимала» этого языка.

К недостаткам языка Бейсик можно отнести множество его версий (вариантов).

Каждая подобная версия предполагает использование своей системы программирования, включающей язык программирования и транслятор программ с этого языка (компилятор или интерпретатор).

С ПЭВМ IBM PC, которые преобладают в машинном парке России, связана своя система программирования, своя версия языка Бейсик — разработка фирмы MicroSoft.

Основная особенность этой системы программирования заключается в том, что она развивается в соответствии с потребностями пользователей и ростом возможностей IBM-совместимых ПЭВМ.

Первоначальная разработка этой фирмы — система BASICA, затем появилась система GWBASIC, затем различные версии системы QuickBASIC, последняя из которых — версия 4.5.

В состав MS-DOS, начиная с версии 5.0, фирма MicroSoft включила систему QBASIC, представляющую собой усеченный вариант QuickBASIC 4.5.

Отметим, что разработки фирмы MicroSoft не ограничились QuickBASIC 4.5. В 1990 году появилась MicroSoft Basic Professional Development System версии 7.1 — логическое расширение и развитие

системы QuickBASIC, предназначенная для профессиональных разработок, позволяющая создавать более мощные программные комплексы.

Принципиально новым словом фирмы MicroSoft явилось появление в 1991 г. версии 1.0, в 1995-м — версии 4.0 системы Visual BASIC for Windows (VB/Win), а затем и 5.0 и 6.0.

Новой здесь является возможность для программиста создавать программы с максимально удобным для пользователя диалоговым интерфейсом, использующим такие средства, как световые меню, кнопки (световые), окна и т.д.

Иначе говоря, эта система позволяет создавать программы, в которых организуется диалог с использованием указанных средств.

Язык этой системы является расширением языка QuickBASIC 4.5, принципы составления программ в этой системе те же, что и в системах предшествующих версий (если не касаться процессов программирования ввода и вывода данных).

Отметим также, что все системы программирования фирмы MicroSoft обладают совместимостью «снизу-вверх», т.е. программа, разработанная для некоторой версии системы, может быть выполнена и в системах более поздних версий.

В этом смысле VB/Win не полностью совместима с QuickBASIC 4.5.

Исходя из всего изложенного, далее следовало бы сказать: «А вот теперь мы приступаем к изучению системы VB/Win — последнего слова в области программирования на языке Basic».

И мы действительно к этому приступим, но предварительно займемся изучением усеченного варианта системы QuickBASIC — QBASIC.

Среда QBASIC доступна для любой модели IBM-совместимых ПЭВМ — занимает на диске всего 325 Кбайт. В то же время, обладая достаточно большими возможностями, позволяет создавать комплексы программ для решения серьезных реальных задач прикладного характера, будучи очень простой в освоении. Ее можно рассматривать как переходный мостик к профессионально-ориентированным версиям языка Бейсик.

Ниже подробно описаны основные конструкции языка Бейсик. Для некоторых из них рядом, в скобках или в виде примечаний, приводятся отличия в их записи или выполнении в конкретных версиях Бейсика. Отсутствие таких пояснений говорит о том, что данная конструкция языка во всех указанных версиях Бейсика записывается и выполняется одинаково.

Подчеркнем, что в настоящей главе приведен минимальный (базовый) набор конструкций языка, только тех, знание которых действительно необходимо на начальной стадии обучения программированию. Мы считаем, что на этой стадии следует ограждать учащихся от излишней информации, дабы она не затемняла суть процесса программирования.

12.1. Основные сведения о языке Бейсик

12.1.1. Алфавит языка

В языке используются следующие символы:

- заглавные буквы латинского алфавита от *A* до *Z*,
- арабские цифры 0, 1, 2, ..., 9,
- знаки арифметических операций:

«+» — сложение; «–» — вычитание; «*» — умножение; «/» — деление; «^» — возведение в степень; «\» — деление нацело; «MOD» — деление по модулю.

Последние две операции допустимы лишь в QBASIC;

- знаки операций отношения:

  «=» — равно,
  «>» — больше,
  «<» — меньше,
  «>=» — больше или равно,
  «<=» — меньше или равно,
  «<>» — не равно;

- разделители и прочие символы:

  . — точка,
  , — запятая,
  ; — точка с запятой,
  : — двоеточие,
  ! — признак вещественной величины,
  # — признак вещественной величины двойной точности,
  % — признак целой величины,
  & — признак длинной целой величины (в QBASIC),
  $ — признак текстовой величины (в Бейсике УКНЦ и БК 0010 — «¤»),
  () — скобки круглые,
  " — кавычки.

В качестве символов языка Бейсик кроме перечисленных используются некоторые слова английского языка (LET, GOTO и т.д.). Будем вводить их по мере необходимости.

Используются также буквы русского алфавита, но только в текстовых константах.

12.1.2. Данные

В Бейсике могут использоваться следующие виды данных:

а) константы;

б) переменные:
- простые переменные,
- массивы,

т.е. те же, что использовались выше при работе с алгоритмами (см. 8 и 9). Каждый вид данных в свою очередь включает несколько типов.

Константы. В Бейсике используются числовые и текстовые константы. Числовая константа записывается в программе в виде конкретного числа. Подобные константы делятся на два типа — вещественные и целые.

Вещественная константа — последовательность десятичных цифр (со знаком или без него) и точки либо заканчивающаяся символом «!». Точка разделяет целую и дробную части константы. *Число цифр константы — не более* 7.

Такие константы в ЭВМ представляются обычно с некоторой погрешностью (хотя и очень малой).

Примеры: 78.0 — 65.25 6525!

В Бейсике вещественное число может быть представлено и в *экспоненциальном виде* (*в виде числа с плавающей точкой*) (см. главу 1).

Например,

$$6.83\ \text{E}\ 7\ (\text{т.е. } 6{,}83 \cdot 10^7 = 68\ 300\ 000),$$
$$-2.1\ \text{E} - 7\ (\text{т.е. } -2{,}1 \cdot 10^{-7} = -0{,}00000021).$$

В общем случае число N записывается в следующем виде:

$$\pm \mathbf{m}\text{E}\pm \mathbf{p},$$

здесь **m** называется мантиссой, а **p** — порядком числа N, E — обозначение основания степени, признак числа с плавающей точкой, при этом:
- **m** — содержит не более 7 цифр и $1 < |m| < 10$;
- **p** — целое число и $|\mathbf{p}| \le 38$.

Итак, признак вещественной константы:
- знак! в конце числа;
- буква E в экспоненциальной форме записи;
- любая запись числа без указания типа.

Примеры: 369! – 1.299E 09

В QBASIC подобные числовые константы называются *вещественными константами одинарной точности — количество цифр не более 7.*

Кроме них в этих версиях Бейсика используются *вещественные константы двойной точности — количество цифр такой константы не более 17.*

Признак подобной константы:

- знак # в конце числа;
- буква D в экспоненциальной форме записи.

Примеры: 369.28# 12 345 678.3456# –1.299456747D 09.

Запись числа с точкой без указания типа в QBASIC считается вещественной константой одинарной точности.

Целая константа — это набор десятичных цифр (со знаком или без), оканчивающихся знаком «%». Пример: 78%, –212%.

Целая константа должна лежать в диапазоне от –32 768% до 32 767%.

Длинная целая — константа такого типа используется в QBASIC. Она *лежит в диапазоне от* –2 147 483 647 до 2 147 483 648, заканчивается символом &.

Пример: 4568 883 782&

Целые константы в ЭВМ представляются точно.

Текстовая (*строковая*) *константа* — это последовательность любых символов языка (в том числе заглавных букв русского алфавита), ограниченная кавычками. *Длина текстовой константы не более 255 символов*, в QBASIC — *не более 32 567 символов.*

Примеры: «ХОРОШО» «Y = AX + BZ + C»

Переменные. Переменная —это величина, значение которой может изменяться в процессе выполнения программы, обозначается *именем* (*идентификатором*).

Имя переменной — последовательность не более, чем L латинских букв и цифр, начинающаяся с буквы и заканчивающаяся суффиксом, определяющим тип переменной, L = 40 — в QBASICE.

Внимание! Переменная каждого типа может принимать только те значения, которые допустимы для констант соответствующего типа.

В QBASIC, кроме того, используются *вещественные переменные двойной точности* (суффикс #).

Примеры: AB# CDX148#

При отсутствии суффикса переменная считается вещественной — одинарной точности.

В QBASIC используется еще один тип переменной — *длинная целая* (суффикс &).

Примеры: KXDD132& AAA&

12.1.3. Встроенные математические функции

При решении многих прикладных и математических задач на ЭВМ часто возникает необходимость вычисления различных элементарных математических функций (синус, косинус, логарифм и т.д.), что требует многократного составления одних и тех же программ. Во избежание этого для наиболее употребительных функций программы вычисления их записаны в память ЭВМ, в библиотеки программ, а функции включены в состав языков программирования. Обращение к ним максимально упрощено. Такие функции называют *встроенными*. В языках программирования, в частности в Бейсике, кроме математических, используются текстовые встроенные функции. Ниже приведены наиболее употребительные встроенные математические функции, включаемые в состав практически всех версий Бейсика.

Для вычисления каждой из них в программе достаточно указать лишь ее имя и значение аргумента.

Таблица 12.1

Обозначение в математике	Запись в языке Бейсик	Пояснения
$\sin a^*$	SIN (a)	Аргумент в радианах.
$\cos a$	COS (a)	—«—
e^a	EXP (a)	Экспоненциальная функция, где e = 2,7182 ..., $a \le 78$.
$\ln a$	LOG (a)	Функция натурального логарифма (по основанию e), $a > 0$.
$\lvert a \rvert$	ABS (a)	Функция «абсолютная величина»
$\operatorname{arctg} a$	ATN (a)	Функция арктангенса, результат в радианах
$\sqrt{a}$	SQR (a)	

* Здесь a — любое арифметическое выражение.

Примеры записи встроенных функций:

$$e^{x+2} \rightarrow \text{EXP (X + 2)},$$
$$|\ln y| \rightarrow \text{ABS (LOG (Y))}.$$

12.1.4. Выражения

Выражение представляет собой запись, указывающую, какие операции следует произвести над данными, чтобы получить требуемое значение.

Различают два вида выражений — арифметические и текстовые.

Арифметическое выражение. Это символьная запись, составленная из чисел, имен переменных и элементов массивов, встроенных функций, знаков арифметических операций, круглых скобок и имеющая смысл с точки зрения математики.

Арифметическое выражение задает правило вычисления числового значения.

Пример: (5 * X + 3.3 * COS (X))/LOG (Y).

Текстовое выражение. Общий вид такого выражения рассмотрен в 12.1. В частном случае текстовым выражением является текстовая переменная или константа. Примеры: C$, «КОНЕЦ».

В любой версии Бейсика используются арифметические операции +, –, *, /, ^ (возведение в степень).

В состав языка Бейсик УКНЦ, БК 0010 и в QBASIC включены еще две операции, выполняемые над целыми величинами:

a\b — целочисленное деление. Результат операции — частное от деления a на b;

a MOD b — деление по модулю b. Результат операции — остаток от деления a на b.

Примеры: 9\2 = 4 9 MOD 2 = 9 MOD 4 = 1

При записи арифметических выражений необходимо придерживаться следующих правил и ограничений.

I. Все символы выражения записываются в одну строку. Многоэтажные выражения, верхние и нижние индексы запрещены.

II. Два знака арифметических операций не должны располагаться рядом. Знак умножения опускать нельзя.

III. Операции в арифметическом выражении выполняются в порядке старшинства, т.е.:

1) операции внутри скобок;
2) вычисления встроенных функций;
3) возведение в степень;
4) операции умножения и деления;
5) операции целочисленного деления;
6) операции деления по модулю;
7) операции сложения и вычитания.

Операции равного старшинства выполняются по порядку слева направо.

Исключение: A ^ B ^ C = A ^ (B ^ C).

IV. Тип арифметического выражения определяется типом его результата.

а) Операция деления («/») с целыми величинами дает вещественный результат.

б) Выражение может содержать и целые и вещественные величины. Результат такого выражения — вещественная величина.

в) При наличии в арифметическом выражении величин двойной точности результатом будет величина той же точности.

Примеры записи арифметических выражений:

$$\frac{\sqrt{5x^3+3{,}6}}{Y-5}\cdot A+D \rightarrow SQR(5*X\%\hat{}3+3.6)/(Y\#-5)*A+D\&$$

$$\frac{\sqrt[3]{3x-\cos x}}{2{,}5+|Y|^5} \rightarrow (3*X-COS(X))\hat{}(1/3)/(2.5+ABS(Y)\hat{}5)$$

$$\frac{P\sin(x)+\cos^2}{e^2+\ln G} \rightarrow (P\#SIN(X)+COS(X)\hat{}2)/(EXP(2)+LOG(G)).$$

12.1.5. Понятия оператора и программы

Алгоритм решения задачи, записанный на языке Бейсик, называется *программой* на языке Бейсик. Программа указывает ЭВМ, какие операции и в каком порядке нужно выполнить для решения задачи.

Текст программы, как и осмысленный текст на русском языке, состоит из отдельных предложений. В языке Бейсик они называются *операторами*.

Каждый оператор записывается строго определенным образом. Как правило, он содержит имя и данные (в том или ином виде) и указывает, какую операцию и над какими величинами ЭВМ должна выполнить.

Пример оператора: INPUT A, B, C.

Здесь INPUT — имя оператора; A, B, C — список данных (аргументов).

Возможны операторы, которые содержат только имя, их будем называть *операторами без аргументов*, например:

```
STOP RESTORE
```

Программа на языке Бейсик представляет собой последовательность строк. В начале каждой строки ставится ее номер. Строки нумеруются по порядку, обычно с шагом 10, т.е. 10, 20, 30, 40 и т.д. В каждой строке записывается один или несколько операторов, разделяемых символом «:».

В QBASIC нумерация строк необязательна, номер следует указывать для строки, к которой обращается оператор GOTO.

Пример:

```
30 X=20: Y=40: Z=20
40 PRINT X, Y, Z
```

Ограничения: длина строки программы в QBASIC — не более 255 символов (длина строки экрана дисплея — 80 позиций).

Пример программы на языке Бейсик, вычисляющей функцию $y = 2x + 5/c$:

Номер строки	Операторы	Пояснения
10	REM Пример 1	Название программы
20	INPUT X, C	Ввод данных
30	Y = 2*X + 5/C	Вычисление Y
40	PRINT «Y = Y»	Вывод результатов
50	END	Останов

Приведенная программа состоит из пяти строк. Каждая строка содержит один оператор и начинается с номера строки.

12.1.6. Операторы языка Бейсик

В настоящем параграфе мы рассмотрим лишь те операторы, которые необходимы для начального знакомства с программированием.

Оператор присваивания. Он присваивает переменной значение некоторого выражения.

Примеры записи оператора:

```
30 LET R=3*X+2*SIN(Y)^2
40 LET B$="Тетя Маша"
```

Здесь первый оператор вычисляет значение R по формуле

$$R = 3x + 2\sin y.$$

Второй оператор присваивает текстовой переменной B$ значение текстовой константы.

Общий вид оператора:

LET X = A где LET — имя оператора (переводится «пусть»), может быть опущено;

X — имя переменной или элемента массива (см. ниже);

A — выражение, арифметическое или текстовое.

Примеры:

```
50 Y=LOG(X)^2+ABS(Z)
60 LET L$="ПЕТЯ + ДАША"
70 LET C=41.25
```

Здесь первый оператор вычисляет значение y по формуле $y = \ln^2 x + |z|$.

Второй оператор присваивает текстовой переменной L$ значение текстовой константы.

Работа оператора: вычисляет значение выражения A и присваивает его переменной X, т.е. символ «=» означает в данном случае не факт равенства величин X и A, а требование выполнить указанную операцию.

Особенности работы оператора:

1. Если X — целая, а значение A — вещественная величина, то результат операции округляется до ближайшего целого.

Пример: после выполнения операторов

```
20 X=2.9
30 P%=X
```

переменной P% будет присвоено значение 3.

2. Если X — вещественная одинарной точности, а значение A — вещественное двойной точности, то значение A округляется.

3. Допустим оператор вида X=X+A.

В этом случае символ X справа от знака «=» рассматривается как предшествующее, а тот же символ слева от знака «=» как последующее значение переменной X.

Оператор ввода данных INPUT. Он служит для ввода в ЭВМ с клавиатуры значений исходных величин в процессе выполнения программы и размещения их в ячейках памяти, выделенных для этих величин.

Оператор дает возможность решать одну и ту же задачу с различными значениями исходных величин без изменения программы.

Пример: оператор INPUT A, B%, C$ вводит с клавиатуры значения переменных A, B%, C$, т.е. присваивает им конкретные значения.

Общий вид оператора:

```
INPUT <Подсказка>; x1, x2 ..., xi, ..., xn
(список вводимых величин),
```

где INPUT — имя оператора (переводится «ввести»);

<Подсказка> — текстовая константа, служит для пояснения, какие величины нужно ввести и в каком порядке; может быть опущена;

x_i — имя переменной или элемента массива (см. далее) (i = 1, 2, ..., n).

Пример: оператор

```
20 INPUT "ВВЕДИТЕ X%, B!, Q$, R$"; X%, B!, Q$, R$
```

выдает на экран сообщение

```
ВВЕДИТЕ X%, B!, Q$, R$?
```

и ожидает ввода значений четырех величин.

Работа оператора: при выполнении оператора INPUT ЭВМ работает в режиме диалога с пользователем:

1) оператор прерывает выполнение программы, выдает на экран текст подсказки и знак «?» — приглашение к вводу данных,
2) пользователь должен в ответ для каждой переменной x_i оператора ввести с клавиатуры ее значение c_i, разделяя последние запятыми, т.е. на экране появляется список:

$$?c_1, c_2, \ldots, c_n.$$

Пример:

```
ВВЕДИТЕ XX, B!, Q$, R$? 3, 6.5, "Я", ДА
т.е. ci — числовая или текстовая константа, ее тип
должен совпадать с типом величины xi.
```

Исключение. Целая константа c_i записывается без знака «%», а текстовая константа может записываться без ограничивающих апострофов, если она не содержит кавычек, запятой или пробела.

Пример: для оператора INPUT Y%, X, C$, R%, K$ возможен такой список значений:

```
? 30, 5, «ВЕРНО», 30, НЕТ;
```

3) по окончании набора данных пользователь должен нажать клавишу Ввод (Enter) — данные с экрана вводятся в ЭВМ и каждой величине x_i присваивается ее значение c_i, одновременно проверяется соответствие типов этих величин и количество вводимых значений. При этом:
   - если обнаружено несоответствие типов, то выдается сообщение об ошибке, и следует повторно набрать весь список данных:

$$c_1, c_2, \ldots, c_n,$$

   - если введено недостаточное количество значений, то на экран повторно выдается знак «?» и ЭВМ ожидает продолжения ввода,
   - если введены лишние данные, то они игнорируются.

В QBASIC во всех подобных случаях выдается сообщение об ошибке и требуется повторить ввод всех данных.

При правильном вводе данных ЭВМ продолжает выполнение программы.

Оператор вывода PRINT. Он служит для вывода значений величин на экран монитора в процессе выполнения программы.

Пример:

```
10 X=5: B$="ИВАНОВ"
20 PRINT X; B$; "ВСЕ"
```

Оператор PRINT в этом примере выведет на экран значения величин X, B, «ВСЕ», т.е. 5 ИВАНОВ ВСЕ

Общий вид оператора:

PRINT x_1; x_2; ...; x_i; ...; x_n

или

PRINT x_1; x_2; ...; x_i; ...; x_n
(список выводимых величин)

где PRINT — имя оператора (переводится «печатать»); x_i — элемент списка выводимых величин ($i = 1, 2, ..., n$), может являться константой, переменной или элементом массива (см. ниже), выражением, функцией TAB.

Пример записи оператора:

```
40 PRINT "ЗАДАЧА 1"; B; A+B/2; TAB(50); 30
```

Работа оператора: оператор просматривает последовательно элементы списка $x_1, x_2, ..., x_n$, и для каждого x_i,; выводит на экран его значение.

При этом:

- значение числовой величины x_i при $|x_i| < 10^{-7}$ или $|x_i| > 10^7$ выводится в виде числа с плавающей точкой (пример: 2.66 E+08);
- знак «+» перед числом заменяется пробелом;
- целые числа печатаются без знака «%», текстовые константы — без кавычек;
- значения величин выводятся построчно — после заполнения текущей строки происходит автоматический переход к следующей строке.

Расположение выводимых значений в строке определяется разделителем элементов списка:

а) разделитель «;» (точка с запятой). В этом случае числовые значения дополняются в конце одним пробелом; значения текстовых величин выводятся без каких-либо дополнений.

Пример: в результате выполнения операторов

```
10 X=30
20 PRINT 100; -20; X; "ФАРА"; "ОН"; X-80; "ВСЕ"
```

на экране получим текст:

```
100—20 30 ФАРАОН-50 ВСЕ
```

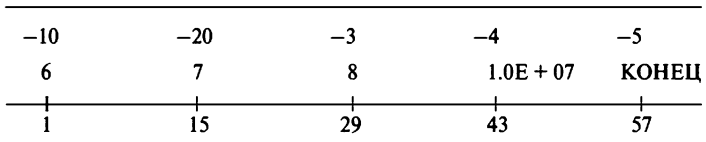
б) разделитель «,» (запятая). В этом случае экран разбивается на пять колонок по 14 позиций в каждой. Значение каждой величины печатается с начала очередной колонки.

Внимание! Ниже приведены примеры использования оператора PRINT в QBASIC. Для версий Бейсика, в которых нумерация позиций в строке экрана начинается с нуля, в этих примерах номера позиций выводимых результатов будут отличаться на единицу.

Пример: после выполнения операторов:

```
30 B=-10
40 H=B 50 D=10000000
60 PRINT B, B+H,-3,-4,-5,6,7,8,0, "КОНЕЦ"
```

На экране получим такие две строки:



Здесь числа 1, 15, 29, 43, 57 означают номера позиций строки экрана (на экран они не выводятся).

Особенности записи и работы оператора PRINT:

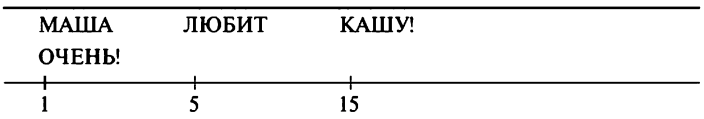
1) Оператор PRINT может записываться без списка величин. В этом случае он выводит на экран пустую строку (строку пробелов);
2) в одном операторе PRINT можно использовать различные разделители («,» и «;»);
3) если в программе содержится несколько операторов PRINT, то работают они как один оператор с общим списком выводимых величин с единственным исключением: при отсутствии в конце оператора PRINT разделителя («,» или «;») следующий за ним в программе оператор PRINT выводит величины с начала новой строки экрана.

Пример: в результате выполнения фрагмента программы

```
40 PRINT "МАША";
50 PRINT "ЛЮБИТ",
```

```
60 PRINT "КАШУ!"
70 PRINT "ОЧЕНЬ!"
```

На экране получим две строки такого вида:



Здесь 1, 5, 15 — номера позиций строки экрана.

Дополнение. Отметим роль операторов INPUT и PRINT в языке Бейсик — они позволяют организовать диалог пользователя с ЭВМ (программой). Приведем фрагмент программы:

```
10 PRINT "Я — ПЭВМ IBM! А КАК ВАШЕ ИМЯ?"
20 INPUT A$
30 PRINT "ЗДРАВСТВУЙТЕ, "; A$;"!"
40 PRINT "КАК ПОГОДА?"
50 INPUT B$
60 PRINT "ДА, ПОГОДА"; B$; "!"
```

Здесь в строке 20 вводится имя пользователя (A$), а в строке 30 печатается слово «Здравствуйте» и имя пользователя. Аналогично в строке 50 вводится слово, определяющее характер погоды («хорошая», «плохая» и т.д.), а в строке 60 печатается фраза, содержащая это слово. Такой диалог, очевидно, можно продолжать долго.

Функция TAB в операторе PRINT. Она служит для вывода (печати) значения величины в определенных позициях строки экрана (листа бумаги принтера).

Пример: оператор PRINT TAB (41); A выведет значение A с 41-й позиции строки экрана.

Общий вид функции:

TAB(n)

где **n** — целое число или арифметическое выражение.

Работа оператора PRINT с функцией TAB:
оператор

```
PRINT TAB(n); X
```

выводит значение X с *n*-й позиции строки.

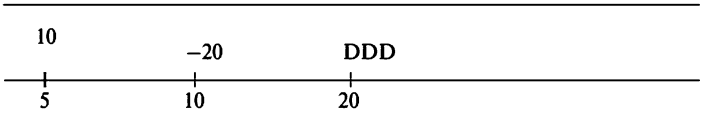
В QBASIC номера позиций строки экрана от 1 до 80.

Ограничение. Если n меньше номера текущей позиции строки, то X выводится с *n*-й позиции следующей строки.

Пример: в результате выполнения операторов

```
10 C=10
20 B=-20
30 PRINT TAB(5); C, TAB(10); B; TAB(20); "DDD"
```

На экране получим строки:



Здесь числа 5, 10, 20 — номера позиций строки экрана.

В этом примере TAB (10) выводит значение B на следующей строке, так как запятая после C переводит курсор в 14-ю позицию.

Вывод информации на печатающее устройство. В QBASIC и прочих версиях Бейсика для вывода информации на печать (в процессе выполнения программы) используется оператор LPRINT. Он записывается и работает аналогично оператору PRINT и также использует функцию TAB.

Оператор REM. Он служит для включения в текст программы пояснений, необходимых для ее понимания. Оператор обращается не к ЭВМ, а к человеку, который будет разбираться в программе.

Пример оператора:

```
10 REM ВЫЧИСЛЕНИЕ ИНТЕГРАЛА. ИВАНОВ.
```

Общий вид оператора:

```
REM <текст>
```

где REM — признак оператора (от английского *remark* — замечание, примечание); <*текст*> — последовательность любых символов языка Бейсик.

Пример: 110 REM БЛОК РАСЧЕТА УДОЙНОСТИ.

В QBASIC, в Бейсике Корвет, УКНЦ и БК 0010 вместо символов «REM» можно использовать символ «‘» (апостроф) и записывать оператор в одной строке с любым другим оператором.

Пример: 110 Y = 4*X^3+6 ‘Расчет прочности.

Оператор STOP. Он прерывает выполнение программы и выдает сообщение

```
ОСТАНОВ В СТРОКЕ n
```

где ***n*** — номер строки оператора STOP. Оператор можно поместить в любой строке программы. Число таких операторов в программе не ограничивается.

Он полезен при отладке программы. После его выполнения можно изменить значения некоторых переменных, командой PRINT... вывести на экран их значения (но не редактировать программу), а затем продолжить работу программы с точки останова командой CONT (в некоторых версиях командой GOTO m).

В QBASIC оператор STOP, вызвавший прерывание работы программы, выделяется в тексте ярким цветом, никаких сообщений не выдается. Для продолжения работы программы с точки прерывания выполнить команду:

```
/Run/Continue(/Запуск/Продолжить) (см. 13.3.2).
```

Оператор END. Он прекращает выполнение программы и является последним ее оператором.

Итак, мы рассмотрели операторы, необходимые для начального знакомства с программированием. Остальные будем вводить по мере необходимости.

Контрольные вопросы

1. Буквы какого алфавита используются в Бейсике?
2. Какие типы данных в Бейсике вам известны?
3. Как обозначаются константы и переменные в Бейсике?
4. Какие типы констант и переменных существуют в Бейсике? Как обозначаются переменные различных типов?
5. Что называется арифметическим выражением в Бейсике? Каков порядок выполнения операций в таком выражении?
6. Что такое оператор в Бейсике? Какие операторы вы знаете?
7. Каково назначение и работа оператора LET?
8. Каково назначение операторов INPUT и PRINT в Бейсике? Как записываются эти операторы?
9. Как обеспечить в программе выдачу результатов на печатающее устройство?
10. Как записать в программе пояснения к тексту программы?

12.2. Начала программирования на языке Бейсик

12.2.1. Общие положения

Говоря о программировании, т.е. о составлении программы решения некоторой задачи, в дальнейшем будем считать, что схема алгоритма задачи задана (исключая, может быть, случаи простейших

(линейных) алгоритмов). Процесс составления программы на языке Бейсик будем рассматривать как процесс перевода алгоритма с языка схем алгоритмов на язык Бейсик, заключающийся в замене каждого блока схемы соответствующим оператором (группой операторов) Бейсика.

Выше мы делили алгоритмы в зависимости от их структуры на линейные, разветвляющиеся, циклические. При нашем подходе к составлению программ каждому виду алгоритмов будут соответствовать программы той же структуры и того же названия. В том же порядке будем и рассматривать их далее, каждый вид в отдельности, выявляя специфику процесса программирования для каждого из них, а пока выясним, как выглядит структура программы на языке Бейсик.

Составные части программы и порядок их расположения определяются схемой (рис. 12.1):

В общем случае структура программы может быть и иной, как будет показано в 10.4, но в простых задачах, с которыми приходится иметь дело на начальной стадии обучения программированию, рекомендуется составлять программы указанной структуры (см. рис. 12.1).

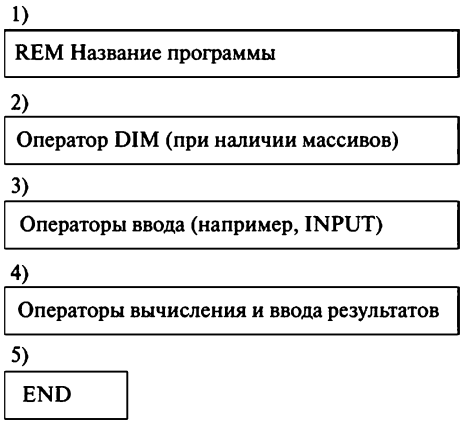


Рис. 12.1

Пример программы на языке Бейсик:

```
10 REM ВЫЧИСЛЕНИЕ ОРБИТЫ СПУТНИКА
20 INPUT X, Y
30 A=SQR(X)*3+2/Y^0.5
40 B=X^2+2*Y^3
50 PRINT A; B
60 END
```

После выполнения этой программы при X=9 и Y=4 данные и результаты будут выведены на экран дисплея в таком виде:

```
?9, 4
 10  209
```

Рекомендации по организации программ

1. Перед оператором INPUT следует записывать оператор вывода имен вводимых величин («подсказку») или использовать возможность вывода «подсказки» оператором INPUT. Это позволяет избежать многих ошибок при вводе данных.

2. В программе следует чаще использовать оператор REM. Это позволяет сделать программу более понятной для пользователя, облегчает, например, внесение изменений в программу. В частности, начало каждого блока программы должно начинаться с описания названия блока или даже назначения его, можно пояснять смысл вводимых и вычисляемых величин и т.д.

3. В операторах PRINT следует предусматривать вывод имен выводимых величин. Это облегчает анализ результатов решения задачи.

4. В программе следует предусматривать вывод не только результатов решения задачи, но и всех исходных данных. Это необходимо, например, при документировании результатов выполнения программы.

5. Не следует записывать в одной строке программы большое число операторов. Это сокращает длину программы, но уменьшает ее наглядность и усложняет исправление ошибок в ней.

Пример: запишем программу предшествующего примера с учетом приведенных рекомендаций.

```
10 REM ВЫЧИСЛЕНИЕ ОРБИТЫ СПУТНИКА
20 PRINT "ВВЕСТИ X, Y"
30 INPUT X, Y
40 REH РАСЧЕТ АПОГЕЯ
50 A=SQR(X)*3+2/Y^0.5
60 REM РАСЧЕТ ДЛИНЫ ОРБИТЫ
70 B=X^2+2*Y^3
80 PRINT "A=":A;" B=";B;" ПРИ X=";X; "Y="; Y
90 END
```

Результатом выполнения программы будут такие сообщения (на экране дисплея):

```
Ввести X, Y
?9, 4
A= 10  B= 209  ПРИ  X= 9  Y= 4
```

Очевидно, эти сообщения более понятны, чем в первом случае.

12.2.2. Линейные программы

Линейной называется программа, являющаяся записью линейного алгоритма. В такой программе все операторы выполняются строго последовательно, т.е. после выполнения каждого из них (кроме END) ЭВМ автоматически переходит к выполнению следующего за ним оператора.

Составление простейших программ

Простейшими будем называть линейные программы, не содержащие массивов. Составление таких программ требует знания ранее рассмотренных операторов, понимания их соответствия блокам схемы алгоритма и производится согласно такому несложному правилу:

рассматриваем блоки схемы алгоритма (считаем, что она дана) по порядку, начиная с первого, и для каждого из них записываем соответствующий ему оператор Бейсика, т.е.

для блока *Начало* — оператор REM с названием программы;

для блока *Ввод* — оператор ввода;

для блока *Процесс* — оператор присваивания;

для блока *Вывод* — оператор вывода;

для блока *Останов* — оператор END.

В этом все правило!

Теперь приведем примеры конкретных программ рассматриваемого вида.

Задача 12.2

Вычислить периметр прямоугольного треугольника, если заданы длины его катетов.

Задача 12.3

Переставить значения величин A и B.

Решение задач 12.2 и 12.3. Эти задачи рассматривались в главе 10, поэтому на рис. 12.2 приведены без пояснений схема алгоритма и программа задачи 10.2, а на рис. 12.3 — то же для задачи 10.3.

На рис. 12.2 и 12.3 стрелками показано соответствие операторов программы блокам схемы алгоритма.

Приведем программы решения еще двух задач. Программа задачи 12.1 иллюстрирует использование величин различного типа. Программа задачи 12.2 демонстрирует организацию вывода данных на печатающее устройство.

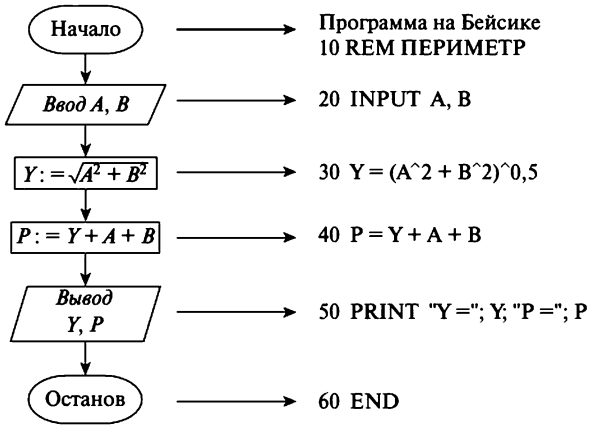


Рис. 12.2

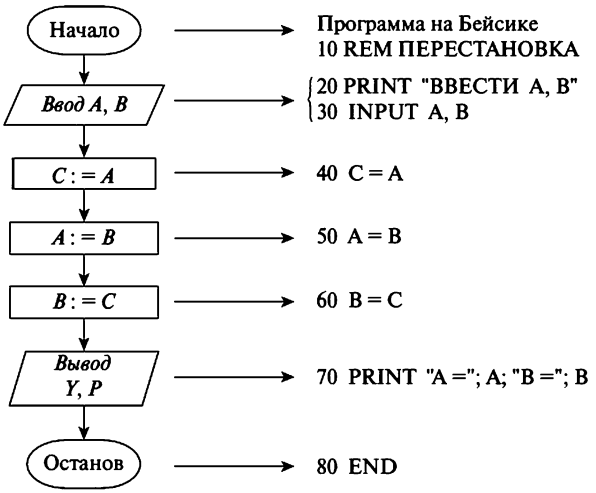


Рис. 12.3

Задача 12.1

Вычислить значения Y и Z по формулам:

$$Y = (3|X| + \sqrt[3]{C}) + \operatorname{tg}\alpha Z,$$
$$Z = 2D - 3B,$$

где D и B — целые величины.

Решение

Исходные данные: D%, B%, C, X, A(α).

Результат: *Y*, *Z%*.

Порядок выполнения операций здесь очевиден, поэтому сразу напишем программу:

```
10 REM ЗАДАЧА 1
20 PRINT "ВВЕСТИ D%, B%, C, X, A"
30 INPUT D%,B%,C, X, A
40 R$="ИВАН0В, 10А КЛ"
50 Z%=2*D%-3*B%
60 Y=(3*ABS(X)+C:(1/3)+SIN(A)/COS(A))*Z%
70 PRINT "Z%=";Z%; "Y="; Y
80 PRINT R$
90 END
```

Задача 12.2

Вычислить значение функции $Y = A^2 + B^2$ и отпечатать (вывести на принтер) значения исходных данных и результатов.

Решение

Программа вычисления функции *Y*:

```
20 REM ВЫВОД НА ПЕЧАТЬ
30 PRINT "ВВЕСТИ A, B"
40 INPUT A, B
50 Y=A^2+B^2
60 LPRINT "ДАННЫЕ:"," A=";A;" B=";B
70 LPRINT
80 LPRINT "РЕЗУЛЬТАТ:"," Y=";Y
90 END
```

Оператор в 30-й строке этой программы выводит текст на экран дисплея, а операторы 60—80 — на принтер. Оператор в 70-й строке печатает пустую строку на листе бумаги.

Линейные программы с массивами

Массивы в языке Бейсик. Напомним определение массива: *массивом* называется упорядоченная совокупность однородных величин, обозначенных каждая одним и тем же именем с различными целочисленными индексами, изменяющимися по порядку.

В Бейсике используются одно- и двумерные массивы (в QBASIC допустимы даже восьмимерные). Они, как и простые переменные, могут быть различных типов: целые, вещественные, текстовые (строковые) и т.д.

Внимание! В Бейсике отсутствуют операции обработки массивов в целом, т.е. операции вида «ввести массив P(1:99)», которые мы привыкли использо-

вать в главах 8 и 9. Для выполнения операции над массивом необходимо перечислить операции, выполняемые над каждым его элементом.

Рассмотрим общий вид элемента массива в Бейсике:

— элемент одномерного массива: <*имя*> (k);

— элемент двумерного массива: <*имя*> (i, j),

где <*имя*> — имя массива, должно отвечать тем же правилам, что и имя простой переменной;

k — *индекс* (номер) элемента одномерного массива, $k \le 0$;

i, j — *индексы* элемента двумерного массива (номера строки и столбца, на пересечении которых он находится), $i \ge 0, j \ge 0$. В QBASIC можно установить начальные значения k, i, j равными 1. Индексы k, i, j могут быть представлены любыми арифметическими выражениями. При вычислении выражения, представляющего индекс, в QBASIC результат округляется до ближайшего целого.

Примеры записи элементов массива:

```
P$(0), C2(101), X(46,5*K+1), T%(N/2, M).
```

В схеме алгоритма те же элементы имели бы такие обозначения: P_0, $C2_{101}$, $X_{46,5k+1}$, $T_{n/2>m}$.

Примечание. Далее в программах на Бейсике нулевые элементы (нулевые строки и столбцы) массивов использовать не будем, чтобы не усложнять работу с индексами.

Внимание! Если в программе используется массив, то он должен быть предварительно объявлен, т.е. ЭВМ должна быть сообщена информация о типе и размерах этого массива с помощью оператора DIM.

Пример: оператор DIM P$(6), B% (4,8) сообщает о наличии в программе текстового P(0:6) и целого B(0:4, 0:8) массивов.

Исходя из информации, содержащейся в операторе DIM, ЭВМ выделяет (резервирует) для каждого массива область памяти требуемого размера.

Общий вид оператора DIM:

— в случае одномерного массива:

```
DIM <имя> (d),
```

— в случае двумерного массива:

```
DIM <имя> (n, m)
```

где DIM — имя оператора (от слова *dimension* — «размерность»); <*имя*> — имя массива; **d**, **n**, **m** — размеры массива, т.е. **d** — номер последнего элемента одномерного массива; **n(m)** — номер последней строки (последнего столбца) двумерного массива.

Размер массива выражается в большинстве версий Бейсика (в том числе в QBASIC) целым числом или целой переменной.

Особенности записи оператора DIM:

1) в одном операторе DIM можно объявлять любое число массивов (см. пример);
2) оператор DIM рекомендуется помещать в начале программы;
3) не следует использовать в программе простую переменную и массив с одним и тем же именем.

Пример: оператор DIM D% (2), A (2,3), K$ (3) сообщает:

- массив D% — одномерный целый, содержащий элементы D%(0), D%(1), D%(2);
- массив K — одномерный текстовой, включает элементы K$(0), к$(1), K$(2), K$(3);
- массив A — двумерный вещественный, включает такие элементы:

A(0,0)	A(0,1)	A(0,2)	A(0,3)
A(1,0)	A(1,1)	A(1,2)	A(1,3)
A(2,0)	A(2,1)	A(2,2)	A(2,3)

т.е. содержат три строки и четыре столбца.

Составление линейных программ с массивами. Прежде всего отметим особенности работы с массивами в программе.

1. Элементы массивов получают значения с помощью операторов ввода или присваивания как простые переменные. При вводе (выводе) массивов в операторах ввода (вывода) перечисляются имена всех вводимых (выводимых) элементов массива.

Пример: программа ввода и вывода массива P (1:3) может иметь такой вид:

```
20 DIM P(3)
30 INPUT P(1), P(2), P(3)
40 PRINT P(1), P(2), P(3)
50 END
```

2. Все массивы можно разделить на два вида:

- *массивы постоянного размера* [например, P(1:7), B(1:4, 1:8)];
- *массивы переменного размера* [например, A(1:*k*); C(1:*m*, 1: *d*)]. В главах 8 и 9 мы использовали оба вида, не делая различий между ними.

В некоторых версиях Бейсика оператор DIM не позволяет объявлять массивы переменного размера, поэтому использование их в программе требует некоторых ухищрений. В версиях Бейсика, рассматриваемых в пособии, таких ограничений нет, допустимо использовать массивы любого вида.

Следует только помнить: *переменные — размеры массивов должны быть определены до обращения к оператору DIM.*

Пример:

```
10 INPUT M
20 DIM X(M)
```

Линейные программы с массивами составляются в соответствии с рассмотренным ранее правилом составления простейших программ, которое можно дополнить одним пунктом — «после оператора REM следует записать в программе оператор DIM». Кроме того, необходимо учитывать только что изложенные сведения о работе с массивами.

Теперь покажем построение рассматриваемых программ на конкретных примерах. Для начала вернемся к задаче 8.5. Схема алгоритма ее приведена на рис. 10.11. Заменив каждый блок этой схемы соответствующим оператором согласно правилу, приведенному в 12.2, и добавив оператор DIM, получим программу, приведенную ниже. После ознакомления с этой программой рекомендуем читателю самому составить программу решения задачи 10.7 и сравнить ее с приведенной ниже:

```
10 REM СДВИГ
20 DIM B(4)
30 PRINT "ВВЕСТИ МАССИВ
   B(4)"
40 INPUT B(1), B(2),
   B(3), B(4)
50 D=B(1)
60 B(1)=B(2)
70 B(2)=B(3)
80 B(3)=B(4)
90 B(4)=D
100 PRINT "МАССИВ B=(";
110 PRINT B(1); B(2);
    B(3); B(4); ")"
120 END
```

```
10 REM СУММА
20 DIM B(3, 3), S(2)
30 PRINT "ВВЕСТИ МАТР. B(3, 3)"
40 INPUT B(1, 1), B(1, 2), B(1, 3)
50 INPUT B(2, 1), B(2, 2), B(2, 3)
60 INPUT B(3, 1), B(3, 2), B(3, 3)
70 S(1) =B(1, 1)+B(1, 2)+B(1, 3)
80 S(2)=B(1, 3)+B(2, 3)+B(3, 3)
90 PRINT "S(1)="; S(1); "S(2)=";
   S(2)
100 END
```

Контрольные вопросы

1. Какой оператор в Бейсике указывает тип и размеры массива?
2. Каково обозначение элементов массива в Бейсике? Каково наименьшее значение индекса элемента массива?

Задачи для самостоятельного решения

В приведенных далее задачах составить схему алгоритма и программу.

1. Вычислить $Z=\sqrt{r^2-\dfrac{2}{\cos x-2}}$ где $r=3c+\ln|b|+4$.

2. Вычислить среднее арифметическое переменных *B*, *C* и *D*.
3. Рабочие Иванов и Петров изготовили за смену A и B деталей соответственно, перевыполнив норму. Определить процент перевыполнения нормы (норма — C деталей в смену).
4. Определить разницу в возрасте невест для двух братьев Пети и Димы. Их возраст *a* и *b* соответственно. Возраст невесты определяется по формуле

$$V = G/2 + 7,$$

где G — возраст жениха.

5. Вычислить значение *y*:

$$y = (3x^3 + 2{,}5a - 10)/|k|,$$

где $x = \ln^2|z| + 0{,}5z$;

$$a = 2\cos r + \frac{1}{\sin r + 2}.$$

Примем $k \neq 0$; $z \neq 0$.

6. Вычислить объем и площадь поверхности цилиндра диаметром *D* и высотой *H*.
7. Вычислить стоимость мебельного гарнитура, содержащего четыре стула, два кресла и один стол. Стоимость изделий соответственно A, B и C руб.
8. Определить среднее арифметическое элементов массива C(1:5).
9. Определить произведение сумм элементов каждой строки матрицы P(1:2, 1:3).
10. Переставить соответствующие элементы первой и второй строк матрицы A(1:2, 1:2).
11. Переставить элементы массива R(l: 6): 1-й элемент и 6-й, 2-й и 5-й, 3-й и 4-й.

12.2.3. Разветвляющиеся программы

Операторы передачи управления

Разветвляющейся называют программу, которая является записью разветвляющегося алгоритма.

В рассмотренных линейных программах операторы выполнялись строго в порядке нумерации строк программы. В разветвляющихся программах часто требуется обеспечить иной порядок выполнения операторов. Для этой цели служат операторы передачи управления. Рассмотрим два из них: оператор безусловного перехода и оператор условного перехода (условный оператор).

Оператор безусловного перехода. Он служит для перехода из одной точки (строки) программы к другой.

Общий вид оператора:

GOTOn,

где GOTO — имя оператора (переводится — «перейти к...»); **n** — номер строки программы.

Работа оператора: оператор обеспечивает переход к строке программы с номером **n**.

Рассмотрим примеры использования оператора.

Пример 1:

```
30 X=X+1
40 Y=(X+2)*3
50 GOTO 30
```

Пример 2:

```
30 X=3*A
50 GOTO 70
60 Y=2*X
70 Z=5*X
```

В примере 1 оператор перехода обеспечивает циклическое выполнение двух операторов. В примере 2 оператор GOTO 70 позволяет обойти 60-ю строку программы.

Пояснение. Современная технология структурного программирования позволяет в принципе обойтись без использования оператора GOTO. Более того, использование этого оператора считается дурным тоном. Однако практика убеждает, что во многих случаях применение GOTO позволяет значительно упростить программу и, самое главное, делает ее более понятной начинающему программисту. Приведем мнение известного программиста Г. Майерса в отношении использования оператора GOTO:

«Позиция в отношении оператора GOTO должна быть следующей: избегать использования GOTO всюду, где это возможно, но не ценой ясности программы ... в определенных случаях GOTO желательнее других вариантов» [48].

Логические выражения в Бейсике. Далее при рассмотрении условного оператора нам потребуются понятия «отношение» и «логическое выражение», поэтому вспомним их. В главе 1 мы рассмотрели их содержательный смысл; в языке Бейсик они имеют тот же смысл, но записываются по правилам этого языка.

Отношение в Бейсике — это конструкция вида

A@B,

где **A** и **B** — либо арифметические выражения (оба), либо текстовые; **@** — один из знаков операции отношения:

<, >, <=, >=, =, <> (не равно).

Примеры:

```
(LOG(X)+2)<=(ABS(Y-3)+K)  "a" < "d"
"Да" <> "Нет"  "Хорошо" < "Тигр"
```

Логические операции в Бейсике изображаются служебными словами:

AND (**и**); O*R* (**или**); NOT (**не**).

Логическое выражение в Бейсике — это отношение либо формула, составленная из отношений, круглых скобок и символов логических операций.

Логическое выражение может быть либо *истинным*, либо *ложным*, соответственно говорят, что логическое выражение принимает значение «истина» или «ложь». В QBASIC истинное выражение получает значение −1, ложное 0, в чем несложно убедиться, если ввести в машину следующую программу:

```
30 U=10
40 PRINT "(ЗНАЧЕНИЕ U<20)=", U<20
40 END
```

Результат:

(Значение U <20) = −1

При U=100 тот же оператор PRINT выдаст:

(Значение U <20) = 0

Вычисление значения логического выражения. В случае сложных выражений подобного типа истинность или ложность их далеко не очевидна. Для выявления истинности выражения требуется скрупулезно выполнить все операции, включенные в состав выражения, аналогично тому, как вычисляется значение арифметического выражения. Правила вычисления значения логического выражения:

при вычислении значения логического выражения операции выполняются в таком порядке, вычисляются:

1) значения арифметических выражений;
2) значения отношений;
3) все операции NOT, затем AND и в последнюю очередь все операции OR.

Пояснение. В математике значения «Истина», «Ложь» принято обозначать символами «1» и «0», соответственно, используем и мы эти обозначения.

Пример: Вычислить значение логического выражения при $X = 3$ и $Y = -2$:

$$(X < 7) \text{ AND } (X - Y < X * X) \text{ OR } (\text{ABS } (Y) > 3).$$

Подставим в выражение значения X и Y и выполним все требуемые операции в порядке, определяемом приведенным выше правилом:

$$(3 < 7) \text{ AND } (3 - (-2) < 3 * 3) \text{ OR } (\text{ABS } (-2) > 3) =$$
$$= (3 < 7) \text{ AND } (5 > 9) \text{ OR } (2 > 3) = 1 \text{ AND } 1 \text{ OR } 0 = 1 \text{ OR } 0 = 1 \text{ («Истина»).}$$

То есть заданное логическое выражение истинно при указанных значениях X и Y.

Условный оператор. Он обеспечивает в программе проверку условий и организацию ветвлений.

Пример записи оператора:

```
IF X<10 THEN Z=10 ELSE Y=20
```

Данный оператор проверяет истинность условия «X<10», после чего вычисляет значение Z, если условие истинно, и значение Y, если условие ложно.

Общий вид условного оператора. Существует два варианта оператора:

Вариант 1 (полная форма записи оператора):
IF <*условие*> THEN **P1** ELSE **P2**
Вариант 2 (сокращенная форма записи оператора):
IF <*условие*> THEN **P1**

Рис. 12.4

Здесь IF (если), THEN (то), ELSE (иначе) — служебные слова языка Бейсик.

<*условие*> — логическое выражение, т.е. отношение или составное условие.

P1 (P2) — оператор или группа операторов; в Бейсике УКНЦ и БК 0010 — один оператор любого вида. Приведем примеры записи оператора.

Пример 1:

```
30 IF X<Y THEN A=X: B=Y ELSE A=Y: B=X
40 PRINT "A="; A," B=";B
```

Пример 2:

```
20 Y=COS(X): Z=SIN(X)
30 IF (2<=X) AND (X<=10) THEN Y=SIN(X): Z=Y*X
40 R=Y+Z^2
```

В примере 2 проверяемое условие: 2≤X≤10; P1 включает два оператора: Y=SIN(X) и Z = Y*X.

Работа условного оператора. Работа каждого варианта оператора определяется соответствующей схемой.

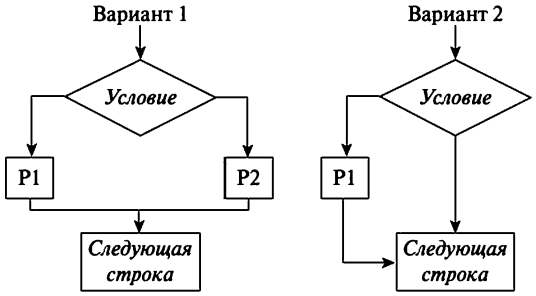


Рис. 12.5

Подчеркнем, после выполнения оператора(ов) P1 или P2 во всех случаях обеспечивается переход к следующей строке.

Поясним работу условного оператора в приведенных выше примерах. В примере 1 проверяется истинность условия «X<Y» и в любом случае величинам A и B присваиваются значения, удовлетворяющие условию: «A<B».

В примере 2 использована сокращенная форма условного оператора. Здесь в случае истинности условия 2≤X≤10 выполняются последовательно операторы:

$$Y = SIN(X) \text{ и } Z=Y*Y,$$

в противном случае выполнятся операторы строки 40.

Особенности записи и работы условного оператора:

1) условный оператор может занимать лишь одну строку;
2) в условном операторе проверка равенства вещественных величин не имеет смысла ввиду приближенного представления подобных величин в ЭВМ.

Поэтому, если X, Y вещественные, то условие X=Y можно, например, заменить таким:

$$ABS\ (X - Y) < 0.001.$$

Здесь 0.001 — интересующая нас степень близости *X* и *Y*, определяемая конкретным смыслом задачи, в частности точностью представления переменных *X* и *Y*.

Рекомендации по составлению разветвляющихся программ. При составлении указанных программ исходим из того, что схема алгоритма решения задачи задана, надо лишь перевести алгоритм на язык Бейсик. Этот процесс наиболее прост, если схема алгоритма имеет вид схемы, изображенной на рис. 12.15. В таком случае поступаем следующим образом.

1. Преобразуем условия логических блоков схемы алгоритма так, чтобы выполнялось правило 1:
   «Выход *Нет* логического блока должен быть связан с нижележащим блоком»;
   для этого используем следующий прием:
   «Условие логического блока заменяем на противоположное, одновременно выходы *Да* и *Нет* блока меняем местами».
2. Записываем оператор REM с именем программы и оператором DIM, если в программе будут использоваться массивы.
3. Для каждого блока схемы по порядку, начиная с первого, записываем соответствующий оператор (группу операторов) Бейсика. При этом для каждого логического блока и соответствующей ему ветви алгоритма запишем одну строку программы вида:

$$\text{IF } \langle \textit{условие} \rangle \text{ THEN P1: GO TO n} \qquad (12.1)$$

где P1 — оператор или группа операторов, составляющих ветвь.

В итоге получаем программу с четкой, «изящной» структурой, идентичной структуре схемы алгоритма.

Однако не всегда операторы каждой ветви помещаются в одной строке программы, и тогда четкость структуры нарушается. В этом случае полезен прием, позволяющий уменьшить длину оператора:

«Если оператор P1 некоторой ветви содержит оператор LET вида R=Z и выражение Z достаточно громоздко, то в линейной части программы, предшествующей этой ветви, следует записать, например, оператор вида Y=Z. Тогда оператор R=Z можно заменить таким: R=Y, содержащим все три символа!»

Аналогично, если P1 включает оператор PRINT, содержащий текстовую константу, то ее можно заменить текстовой переменной, а значение этой переменной следует присвоить также в начале программы, в линейной ее части. Например, оператор

```
40 PRINT "ВВЕДИТЕ ДАТУ"
```

можно заменить двумя короткими:

```
20 R$="ВВЕДИТЕ ДАТУ"
...
40 PRINT R$
```

Второй прием для достижения той же цели применим и в том случае, когда ветвь содержит несколько операторов:

«Все операторы ветви включаем в одну подпрограмму (см. ниже) и помещаем в конце программы, начиная со строки с номером ns. Тогда оператором P1 будет один оператор вызова подпрограммы вида GOSUB ns». И это, повторяем, лишь тогда, когда схема имеет вид схемы, представленной на рис. 12.15. В противном случае следует:

- разбить схему на фрагменты («куски»), содержащие каждый по одному логическому блоку, — это будут элементарные по сложности фрагменты;
- каждый фрагмент заменить соответствующей программой, записать их последовательно и объединить операторами GOTO в единую программу.

При этом не должно нарушаться правило 2:

«В программе разрешается выполнять переход только по направлению сверху вниз (исключение возможно лишь при организации цикла с помощью операторов GOTO и IF)».

Это правило преследует цель облегчить чтение и понимание программ, особенно сложных, которые в этом случае обычно и без того весьма запутанны.

Примеры составления разветвляющихся программ. Обратимся в первую очередь к задачам, рассмотренным в главе 8, для которых составлены схемы алгоритмов.

Пример: составить программу решения задачи 8.13. Схема алгоритма ее приведена на рис. 8.25, *а*.

Решение (программа задачи 8.13).

```
10 REM РАЗВЕТВЛЕНИЕ
20 PRINT "ВВЕСТИ X, Z"
30 INPUT X, Z
40 A=X*Z
50 IF X>=10 THEN Y=A: GOTO 80
60 IF X<=5 THEN Y=A: GOTO 80
70 Y=X+Z
80 PRINT "Y=";Y
90 END
```

В этой программе мы преобразовали логические блоки, используя прием 1, и сократили длину строк 50 и 60, используя прием 2.

Задача 12.3

Составить программу нахождения значения наибольшего элемента главной диагонали матрицы A (1:3, 1:3) в соответствии со схемой алгоритма рис. 10.22.

Решение (программа задачи 12.3).

```
5 REM МАКСИМУМ
10 DIM A(3, 3)
20 PRINT "ВВЕСТИ МАТР. A(3, 3)"
30 INPUT A(1, 1), A(1, 2), A(1, 3)
40 INPUT A(2, 1), A(2, 2), A(2, 3)
50 INPUT A(3, 1), A(3, 2), A(3, 3)
60 IF A(1, 1)>A(2, 2) THEN GOTO 90
70 IF A(2, 2)>A(3, 3) THEN Y=A(2, 2): GOTO 110
80 GOTO 100
90 IF A(1, 1)>A(3, 3) THEN Y=A(1, 1): GOTO 110
100 Y=A(3, 3)
110 PRINT "Y=";Y
120 END
```

Здесь блок Y=$a_{3,3}$ является общим для двух логических блоков (см. рис. 10.22). В программе он размещен в соответствии с правилом 2.

Задача 12.4

Составить программу ввода величины t — времени суток и выдачи текста:

«Вы уже проснулись?», если t < 10;

«Не пора ли обедать?», если t = 12;

«Еще не вечер!», если t ≥ 18;

«Как работается?» в остальных случаях.

Рекомендуем учащимся для данной задачи самим составить схему алгоритма, совпадающую по структуре со схемой, представленной на рис. 8.15, а также и программу задачи с учетом приведенных выше рекомендаций и сравнить ее (программу) с приведенной ниже:

```
10 REM ВРЕМЯ
20 PRINT "КОТОРЫЙ ЧАС?"
30 INPUT T
40 C$ = "ВЫ УЖЕ ПРОСНУЛИСЬ?"
50 S$ = "НЕ ПОРА ЛИ ОБЕДАТЬ?"
60 IF T<10 THEN PRINT C$: GOTO 100
70 IF T = 12 THEN PRINT S$: GOTO 100
80 IF T> = 18 THEN PRINT "ЕЩЕ НЕ ВЕЧЕР!": GOTO 100
90 PRINT "КАК РАБОТАЕТСЯ?"
100 END
```

Задачи для самостоятельного решения

Для каждой из приведенных задач составить схему алгоритма и программу.

1. Составить программу ввода значения температуры воздуха t и выдачи текста «Хорошая погода!», если t > 10°, и текста «Плохая погода!», если t ≤ 10°.

2. Составить программу ввода оценки P, полученной учащимся, и выдачи текста:

«Молодец!», если $P = 5$;

«Хорошо!», если $P = 4$;

«Лентяй!», если $P = 3$.

3. Заданы длины сторон треугольника — A, B, C. Определить, является ли треугольник равнобедренным.

4. Определить сумму элементов массива $C(1:3)$, значения которых больше нуля.

5. Элементы каждой строки матрицы $A(1:2, 1:2)$ переставить в порядке возрастания их значений.

6. Элементы массива $C(1:3)$ вывести в порядке возрастания их значений.

Кроме приведенных задач, рекомендуем вновь рассмотреть задачи главы 8.3.2 и составить для каждой из них программу, использующую составленную ранее схему алгоритма.

12.2.4. Циклические программы с операторами передачи управления

Составление циклических программ. *Циклической* называется программа, которая является записью циклического алгоритма.

Любой циклический алгоритм можно рассматривать как частный случай разветвляющегося алгоритма, а потому любую циклическую программу можно составить с помощью тех же операторов (в частности, GOTO и IF), которые используются в разветвляющихся программах. Те же правила и перехода от алгоритма к программе, т.е. рассматриваем схему алгоритма сверху вниз начиная с первого блока (*Начало*) и записываем для каждого блока соответствующий ему оператор или группу операторов.

Особенно прост процесс перехода к программе от схем алгоритмов, построенных с использованием типовой схемы, приведенной на рис. 12.28, *а*, и метода алгоритмизации, изложенного в главе 9. На подобные схемы мы и ориентируемся. В таком случае блок проверки окончания цикла и блок изменения аргумента цикла будут изображаться оператором-строкой вида:

```
IF <условие> THEN P1: GO TO n
```

Проиллюстрируем изложенное.

Задача 12.5

Составить программу, соответствующую схеме алгоритма рис. 12.6.

Решение

Записав для каждого блока схемы соответствующий оператор (показан стрелками на рис. 12.6), получим программу решения задачи.

Приведем еще ряд примеров построения циклических программ.

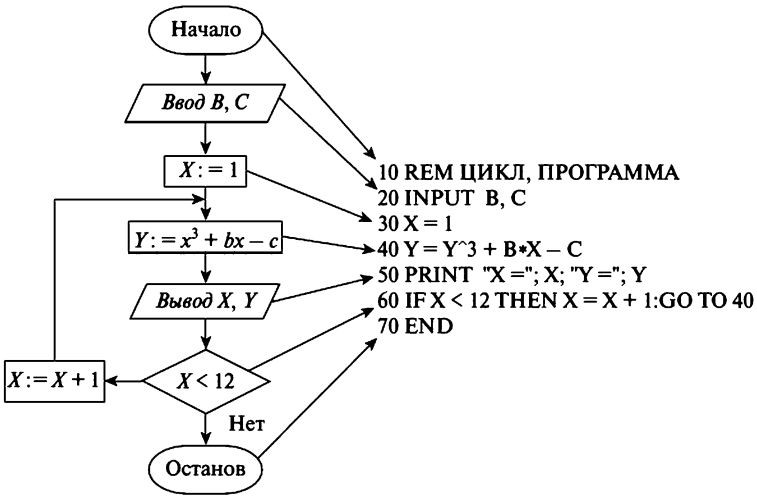


Рис. 12.6

Задача 12.6

Ввести (вывести на экран) одномерный массив P(1:15).

Пояснение. Бейсик может работать лишь с отдельными элементами массивов. Поэтому ввод (вывод) массива заключается в многократном выполнении операции Ввод (Вывод) одного элемента массива, что и отражает схема алгоритма рис. 12.7.

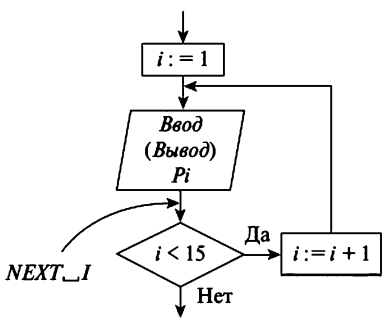


Рис. 12.7

Решение

Записав для каждого блока схемы рис. 12.7 по порядку соответствующий ему оператор, получим программу ввода одномерного массива такого вида:

```
10 REM ВВОД МАССИВА
20 DIM P(15)
```

```
30 I=1
40 PRINT "ВВЕСТИ"; I; "-Й ЭЛЕМЕНТ"
50 INPUT P(I)
60 IF I<15 THEN I=I+1: GOTO 40
70 END
```

Строка 40 программы обеспечивает вывод на экран номера вводимого элемента («Подсказка»).

Программа вывода массива должна отличаться от программы ввода массива наличием в строке 50 оператора «PRINT P(I);» вместо «INPUT P(I)» и отсутствием оператора PRINT в строке 40. Надеемся, что читатель сам сможет составить эту программу.

Блоки ввода-вывода массивов на Бейсике существенно «загромождают» программы, поэтому будет полезен следующий прием, позволяющий уменьшить размеры этих блоков.

Рекомендация[1]. Известно, что во многих задачах можно совмещать процесс вычислений и вывода результатов за счет вывода в каждом цикле циклического алгоритма (программы) порции результатов, полученной в этом цикле (см. задачу 12.5).

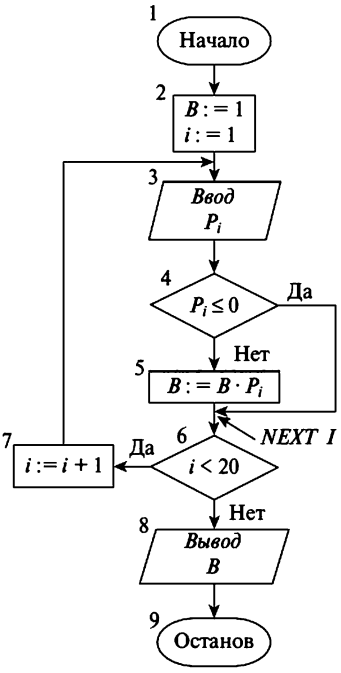


Рис. 12.8

Аналогично при решении многих задач можно совмещать процесс ввода данных с их обработкой (и даже с выводом результатов), т.е. в каждом цикле вводить порцию данных и в этом же цикле обрабатывать ее (и выводить полученные результаты). Этот подход иллюстрирует программа решения следующей задачи.

Задача 12.7

Вычислить произведение элементов массива P(1:20), больших нуля.

Решение

Схема алгоритма и программа задачи приведены на рис. 12.8.

В этой программе совмещается в каждом цикле ввод одного элемента массива с его обработкой.

[1] Рекомендация несколько противоречит модульному принципу организации программы, но для простых задач, где вся программа состоит из одного модуля, она вполне применима и полезна.

```
10 REM ПРОИЗВЕДЕНИЕ
20 DIM P(20)
30 B=1: I=1
40 PRINT "ВВЕСТИ"; I; "-Й ЭЛЕМЕНТ"
50 INPUT P(I)
60 IF P(I) <=0 THEN GOTO 80
70 B=B*P(I)
80 IF I<20 THEN I=I+1: GOTO 40
90 PRINT "B="; B
100 END
```

В разных задачах возможны разные варианты такого совмещения. Так, в задаче 12.17 можно было бы вводить и обрабатывать в каждом цикле по два элемента массива A, а в задаче 12.21 — по одной строке матрицы A (после небольшого изменения алгоритмов этих задач).

Далее с целью закрепления материала рекомендуем читателю вернуться к задачам для самостоятельного решения в 12.4 и для каждой из них составить циклическую программу.

12.2.5. Подпрограммы

Основные понятия и определения. Понятие «подпрограмма» — одно из важнейших в программировании. Использование подпрограмм — наиболее мощное средство повышения эффективности применения ЭВМ и снижения затрат на разработку программ.

Подпрограммой (*п/п*) называется участок программы, оформленный определенным образом, к которому можно обращаться из разных точек программы любое число раз. При этом п/п может решать каждый раз одну и ту же задачу с разными значениями исходных данных. Программу, в которой используется п/п, называют *рабочей*.

Целесообразно использовать п/п тогда, когда в процессе решения задачи многократно встречается некоторая подзадача.

При этом возможны два случая:

1) указанная подзадача является типовой, часто встречающейся задачей — математической, технической и т.д. Пример подобной задачи — вычисление определенного интеграла, решение уравнения с одним неизвестным и т.д. В настоящее время для большинства подобных задач составлены программы (оформленные обычно, как п/п). Они публикуются в весьма многочисленных сборниках программ, журналах.[1]

В этом случае задача программиста — найти нужную п/п и включить ее в текст рабочей программы;

[1] Таким сборником программ и п/п на языке Бейсик является книга В.П. Дьяконова [9].

2) Для упомянутой подзадачи не существует (не удалось найти) готовой программы (п/п). В этом случае программист должен:

- составить программу решения подзадачи,
- оформить ее, как п/п,
- включить п/п в текст рабочей программы.

Использование п/п позволяет существенно уменьшить объем рабочей программы.

С термином «подпрограмма» тесно связано еще одно понятие: *оператор вызова* п/п — оператор, обеспечивающий исполнение п/п. Он помещается в той точке рабочей программы, где необходимо выполнить п/п.

Пример: покажем, как будет выглядеть на языке Бейсик п/п (назовем ее «Функ»), вычисляющая функцию $Y = AX^2 + B$.

Для этой п/п исходные данные: A, X, B; результат: Y. Подпрограмма «Функ»:

```
100 REM SUB ФУНК
110 Y = A*X^2+B
120 RETURN
```

Приведем теперь пример рабочей программы (с п/п «Функ»), вычисляющей значение функции Z:

$$Z = 3\mathrm{Sin}^2 R + 5.$$

Для этой программы исходные данные: R; результат: Z.

Очевидно, для вычисления Z можно использовать нашу п/п, если принять:

$$A = 3, \quad B = 5, \quad X = \sin R.$$

Именно эти операции и реализует в первую очередь рабочая программа, после чего выполняет п/п. В этом ее суть:

```
10 REM РАБОЧАЯ ПРОГРАММА
20 REM ВВОДИМ ДАННЫЕ ДЛЯ РАБОЧЕЙ ПРОГРАММЫ
30 INPUT R
40 REM ВЫЧИСЛЯЕМ ДАННЫЕ ДЛЯ П/П
50 A=3
60 B=5
70 X=SIN(R)
80 REM ПЕРЕХОДИМ К П/П
90 GOSUB 130
100 Z=Y
110 PRINT "Z="; Z
120 STOP
130 REM SUB ФУНК (НАЧАЛО П/П)
140 Y=AX^2+B
```

```
150 REM ВОЗВРАЩАЕМСЯ К СТРОКЕ 100
160 RETURN
170 END
```

Пояснение. Назначение операторов в строках 30—70 программы, видимо, уже понятно. Оператор GOSUB обеспечивает переход к п/п, к ее 1-й строке с номером 130. После завершения работы п/п оператор RETURN осуществляет переход к строке 100 рабочей продрам -мы, и выполнение ее продолжается. Оператор строки 100 запоминает результат, выданный подпрограммой.

Итак, мы показали смысл конструкций языка Бейсик, связанных с понятием «подпрограмма». Теперь рассмотрим вид и работу этих же конструкций в общем случае.

Подпрограмма. Она имеет структуру, изображенную на рис. 12.9.

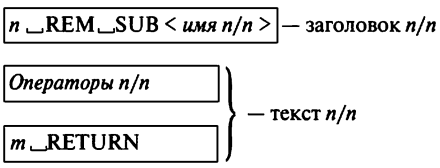


Рис. 12.9

где n(m) — номер первой (последней) строки п/п; SUB — признак п/п (от англ. *subroutine* — подпрограмма), позволяет выделить заголовок п/п из числа прочих операторов REM (необязательный элемент); *<имя п/п>* — любой текст; *<операторы п/п>* — любые операторы языка Бейсик (кроме END); RETURN — оператор без аргументов, обеспечивает возврат и рабочую программу после выполнения п/п (переводится «возврат»).

Особенности записи подпрограммы.

1. Подпрограмма помещается обычно в конце рабочей программы (рис. 12.10).
2. В подпрограмме может содержаться обращение к другой п/п.

Оператор вызова п/п. О назначении оператора уже было сказано. Общий вид оператора:

m GOSUB *n*

где *m* — номер строки; GOSUB — имя оператора (означает «перейти в п/п); *n* — номер строки-заголовка п/п.

Пример:

```
50 GOSUB 200
```

Работа оператора: оператор запоминает номер строки программы, следующей за ним по тексту, и переходит к выполнению п/п начиная с *n*-й строки.

Оператор RETURN. Он обеспечивает возврат из п/п в рабочую программу непосредственно к строке, следующей за оператором GOSUB.

Рабочая программа. Общий вид ее структуры представлен на рис. 12.10.

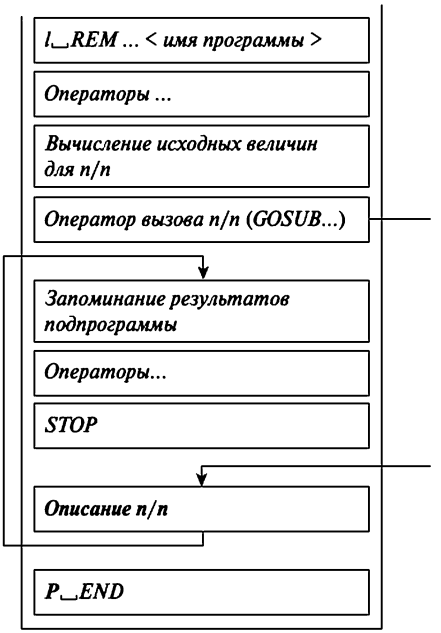


Рис. 12.10

Здесь стрелками показан порядок выполнения п/п и операторов рабочей программы.

Подчеркнем, что в рабочей программе необходимо определить значения всех исходных для п/п величин до обращения к п/п (до оператора GOSUB), а после него запомнить результаты и тем избежать их уничтожения в случае повторного обращения к п/п.

Рассмотрим еще один пример использования п/п. Для этого вновь обратимся к задаче 12.3: задана матрица $A(1{:}3,\ 1{:}3)$; определить $Y = \max\{a_{11}, a_{22}, a_{33}\}$.

Составим второй вариант программы решения этой задачи в соответствии со схемой алгоритма на рис. 10.24. Из этого алгоритма следует, что для получения результата достаточно дважды решить подзадачу: «определить значение наибольшей из двух величин», что может быть выполнено подпрограммой (назовем ее MAX-2) нахождения $Q = \max\{N, M\}$.

П/п MAX-2:

```
10 REM SUB MAX-2
20 IF N>M THEN Q=N: GOTO 40
30 Q=M
40 RETURN
```

Для п/п MAX-2 исходные данные: N, M; результат: Q. Используя эту п/п, составим программу следующего вида решения всей задачи [операторы DIM и ввод матрицы A мы опустили — они выглядят так же, как и в первом варианте этой программы (см. 10.2.3)]:

```
40 REM ВЫЧИСЛЯЕМ ДАННЫЕ ДЛЯ П/П
50 N=A(1,1): M=A(2,2)
60 REM ПЕРЕХОДИМ К П/П MAX-2
70 GOSUB 180
80 REM ФИКСИРУЕМ РЕЗУЛЬТАТ РАБОТЫ П/П
90 R=Q
100 REM ВЫЧИСЛЯЕМ ДАННЫЕ ДЛЯ П/П
110 N=R: M=A(3,3)
120 REM ПЕРЕХОДИМ К П/П MAX-2
130 GOSUB 180
140 Y=Q
150 PRINT "Y=";Y
160 STOP
170 REM СТРОКИ 180–210 – ТЕКСТ П/П MAX-2
180 REM SUB MAX-2
190 IF N>M THEN Q=N: GOTO 210
200 Q=M
210 RETURN
220 END
```

Контрольные вопросы

1. Что такое подпрограмма?
2. Какие основные понятия связаны с понятием «подпрограмма»?
3. Как записывается оператор вызова п/п? Каково его назначение?
4. Каким образом обеспечивается возвращение в рабочую программу из п/п при выполнении п/п?
5. Как передаются значения исходных данных в п/п?

Задачи для самостоятельного решения

1. Дана матрица $B(1{:}3, 1{:}3)$. Вычислить произведение сумм элементов 1-й и 3-й строк матрицы.

Пояснение. Составить п/п вычисления суммы элементов k-й строки матрицы A и использовать ее дважды: при $k = 1$ и $k = 3$.

2. Переставить элементы массива $P(1{:}6)$ — 1-й с 6-м, 2-й с 3-м, 4-й с 5-м, составив и используя п/п перестановки двух элементов массива P с номерами n и m.

3. Найти площадь кругового кольца с заданным внешним R и внутренним r радиусами, используя п/п вычисления площади круга.

4. Задан массив $D(1{:}\ 6)$. Определить сумму каждой из следующих троек элементов массива: $d_1 \div d_3$, $d_2 \div d_5$, $d_2 \div d_4$.

Пояснение. Составить п/п вычисления суммы трех последовательно расположенных элементов массива D с номерами от k до m. Решение задачи потребует трехкратного обращения к п/п.

5. На числовой оси расположены пять точек с координатами b_1, b_2, b_3, b_4, b_5, и точка A с координатой $x(b_1 \le x \le b_5)$. Определить, лежит ли точка A на одном из отрезков $[b_1, b_2]$ или $[b_3, b_4]$.

Пояснение. Составить п/п, которая определяет, лежит ли точка A на отрезке $[c, d]$ и, при выполнении условия, присваивает текстовой переменной, например, R значение «Да».

В начале рабочей программы переменной R присвоить значение «Нет».

12.3. Программирование задач с использованием оператора цикла и файлов

12.3.1. Циклические программы с операторами FOR и NEXT

Из предыдущего раздела читатель должен был уяснить, что любой циклический алгоритм можно записать на языке Бейсик с использованием операторов IF и GOTO. Однако в Бейсике есть специальные операторы для этой цели, позволяющие создавать более компактные программы — операторы FOR и NEXT. Правда, они пригодны для описания не всех видов циклических алгоритмов.

Операторы FOR и NEXT

Пример использования операторов:

```
40 FOR X=2 TO 12 STEP 2
50 Y=X^3+B*X-C
60 PRINT "X="; X, "Y=";Y
70 NEXT X
```

Приведенные операторы описывают циклический участок алгоритма (рис. 12.6).

Общий вид операторов (рис. 12.11).

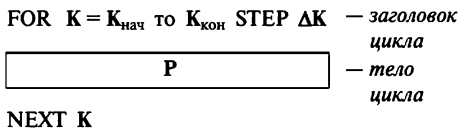


Рис. 12.11

Здесь FOR — имя оператора (переводится «для»); **K** — переменная (обычно целого типа), называемая *переменной цикла*; $\mathbf{K}_{нач}$, $\mathbf{K}_{кон}$, $\Delta\mathbf{K}$ — начальное, конечное значения и шаг изменения величины **K** соответственно, могут быть представлены любыми арифметическими выражениями; TO, STEP — символы языка Бейсик (переводятся — «до», «шаг»); **P** — оператор или группа операторов языка; NEXT — имя оператора (переводится «следующий»).

Пример: в приведенном выше фрагменте программы переменная цикла — X, $\mathbf{K}_{нач}$ — 2, $\mathbf{K}_{кон}$ — 12, $\Delta\mathbf{K}$ — 2.

Исключение. Если $\Delta\mathbf{K} = 1$, то конструкцию STEP 1 можно опустить.

Пример:

```
40 FOR N=1 TO 100
50 Y=N^2
60 PRINT N; "Y=";Y
70 NEXT N
```

Приведенные операторы вычисляют квадраты всех целых чисел от 1 до 100.

Работа операторов. Совместная работа операторов FOR и NEXT определяется схемой рис. 12.12, где

$$<\text{условие}>=\begin{cases}\mathbf{K} \le \mathbf{K}_{кон}, \text{если } \Delta\mathbf{K}>0;\\ \mathbf{K} \le \mathbf{K}_{кон}, \text{если } \Delta\mathbf{K}<0.\end{cases}$$

Иначе говоря, операторы FOR и NEXT обеспечивают изменение значения **K** от **K**, до **K** u с шагом $\Delta\mathbf{K}$ и выполнение оператора (операторов), заключенных между FOR и NEXT, при каждом значении **K**.

Ограничение. Изменять значение величин $\mathbf{K}_{нач}$, $\mathbf{K}_{кон}$, $\Delta\mathbf{K}$ в процессе выполнения операторов FOR и NEXT не рекомендуется.

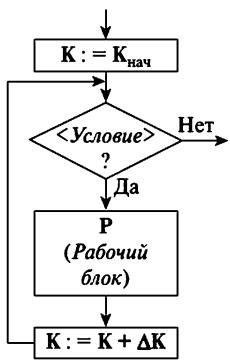


Рис. 12.12

Сопоставляя рис. 12.11 и 12.12, нетрудно установить связь схемы циклического алгоритма с изображающими его на Бейсике операторами FOR и NEXT и заметить, что **P** — это оператор (операторы), описывающий операции рабочего блока схемы алгоритма в программе, NEXT **K** — оператор, помещаемый в программе непосредственно после оператора (операторов) **P**; заголовок цикла — содержит всю информацию, касающуюся переменной **K** и «разбросанную» по остальным трем блокам схемы алгоритма.

Эти факты позволяют формально подойти к составлению рассматриваемых программ.

Оператор WHILE...WEND

Такой оператор используется в QBASIC. Сразу отметим — это очень полезный и удобный оператор.

Общий вид оператора:

WHILE <*условие*> — *заголовок цикла*
P — *тело цикла*
WEND

где WHILE — символ языка Бейсик (переводится «пока»); <*условие*> — условие повторения цикла, логическое выражение (отношение); **P** — группа операторов языка, включает рабочий блок циклического алгоритма и блок изменения переменных; WEND — символ языка, завершает тело цикла.

Присваивание переменным начальных значений выполняется до цикла.

Примеры записи оператора:

Программа 1:

```
30 X=2
40 WHILE X<=12
50 Y=XY^3+B*X-C
60 PRINT "X="; X," Y-"; Y
70 X=X+2
80 WEND
```

Программа 2:

```
50 N=1: Y=1
60 WHILE Y<=200
70 Y=T^2
80 PRINT N; "Y="; Y
90 N=N+1
100 WEND
```

Работа оператора: выполняет все операторы тела цикла пока истинно <*условие*>.

Особенности оператора. Его можно использовать:

- при неизвестном заранее количестве повторений цикла;
- если закон изменения параметра цикла выражается произвольной формулой.

Составление программ с использованием операторов FOR и NEXT

В этом случае, как и ранее, исходим из того, что схема алгоритма решения задачи задана.

Рекомендации. Для перехода от схемы алгоритма к программе, использующей указанные операторы, можно применять следующий формальный подход:

1) в схеме алгоритма формулы, содержащиеся в блоке «изменение аргументов» каждого цикла (кроме формулы вычисления переменной цикла), переносим в конец рабочего блока соответствующего цикла и помечаем выход рабочего блока каждого цикла символом NEXT **K**, подразумевая под **K** имя переменной этого цикла;
2) записываем оператор REM с именем программы и, если в программе будут использоваться массивы, оператор DIM;
3) просматриваем схему алгоритма по порядку сверху вниз и для каждого блока записываем соответствующий ему оператор или группу операторов так же, как в случае построения разветвляющихся программ, но с учетом ряда особенностей:
   а) блок начальных значений цикла по переменной **K** заменяем оператором FOR **K**... . Если в этом блоке кроме **K** указаны иные переменные, то начальные значения им присваиваем до оператора FOR **K**...,
   б) формулу вычисления значения переменной цикла **K** (в блоке «изменение аргументов») и блок проверки окончания цикла опускаем,
   в) символы NEXT **K** переносим в программу на соответствующее место в порядке очередности.

Ограничение. Вход в цикл, минуя оператор FOR, в Бейсике запрещен. При необходимости выполнить такой переход в некотором цикле следует отказаться от использования операторов FOR и NEXT в этом цикле (но лучше отказаться от такого перехода).

Примеры программ. Рассматриваемые ниже задачи должны позволить читателю сопоставить результаты применения двух разных подходов к составлению циклических программ. Сначала попытаемся на конкретной задаче пояснить изложенный выше формальный подход.

Задача 12.8

Вычислить значения функции $Y = \sqrt{A^2 + B^2}$ при одновременном (синхронном) изменении величин A и B:

$$A = 2, 4, 6, ..., 2M;$$
$$B = 5, 10, 15, ..., 5M.$$

Решение

Параметры переменных A и B:

$$A := \begin{vmatrix} 2 \\ A+2; \\ 2M \end{vmatrix} \quad B := \begin{vmatrix} 5 \\ B+5. \\ 5M \end{vmatrix}$$

Схема алгоритма задачи приведена на рис. 12.13. Все формулы и условия, составляющие «начинку» схемы, заданы в формулировке задачи в явном виде. Пояснений к схеме рис. 12.13, надеемся, не требуется.

```
10 REM ЦИКЛ
20 PRINT "ВВЕСТИ M"
30 INPUT M
40 PRINT "A", "B", "Y"
50 B=5
60 FOR A=2 TO 2*M STEP 2
70 Y=SQR(A^2+B^2)
80 PRINT A, B, Y
90 B=B+5
100 NEXT A
110 END
```

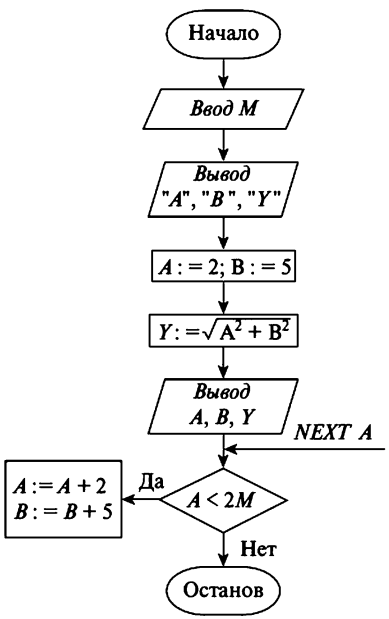


Рис. 12.13

Поясним процесс составления приведенной программы. Здесь операцию A := 2 мы заменили оператором FOR A... (строка 60 программы). Оператор B=5 вынесли за цикл (строка 50). Оператор B=B+5 записали после всех операторов, представляющих рабочий блок (строка 90). Символы NEXT A перенесли в программу после описания всех операций рабочего блока (строка 100). Происхождение остальных операторов программы должно быть понятно.

Отметим еще одну особенность рассматриваемой программы. Значения результатов и исходных данных выводятся здесь в виде таблицы, состоящей из трех колонок по 14 позиций в каждой. Это обеспечивается использова-

нием запятой в качестве разделителя выводимых величин в операторе PRINT A, B, Y (см. 12.1.6). Оператор в строке 40 выводит на экран название каждой из трех колонок таблицы.

Перейдем теперь к программам ввода-вывода массивов, составив их с использованием операторов FOR и NEXT и обобщив на случай использования массивов переменного размера.

Типовые программы ввода и вывода массивов

Задача 12.9, а

Составить программу ввода (вывода) одномерного массива $P(1{:}l)$.

Решение

Будем исходить из схемы алгоритма, изображенной на рис. 12.7.

Программа ввода массива имеет такой вид:

```
20 REM ВВОД МАССИВА
30 INPUT "ВВЕСТИ L"; L
40 DIM P(L)
50 PRINT "ВВЕСТИ МАССИВ P"
60 FOR 1=1 TO L
70 INPUT P(1)
80 NEXT I
90 END
```

Оператор в строке 30 программы обеспечивает ввод фактического значения L.

Программа вывода массива $P(1{:}1)$ будет отличаться от приведенной отсутствием строк 30—50 и наличием в строке 70 оператора «PRINT P(I);».

Задача 12.9, б

Составить программу построчного ввода (вывода) матрицы $A(1{:}n, 1{:}m)$.

Решение

Воспользуемся схемой рис. 12.14, пометки на ней уже сделаны.

Программа ввода матрицы имеет вид:

```
20 REM ВВОД МАТРИЦЫ
30 INPUT "ВВЕСТИ N, M"; N, M
40 DIM A(N, M)
50 PRINT "ВВЕСТИ МАТРИЦУ A"
60 FOR I=1 TO N
```

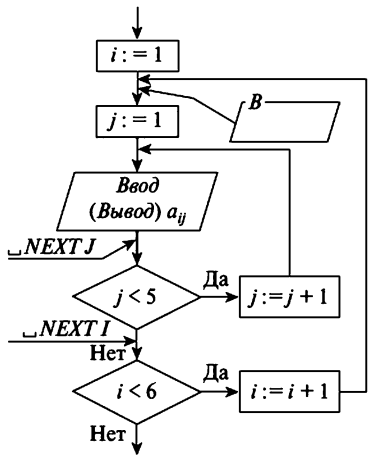


Рис. 12.14

```
70 FOR J=1 TO M
80 INPUT A(I, J)
90 NEXT J
100 NEXT I
110 END
```

Программа вывода матрицы имеет вид:

```
110 REM ВЫВОД МАТРИЦЫ
130 FOR 1=1 TO N
140 PRINT
150 FOR J=1 TO M
160 PRINT A(I, J);
170 NEXT J
180 NEXT I
190 END
```

Пояснение. Программа вывода матрицы, как правило, должна выводить каждую строку матрицы с новой строки экрана. Это обеспечивает блок B (положение его в схеме алгоритма показано на рис. 12.14). Обычно этот блок содержит один или несколько операторов PRINT без аргументов. В нашей программе он представлен 40-й строкой.

Внимание! Именно последний символ строки 60 (разделитель в конце оператора PRINT) определяет размещение элементов матрицы на экране — будут ли они располагаться построчно («;»), в пять колонок («,») или в один столбец (отсутствие разделителя).

Примечание. Приведенные программы ввода-вывода массивов можно использовать как фрагменты в составе различных программ, меняя лишь имена величин и размеры массивов на фактические.

Рассмотрим задачу другого типа.

С использованием операторов FOR и NEXT составить программу решения задачи 12.7 для случая — задан массив $P(1;m)$.

Решение

Обратимся к схеме алгоритма задачи (рис. 12.8). Программа имеет вид:

```
10 REM ПРОИЗВЕДЕНИЕ
20 PRINT "ВВЕСТИ M"
30 INPUT M
40 DIM P(M)
50 B=1
60 FOR I=1 TO M
70 INPUT P(1)
80 IF P(I)>0 THEN B=B*P(I)
90 NEXT I
100 PRINT "B="; B
110 END
```

В заключение хотелось бы предложить читателю проверить свои силы в решении более серьезной задачи. Для этого предлагаем ему составить программу задачи 11.6 (схема алгоритма задачи приведена на рис. 11.18). Эта задача не имеет каких-либо «подводных» камней, типичная задача обработки массивов. Следует лишь обратить внимание на проверку равенства действительных величин — элементов массивов *A* и *B*.

Полученную программу можно сравнить с приведенной ниже. В ней принято допущение о том, что соответствующие элементы массивов *A* и *B* можно считать равными, если их разность (по абсолютной величине) не более 0,01.

Программа задачи 11.6:

```
10 REM ВСТАВКА
20 PRINT "ВВЕСТИ K, N"
30 INPUT K, N
40 DIM A(N), B(K), C(K)
50 PRINT "ВВЕСТИ МАССИВЫ B(K), C(K)"
60 FOR I=1 TO K
70 INPUT B(I), C(I)
80 NEXT I
90 PRINT "ВВЕСТИ МАССИВ A(N)"
100 FOR J=1 TO N
110 INPUT A(J)
120 NEXT J
130 FOR I=1 TO N-K+1
140 FOR J=1 TO K
150 P=I+J-1
160 S=ABS(A(P)-B(J))
170 IF S>0.01 THEN GO TO 220
180 NEXT J
190 FOR J=1 TO K
200 A (I+J-1)=C(J)
210 NEXT J
220 NEXT I
230 PRINT TAB(20);" МАССИВ A (РЕЗУЛЬТАТ):"
240 FOR I=1 TO N
250 PRINT A(I);
260 NEXT I
270 END
```

Пояснение. В приведенных выше задачах, применив операторы FOR и NEXT, мы, фактически, перешли от схемы алгоритма решения задачи вида 10.28, *а* к схеме алгоритма вида 10.28, *б*. Но в этих задачах вид алгоритма роли не играет.

12.3.2. Программы с файлами

Общие понятия

Внимание! Прежде чем приступать к составлению указанного вида программ, необходимо вспомнить понятия «файл», «каталог (подкаталог)», «файловая система» и пр. — см. главу 3.1.2.

Как сказано в этой главе, *файл* — это поименованная совокупность данных на внешнем носителе (гибком или жестком диске).

Приведем примеры файлов. Файлом может быть совокупность всех операторов программы, записанная на диск, — это *программный файл*. Подобные файлы в MS-DOS имеют тип (расширение) exe, com, bas, pas и пр.

Совокупность всех исходных данных некоторой задачи, записанная на диск, — пример *файла данных*. Ниже мы будем четко различать эти два вида файлов — программные файлы и файлы данных.

Для нас особый интерес будут представлять *файлы данных, к которым можно обращаться из программы на Бейсике*.

Достоинства их:

- позволяют хранить исходные данные и результаты работы программы на диске длительное время;
- позволяют связать по данным две или более программ в том случае, когда результаты выполнения одной программы должны являться исходными данными для другой программы (других программ). В этом случае одна программа записывает данные в файл, а другая (другие) «черпает (ют)» из него данные.

Файлы последовательного доступа

Здесь мы рассмотрим использование файлов данных — такую возможность предоставляет QBASIC.

Подобные файлы подразделяют на два вида:

- *файлы последовательного* доступа — к элементам их можно обращаться строго в порядке расположения элементов в файле;
- *файлы прямого доступа* — к их элементам можно обращаться в произвольном порядке.

Мы ограничимся изучением файлов данных последовательного доступа как наиболее простых для понимания. Кроме того, они обладают особенностью — данные, записанные в такие файлы, можно читать и редактировать с помощью текстового редактора, как обычный текст.

Общие понятия и определения

Внимание! В указанной версии Бейсика файл последовательного доступа является по структуре последовательностью величин любого допустимого в языке типа — целого, вещественного, текстового и т.д.; длина последовательности — не ограничена.

Для обращения к файлу данных в программе используется не его имя или полное имя, а номер, закрепляемый за ним в программе. Номер файла — целое число из диапазона 1 — Nm, где Nm в разных версиях Бейсика может быть различным.

Пояснение. Зачем нужен номер файла? Начнем с аналогии. У каждого человека есть полное имя, включающее фамилию, имя, отчество, но в узком кругу ради краткости используют уменьшительные имена — «Ванечка», «Танечка» и пр. Номер файла — это фактически «уменьшительное» имя файла, используемое в «узком кругу» программы для краткости, взамен громоздкого полного имени файла.

В разных программах за одним и тем же файлом можно закреплять различные номера — безразлично какие.

Режим работы файла. Файл данных можно использовать в программе в любой момент времени в одном из двух режимов:

- режим записи данных в файл — OUTPUT;
- режим чтения данных из файла — INPUT.

Операторы и функции для работы с файлами в QBASIC. Работа с файлами потребует знакомства с новыми для читателя операторами, не встречавшимися ранее.

Оператор OPEN — описывает файл, т.е. сообщает характеристики файла: имя файла, его адрес, номер в программе, режим использования. Главное, он открывает файл, т.е. связывает ОЗУ с файлом на диске; для вновь создаваемого файла, кроме того, выделяет на диске область памяти и очищает ее от «старой» информации.

Пример оператора:

```
10 OPEN "D:\DOG.DAT" FOR INPUT AS #2
```

Здесь оператор OPEN закрепляет номер 2 за файлом DOG.DAT диска D: и подготавливает файл для считывания из него данных (режим INPUT).

Общий вид оператора:

$$\text{OPEN"}<\textit{адресф.}>\backslash<\textit{имяф.}>\text{"FOR}\begin{Bmatrix}\text{OUTPUT}\\ \textit{или}\\ \text{INPUT}\end{Bmatrix}\text{AS \#n}$$

где <*имя ф.*> — имя и тип файла; тип файла данных обычно DAT (может быть опущен); <*адрес ф.*> — адрес файла, т.е. имя диска и имена всех подкаталогов (П/К), в которые входит файл; может быть опущен, тогда файл открывается в текущем П/К или каталоге текущего диска; **n** — номер файла, 1<**n**<255.

Оператор записывается обычно в начале программы.

Оператор CLOSE — закрывает файлы с указанными номерами после прекращения работы с ним, т.е. разрывает связь файлов с ОЗУ. Обычно записывается в конце программы.

Общий вид оператора:

$$\text{CLOSE } \#n_1, \#n_2, \ldots$$

Пример:

```
30 CLOSE #5,#6,#3
```

Оператор CLOSE без параметров закрывает все файлы.

Оператор вывода — записывает данные в файл.

Общий вид оператора:

а) PRINT **#n, x_1, x_2, ..., x_i, ...**

б) WRITE **#n, x_1, x_2, ..., x_i, ...**

где **#n** — номер файла; x_i — константа, переменная или элемент массива любого типа — целого, вещественного, текстового и т.д.

Оба оператора последовательно записывают значения величин x_1, x_2, ... в файл с номером n, т.е. данные операторы работают, как оператор PRINT, с той лишь разницей, что выводят данные не на экран, а в файл.

Оператор WRITE при записи разделяет значения запятой. Он полезен при работе с текстом.

Оператор INPUT # — читает данные из файла.

Общий вид оператора:

$$\text{INPUT } \#n, y_1, y_2, \ldots, y_i, \ldots$$

где **n** — номер файла; y_i — имя переменной или элемента массива любого типа.

Оператор последовательно читает данные из файла с номером **n** и также последовательно присваивает их значения переменным y_1, y_2, ..., т.е. данный оператор работает, как оператор INPUT, с той лишь разницей, что считывает данные не с экрана, а из файла.

Внимание! Тип величины y_i должен строго совпадать с типом считываемого (*i*-го) элемента файла, а значит, и с типом соответствующего элемента x_i; оператора PRINT.

Пример:

```
20 OPEN "C:\PPP" FOR OUTPUT AS #1
30 PRINT #1, A, B$, C%, P
```

В этом примере оператор PRINT #1 записывает четыре различных по типу величины в файл PPP.

Оператор, читающий данные из этого файла, может иметь такой вид:

```
100 INPUT #2, D, K$, R%, L
```

Предварительно файл должен быть открыт оператором:

```
80 OPEN "C:\PPP" FOR INPUT AS #2
```

Функция EOF(n) — принимает значение «истинно», если достигнут конец файла с номером n (EOF — аббревиатура от End Of File).

Функция используется для организации цикла чтения данных из файла при неизвестной его длине, что обычно и бывает!

В этом случае условие повторения цикла — «NOT EOF(n)». Пример применения функции — см. программу задачи 10.12.

Задача 12.10

Записать N чисел в файл DA диска E: (N<199).

Решение

Программа может быть такого вида:

```
10 REM ЗАПИСЬ В ФАЙЛ (BAP.1)
20 OPEN "E:\DA" FOR OUTPUT AS #1
30 DIM D(199)
40 INPUT "ВВЕСТИ N"; N
50 FOR 1=1 TO N
60 INPUT D(I)
70 PRINT #1, D(I)
80 NEXT I
90 CLOSE #1
100 END
```

```
10 REM ЗАПИСЬ В ФАЙЛ
   (BAP.2)
20 OPEN "E:\DA" FOR
   OUTPUT AS #1
30 INPUT "ВВЕСТИ N"; N
40 FOR I=1 TO N
50 INPUT D
60 PRINT #1, D
70 NEXT I
80 CLOSE #1
90 END
```

В этих программах оператор OPEN открывает файл DA на диске E: для работы в режиме записи данных (режим OUTPUT) и присваивает файлу номер 1. 1-й вариант программы в отличие от 2-го позволяет одновременно сохранять данные и в ОЗУ, и в файле.

Примеры программ с использованием файлов

Программа 1 — запись данных в файл LEW диска D::

```
10 OPEN "D:\LEW" FOR OUTPUT AS #3
20 DIM A(3)
```

```
30 A(1)=2: A(2)=10: INPUT A(3)
40 PRINT #3, A(1), A(2), A(3)
50 CLOSE #3
60 END
```

Программа 2 — чтение данных из файла LEW:

```
10 OPEN "D:\LEW" FOR INPUT AS #1
20 DIM B(3)
30 INPUT #1, B(1), B(2), B(3)
40 S=B(1)+B(2)+B(3)
50 PRINT "S="; S
40 CLOSE #1
60 END
```

Дадим некоторые пояснения. Программа 1 формирует в ОЗУ массив 1:3) и создает файл LEW на диске D:. Оператор OPEN в строке 10 открывает файл для записи в него данных (режим OUTPUT) и присваивает ему номер 3. Оператор строки 40 программы 1 записывает данные в файл LEW, в то время как оператор строки 30 программы 2 читает данные из этого файла в массив *B*.

Следующая задача иллюстрирует тот факт, что считывание данных из файла и ввод их с клавиатуры выполняется практически одним и тем же оператором.

Задача 12.11

Составить:

а) программу ввода с клавиатуры переменных *A*, *B*, *C* и вычисления их суммы;

б) программу считывания данных из файла LEW (образованного программой 2 предшествующего примера) и вычисления их суммы.

Решение

Программа а:

```
5 REM ВВОД ДАННЫХ
  С КЛАВИАТУРЫ
10 INPUT A, B, C 20
   S=A+B+C
30 PRINT "S=";S
40 END
```

Программа б:

```
10 REM ВВОД ДАННЫХ ИЗ ФАЙЛА
20 OPEN "D:\LEW" FOR INPUT AS #2
30 INPUT #2,A, B, C 40 S=A+B+C
50 PRINT "S=";S
60 CLOSE #2
70 END
```

Рекомендуем читателю сравнить программы а и б и выявить их различия.

Особенности процесса обмена данными с диском

1. Чтение данных из файла не разрушает данные, они сохраняются, с них как бы снимается копия.
2. При записи данных в файл, которого нет на ВЗУ (на диске), создается новый файл.
3. При записи данных в существующий на ВЗУ (на диске) файл этот файл уничтожается и организуется новый с тем же именем.
4. К одному и тому же файлу можно обращаться из разных программ. При этом в каждой программе файл может иметь свой номер.
5. Файл, открытый для записи в него данных, нельзя использовать для чтения из него данных и наоборот. При необходимости обращаться в программе к файлу для чтения и для записи данных следует сначала открыть его для работы в режиме чтения (записи), выполнить необходимые операции, закрыть файл. После этого вновь открыть его для работы в режиме записи (чтения).

Задача 12.12

Из файла DA, образованного программой задачи 12.10, выбрать все элементы с четными номерами и записать их на тот же диск в файл с именем NET. Проверить содержимое файла NET.

Решение. Программа:

```
10 REM ПЕРЕНОС ДАН. ИЗ ФАЙЛА В ФАЙЛ
20 OPEN "E:\DA" FOR INPUT AS #1
30 OPEN "E:\NET" FOR OUTPUT AS #2
40 WHILE NOT EOF(1)
50 INPUT #1, A, B
60 PRINT #2, B
70 PRINT A; B;
80 WEND
90 CLOSE
100 OPEN "E:\NET" FOR INPUT AS #2
110 PRINT "ФАЙЛ NET: "
120 IF EOF(2) THEN GO TO 160
130 INPUT #2, B 140 PRINT B;
150 GOTO 120
160 CLOSE
170 END
```

В полученной программе оператор в строке 30 открывает файл NET для записи в него данных. Оператор в строке 50 читает из файла DA по две величины в каждом цикле, а следующий за ним помещает каждый четный элемент в файл NET. Оператор OPEN строки 100 вновь открывает уже закрытый файл NET для чтения данных.

Первый цикл в программе организован с помощью оператора WHILE... WEND, условие повторения цикла здесь — «файл DA не закончен», что в переводе на Бейсик означает — NOT EOF (1).

Второй подобный же цикл построен с использованием GOTO, здесь условие окончания цикла — «файл NET закончен», оно выполняется, если значение EOF (2) — «истинно».

Рекомендации по использованию файлов последовательного доступа при решении задач. Здесь следует выделить два вида задач.

Первый вид. Обработка данных всегда выполняется строго в одной и той же последовательности, как, например в обучающих и контролирующих программах. В этом случае использование файлов последовательного доступа естественным образом обеспечивает обработку данных.

Второй вид. Обработка данных возможна в произвольном порядке, как, например при выдаче справок в информационно-справочных программах. В этом случае характер процесса обработки данных не соответствует сущности файлов последовательного доступа и использование подобных файлов требует некоторых ухищрений и специальных приемов.

Пару таких приемов можно предложить на случай работы с файлами небольшого и среднего объема:

1) в начале программы организуем блок, обеспечивающий чтение (перенос) всех данных файла (файлов) в ОЗУ, в массивы удобной для нас структуры. После этого программа может обращаться к элементам этих массивов в произвольном порядке.

Если в процессе работы элементы массивов подвергались изменению, то по окончании сеанса работы с ними переносим содержимое массивов со всеми изменениями в те же файлы.

2) для вызова N-го элемента файла читаем все элементы файла с 1-го по N-й, после чего работаем только с *N*-м элементом.

Однако при записи *N*-го элемента в файл такой прием не проходит!

Задача 12.13

Выдать на экран K-й элемент файла DA (см. задачу 12.10).

Решение

Программа может иметь такой вид:

```
10 REM ЧТЕНИЕ ИЗ ФАЙЛА
20 OPEN "E:\DA" FOR INPUT AS #3
30 INPUT "ВВЕСТИ K"; K
40 FOR 1=1 TO K
50 INPUT #3, K
```

```
70 NEXT I
80 PRINT "К-Й ЭЛ. ФАЙЛА="; D
90 CLOSE #3
100 END
```

Контрольные вопросы

1. Каковы назначение и общий вид оператора цикла?
2. Можно ли войти в тело оператора цикла, минуя заголовок цикла?
3. Что такое файл? Каково его назначение? Как обозначается файл? Какие типы файлов данных существуют?
4. В чем отличие файлов прямого доступа от файлов последовательного доступа?
5. Какова структура файлов последовательного доступа?
6. Каковы операторы для работы с файлами?

Задачи для самостоятельного решения

1. Для каждой из задач 1—12 (см. 10.4 «Задачи для самостоятельного решения») составить программу с использованием оператора цикла.
2. В задачах 2, 3, 6 того же раздела считать, что исходные матрицы записаны в файл с именем РЕК. Составить программы решения задач для этого случая.
3. Составить программы решения задач 2, 5, 7 того же раздела, что и в п. 1, обеспечив в них запись результатов в файл с именем КОТ.

12.4. Решение задачи в режиме диалога

12.4.1. Режимы работы программы

Возможны два режима:

а) *пакетный режим работы программы* — под этим термином будем понимать такой режим, когда задача независимо от способа ввода программы в ЭВМ (ПЭВМ) *решается за один шаг* (ввод данных — вывод результатов) и без вмешательства пользователя в процесс решения задачи. Большинство программ, построением которых мы занимались ранее, работает именно в таком режиме;

б) *диалоговый режим работы программы* — в этом случае в процессе решения задачи программа выдает на экран вопрос пользователю, а после получения ответа (вводимого с клавиатуры) про-

должает решение, сообразуясь с этим ответом. Вопросы может вводить пользователь, тогда отвечать должна ЭВМ (программа). Как правило, выполняется *многократный* обмен сообщениями *за ряд шагов*.

Обращаем внимание на то, что в соответствии с приведенными выше понятиями при использовании ЭВМ в режиме диалога одна программа может решать нашу задачу в пакетном режиме, а другая — в диалоговом.

Итак, *диалоговый режим работы программы* заключается в поочередном и многократном обмене сообщениями пользователя и программы (ЭВМ) в процессе решения задачи. Именно это понятие будет вкладываться в дальнейшем в термины «диалоговый режим», «диалог».

Программы (алгоритмы), работающие в режиме диалога, будем называть *диалоговыми*. Они должны обладать определенной структурой, как правило, достаточно своеобразной. Именно вопросам построения диалоговых алгоритмов и программ посвящена настоящая глава.

12.4.2. Сценарий диалога

Определения и общие понятия. Необходимым условием для разработки диалоговых программ является наличие сценария диалога.

Сценарием диалога называется перечень всех вопросов и ответов, которыми обмениваются пользователь и ЭВМ (программа) в ходе решения задачи. Чтобы было понятнее, о чем идет речь, вспомним, что текст пьесы — это перечень сообщений, которыми обмениваются актеры на сцене при постановке этой пьесы. По аналогии сценарий диалога — это текст пьесы для двух исполнителей — пользователя и ЭВМ.

Диалог пользователя и ЭВМ (а соответственно и сценарий диалога) распадается на *шаги*.

Шагом диалога (*транзакцией*) называется однократный обмен сообщениями, т.е. на одном шаге пользователь выдает одно сообщение и получает на него ответ ЭВМ (или наоборот). В соответствии с этим составление сценария диалога заключается в правильном выборе последовательности шагов и типа диалога на каждом шаге. Сценарий диалога обычно разрабатывается на этапе содержательной постановки задачи.

Обозначения в сценарии диалога: «*A*» — текст, выводимый на экран; <*A*> — значение величины *A*, выводимое на экран; *A*/*B*/*C* — варианты значения некоторой величины.

Пример: запись «фамилия» означает, что нужно вывести на экран слово, заключенное в кавычки, в то время как запись — <фамилия>

означает, что нужно вывести на экран конкретную фамилию некоторого человека, т.е. значение переменной «фамилия».

Запись — ОТВЕТ = <A>/<B>/«конец» означает, что ответ пользователя должен заключаться в выводе на экран значения величин *A* либо *B* либо текста — «конец».

Приведем для примера сценарий диалога вычисления параметров круга и квадрата:

П. 1. ЭВМ: «Параметры какой фигуры вас интересуют?»:
«1. Круг.
2. Квадрат.
3. Вычисления прекратить».

П. 2. Пользователь: ОТВЕТ = 1/2/3 (т.е. 1, 2 или 3).

П. 3. ЭВМ:
а) если ОТВЕТ = 1 — «Введите радиус *R*». Перейти к п. 4;
б) если ОТВЕТ = 2 — «Введите длину стороны *A*». Перейти к п. 6;
в) если ОТВЕТ = 3 — перейти к п. 8.

П. 4. Пользователь: ОТВЕТ = <R> (т.е. значение *R*).

П. 5. ЭВМ: .....» Площадь круга *SK*=«, <*SK*>;
.....» Длина окружности *L*=«, <*L*>.
Перейти к п. 1.

П. 6. Пользователь: ОТВЕТ = <A>.

П. 7. ЭВМ: .....» Площадь квадрата *S*=«, <*S*>;
.....«Периметр P=», <P>. Перейти к п. 1.

П. 8. ЭВМ: «До свидания». Останов.

Пример иллюстрирует тот факт, что сценарий диалога представляет собой словесное описание процесса диалога, порядок (метод), а иногда и алгоритм выполнения диалога.

Подчеркнем также, что приведенный сценарий — пример простейшего сценария, в котором умышленно для ясности и простоты понимания опущен ряд моментов, совершенно необходимых для реального сценария, о чем будет идти речь далее.

12.4.3. Обучающие и контролирующие программы как пример диалоговых программ

Обучающая программа. Назначение ее вытекает из названия. В такой программе рабочая часть может практически отсутствовать — для нее диалог не только естественная, но и основная часть процесса решения задачи. Поэтому сценарий диалога подобной программы часто и есть словесное описание алгоритма решения всей задачи.

Составление такой программы начинают с разбиения всей изучаемой информации на *кадры*, т.е. такие порции информации, каждую из которых обучаемый должен усвоить за один прием.

Далее для каждого кадра составляется *вопрос* для проверки усвоения учащимся информации этого кадра и формулируется *эталон* — правильный ответ на этот вопрос (или правило его вычисления).

Вся информация по каждому кадру вводится в ЭВМ.

В соответствии со сказанным обучение с помощью ЭВМ производится за ряд шагов. На каждом шаге ЭВМ (программа):

- выдает один кадр информации;
- выдает вопрос, на который обучаемый должен ответить;
- принимает ответ обучаемого и сравнивает его с эталоном.

Если ответ верен, то программа выдает следующий кадр информации; если неверен, то сообщает об этом, а иногда и дает некоторые разъяснения, после чего выдает тот же кадр для продолжения его изучения.

Контролирующая программа. Это частный случай обучающей программы. Контролирующие программы предназначены для контроля знаний и в соответствии с этим должны выдавать учащимся только вопросы, проверять правильность ответов и оценивать степень знаний учащихся. Принципиальных отличий от обучающих программ контролирующие не имеют. Особенность их — они определяют число правильных ответов и выдают («выставляют») оценку знаний учащегося.

Для примера рассмотрим построение простейших программ указанного вида.

Задача 12.14

Построить контролирующую программу по теме «Геометрическая фигура — прямоугольник».

Предварительно разобьем информацию, подлежащую контролю, на такие кадры:

1) определение понятия «прямоугольник»;
2) вычисление площади прямоугольника;
3) вычисление периметра прямоугольника.

Текст каждого кадра, вопрос и эталон указаны непосредственно в сценарии.

Сценарий диалога контролирующей программы

П. 1. ЭВМ: «Проверим, что вы знаете о прямоугольнике? Вы готовы отвечать? (д/н)».

П. 2. Пользователь: ОТВЕТ=«д»/«н».

П. 3. ЭВМ:

а) если ОТВЕТ=«н» — «Готовьтесь!» Перейти к п. 16;

б) если ОТВЕТ ≠ «д»/«н» — «Введите правильный ответ». Перейти к п. 1; иначе — перейти к п. 4.

П. 4. ЭВМ: «Введите фамилию».

П. 5. Пользователь: ОТВЕТ = <Фамилия>.

Кадр 1.

П. 6. ЭВМ: «<Фамилия>. Ответьте — которое из трех определений прямоугольника верное: [1]»

«1. ...»;

«2. ...»;

«3. ...».

П. 7. Пользователь: ОТВЕТ=1/2/3.

П. 8. ЭВМ:

а) если ОТВЕТ=1/3 — «Учите определение прямоугольника», «<Фамилия>». Перейти к п. 9;

б) если ОТВЕТ=2 — «Правильно!». «<Фамилия! >». Число правильных ответов М увеличить на 1 (М=М+1). Перейти к п. 9;

в) если ОТВЕТ ≠ 1/2/3 — «Ответ не понятен!».

Перейти к п. 6.

Кадр 2.

П. 9. ЭВМ: «Ответьте на вопрос: чему равна площадь прямоугольника *ABCD*: *AB* = *CD* = 50; *BC* = *DA* = 60?».

П. 10. Пользователь: ОТВЕТ = 300/<*N*>, где N ≠ 300.

П. 11. ЭВМ:

а) если ОТВЕТ ≠ 300 — «Учите формулу вычисления площади!». Перейти к п. 12;

б) если ОТВЕТ = 300 — «Правильно!»; М=М+1.

Кадр 3.

П. 12. ЭВМ: «Ответьте на следующий вопрос: чему равен периметр прямоугольника *ABCD*: *AB*=*CD*=30; *BC*=*CD*=100?».

П. 13. Пользователь: ОТВЕТ = 260/<*N*>, где *N* ≠ 260.

П. 14. ЭВМ:

а) если ОТВЕТ ≠ 260 — «Учите формулу вычисления периметра». Перейти к п. 15;

б) если ОТВЕТ = 260 — «Правильно!»; М=М+1.

П. 15. ЭВМ: «<Фамилия>», «число Ваших верных ответов =», <М>; если М<3 — «Плохо! Учите!». Перейти к п. 1.

[1] Подразумевается, что далее на экран выводятся три различных определения прямоугольника, из которых верно лишь второе.

если М = 3 — «Молодец! Отлично!». Перейти к п. 1.

П. 16. ЭВМ: «До свидания!». Останов.

Примечание. 1. При выводе кадра 1 сценария мы предусмотрели обращение к пользователю по фамилии. Такой прием позволяет сделать диалог более «теплыми, более дружественным. В кадрах 2 и 3 мы не использовали подобное обращение, чтобы не загромождать сценарий.

Итак, мы получили сценарий диалога задачи. Далее, используя его непосредственно, можно составить требуемую программу, опираясь лишь на Основные принципы алгоритмизации (см. 10.1).

Для построения программы в этом случае достаточно каждый пункт сценария заменить соответствующим оператором языка Бейсик — ввода, вывода, условным и т.д. Это самый простой подход, в результате которого, правда, создаются не самые эффективные программы, прямо скажем, громоздкие. Тем не менее рекомендуем читателю написать программу, используя этот подход, с тем, чтобы он мог «прочувствовать» все особенности составления диалоговых программ. Подход, обеспечивающий построение более эффективных программ подобного вида, рассмотрен в 12.4.4.

12.4.4. Использование массивов, циклов и файлов в обучающих (контролирующих) программах

В 12.4.3. мы рассмотрели постороение программ указанного вида. Несложно представить, что при использовании рекомендованного в этом параграфе подхода более или менее реальные программы подобного вида становятся, мягко говоря, громоздкими, и даже очень.

Для уменьшения сложности таких программ следует применять обычные средства:

- массивы и файлы для хранения текстов;
- подпрограммы (п/п) и циклы для реализации повторяющихся операций.

Покажем применение таких средств на примере решения следующей простой задачи.

Задача 12.15

Составить контролирующую программу по теме «Операторы языка Бейсик», реализующую три следующих шага диалога:

Шаг 1.	Вопрос:	«Назовите имя оператора присваивания»	(A(1))
	Эталон:	«LET»	(A(2))
Шаг 2.	Вопрос:	«Назовите имя оператора перехода»	(A(3))
	Эталон:	GOTO	(A(4))

Шаг 3.	Вопрос:	«INPUT — оператор вывода? (Д/Н)»	(A(5))
	Эталон:	«Н»	(A(6))

Примечание. В правой колонке указаны имена элементов массива A, в которых будут размещены приведенные слева от них тексты. Массив A рассмотрен ниже.

Решение

Пояснение. Действуем по обычной схеме: исходные данные — результаты — метод и т.д. Здесь данными являются шесть различных текстов — три вопроса и три эталона. Закономерности в изменении этих величин нет, поэтому представим их в виде массива.

Исходные данные: массив A(1:6) — значения его элементов приведены выше. Результат: M — число правильных ответов; C — оценка.

Пояснение. Исходные данные в нашем случае делятся на постоянные, которые не должны меняться в каждом сеансе работы с программой — массив A, и переменные данные — ответы пользователя, их мы рассмотрим позже.

Поэтому массив A следует не вводить с клавиатуры, а формировать в программе, лишая пользователя (учащегося) доступа к нему.

Метод решения задачи в целом требует решения четыре подзадач:

- представление программы, т.е. сообщение пользователю данных о программе, выяснение его готовности к работе с программой, ввод фамилии пользователя и т.д. Этот этап решения задачи называют «прологом»;
- формирование массива A;
- непосредственный контроль знаний;
- вычисление и вывод оценки.

Соответственно и укрупненная схема алгоритма задачи должна содержать четыре блока.

Схемы блоков 1 и 4 достаточно просты и однозначно определяются сценарием диалога. Будем считать, что они оформлены в виде п/п и номера их начальных строк 100 и 500 соответственно.

Программа блока 2 — линейная, должна включать последовательность из $2k$ операторов присваивания, где k — число вопросов, выдаваемых учащемуся программой.

В нашем случае программа блока 2, оформленная в виде п/п, будет иметь вид:

```
200 REM SUB БЛОК 2
210 A(1)="НАЗОВИТЕ ИМЯ ОПЕРАТОРА ПРИСВАИВАНИЯ"
220 A(2)="LET"
230 A(3)="НАЗОВИТЕ ИМЯ ОПЕРАТОРА ПЕРЕХОДА"
```

```
240 A(4)="GOTO"
250 A(5)="INPUT – ОПЕРАТОР ВЫВОДА? (Д/Н)"
260 A(6)="Н"
270 RETURN
```

Основной блок — 3-й. Именно он определяет сложность всей программы. Именно в этом блоке существенно использование подпрограмм, циклов и т.д. Рассмотрим различные варианты программы этого блока.

Использование цикла. Метод решения задачи блока 3: решить трижды с различными значениями исходных величин одну и ту же подзадачу (назовем ее задача С):

- вывести на экран текст Т1 (вопрос);
- ввести с клавиатуры текст Т2 (ответ);
- проверить совпадение текста Т2 с Е (эталоном): при совпадении вывести на экран текст «Верно!», увеличить на единицу значение М;

иначе — вывести на экран текст «Неверно!».

Теперь составим циклическую программу решения задачи блока 3. Для этого, как обычно, опишем процесс решения задачи по этапам, учитывая, что на каждом этапе решается одна и та же задача С, меняются лишь значения исходных величин.

Этап 1. Решаем задачу С при Т1=A(1) и Е=A(2);

Этап 2. Решаем задачу С при Т1=A(3) и Е=A(4);

Этап 3. Решаем задачу С при Т1=A(5) и Е=A(6);

Этап г. Решаем задачу С при T*i*=A(2*i* – 1) и Е=A(2*i*).

Минуя процесс составления алгоритма, сразу можем написать программу решения задачи — это программа с одним циклом. В рабочий блок войдут все операции задачи С и операции по вычислению Т1 и Е на этапе *i*.

Программа:

```
300 REM SUB БЛОК 3 (КОНТР. ЗНАНИЙ) ВАР.1
310 REM РАБОЧАЯ ПРОГРАММА; "ЦИКЛ"
320 M=0
330 FOR I=1 TO 3
340 T1$=A$ (2*I-1): E$=A$(2*I)
350 PRINT T1$
360 INPUT T2$
370 IF T2$=E$ THEN PRINT "ВЕРНО!": M=M+1: GOTO 390
380 PRINT "НЕВЕРНО!"
390 PRINT: PRINT
400 NEXT I
```

```
410 PRINT "ЧИСЛО ВЕРНЫХ ОТВЕТОВ=";M
420 RETURN
```

Анализируя полученную программу, нетрудно увидеть ее достоинства — *объем программы не зависит от числа шагов диалога*.

Теперь можно составить главную программу, объединяющую четыре наших п/п в единую программу.

```
10 REM КОНТР. ЗНАНИЙ (ГЛАВНАЯ ПРОГРАММА)
20 DIM A$(6)
30 GOSUB 100
40 GOSUB 200
50 GOSUB 300
60 GOSUB 500
70 STOP
100 REM SUB БЛОК 1 (ПРОЛОГ)
...
190 RETURN
200 REM SUB БЛОК 2 (ФОРМ. МАС. А)
...
290 RETURN
300 REM SUB БЛОК 3 (КОНТР. ЗНАНИЙ)
...
490 RETURN
500 REM SUB БЛОК 4 (ВЫВОД ОЦЕНКИ)
...
590 RETURN
700 END
```

В эту программу можно подставлять любой вариант подпрограммы блока 3 (также как и в любой другой блок).

Внимание! Приведенная программа демонстрирует применение модульного принципа программирования в том случае, когда порядок выполнения модулей (подпрограмм) в главной программе фиксирован.

Использование файлов данных. Допустим теперь, что вопросы и эталоны записаны в файл (последовательного доступа).

Пусть это будет файл DAN. DAT на диске D:

Наличие файла приведет лишь к изменению программы блока 2. В этом случае блок 2 должен всего лишь «выгрузить» вопросы и эталоны из файла в массив А (в ОЗУ). Программа этого блока (применительно к версии QBASIC) может быть такой:

```
200 REM SUB БЛОК 2 (С ИСПОЛЬЗОВАНИЕМ ФАЙЛОВ)
210 OPEN "D:\DAN.DAT" FOR INPUT AS #1
220 FOR 1=1 TO 6
```

```
230 INPUT #1, A$(I)
240 NEXT I
250 CLOSE #1
260 RETURN
```

При таком подходе потребуется программа создания файла DAN.DAT. Приведем ее без пояснений.

```
REM ЗАПИСЬ ВОПРОСОВ И ЭТАПОВ В ФАЙЛ
OPEN "D:\DAN.DAT" FOR OUTPUT AS #1
FOR 1=1 TO 3
PRINT "Введите"; I; "-Й ВОПРОС"
INPUT W$
PRINT "Введите";I; "-Й ЭТАЛОН"
INPUT E$
WRITE #1, W$, E$
NEXT I
CLOSE #1
END
```

Данную программу и контролирующую (главную) программу можно объединить в единый комплекс с помощью еще более главной программы. В нее можно включить и программу корректировки файла DAN.DAT — файла вопросов и эталонов.

Приведенные программы ориентированы на QBASIC.

Вывод. Использование файлов и циклов дает возможность создавать универсальные обучающие и контролирующие программы, которые без каких-либо изменений способны работать с любой тематикой, обеспечивая выполнение любого числа шагов диалога (выдачу произвольного числа вопросов), при условии, конечно, однотипности структуры диалога на каждом шаге.

Если же ограничиться использованием массивов и циклов, будут изменяться содержание и размеры только блока 2, остальная часть программы останется неизменной.

ГЛАВА 13

ПОСТАНОВКА ЗАДАЧИ

Итак, мы познакомились с составлением алгоритмов и программ. Однако для того чтобы уметь решать задачи на ЭВМ, этого далеко не достаточно, так как этот процесс включает значительное число и иных разнообразных операций. Кратко мы упомянули о них во введении, где на конкретном примере показали основные этапы решения задачи на ЭВМ.

Теперь можно привести полный перечень этих этапов. Он весьма обширен и внушителен, хотя не все из них обязательно встречаются в каждой задаче, многое зависит от ее специфики.

Этапы решения задачи на ЭВМ.

1. *Содержательная постановка задачи.*
2. *Математическая постановка задачи.*
3. *Формализация задачи (выбор метода решения).*
4. *Составление алгоритма решения задачи.*
5. *Составление программы на языке программирования.*
6. *Отладка программы на ЭВМ — выявление и исправление в ней ошибок.*
7. *Рабочий счет (решение задачи с рабочими данными на ЭВМ) и анализ результатов.*
8. *Составление инструкции к программе.*

Как видно из перечня, составлению алгоритма предшествует большая подготовительная работа — постановка задачи (этапы 1—3). Цель данного процесса — сформулировать задачу предельно четко, понятно; с учетом особенностей используемой ЭВМ и языка программирования; наметить сначала общий подход к ее решению, а затем и строгий математический метод.

Это не так просто, но ни одну серьезную задачу нельзя решить без ее правильной постановки. В этой главе мы и рассмотрим первые три этапа решения задачи на ЭВМ, составляющие этот процесс.

13.1. Содержательная постановка задачи

Содержательная постановка задачи — это формулировка задачи, излагаемая в терминах некоторой конкретной области науки, техники, сельского хозяйства, медицины и т.д.

В ней содержатся все необходимые для решения задачи сведения. Следует сказать, что этап содержательной постановки задачи необходим при решении задачи любым образом, а не только в случае использования ЭВМ.

Суть этого этапа в том, что здесь четко и подробно формулируется: *что дано? что найти? как найти?* Без этого решение никакой задачи немыслимо!

Грамотная постановка задачи существенно облегчает решение задачи, позволяет выявить противоречивость некоторых требований, оценить реальность (или нереальность) некоторых из них, а иногда даже позволяет выявить нецелесообразность решения задачи в целом.

Содержательная постановка включает в общем случае весьма разнообразную информацию о задаче. Мы рассмотрим лишь наиболее существенные пункты этого этапа.

А. Словесное (или словесно-формульное) описание содержания (условия) задачи.

Примечание. Такому описанию должно предшествовать обобщение задачи для максимального расширения области применения будущей программы. Это достигается, например, за счет замены переменными постоянных величин, используемых в задаче (если они при каких-то условиях могут измениться), за счет замены в условии задачи массивов постоянного размера массивами переменного размера (если в этом, конечно, есть смысл) и т.д.

Например, требуется составить программу вычисления величины Z по формуле

$$Z = (3X + bY)/(W + Y).$$

Если заданную формулу заменить такой:

$$Z = (kX + pY)/(W + Y),$$

то получим возможность вычислять значение Z не только при $k = 3$ и $p = 5$, но и при любых других значениях этих величин. При этом усложнение программы незначительно.

Б. Перечень исходных величин и результатов решения задачи (в содержательных терминах) с указанием единиц измерения и точности вычисления.

Примечание. В некоторых задачах указанные величины можно выявить лишь на следующем этапе, после математической постановки задачи.

В. Выбор метода решения задачи и описание его на содержательном уровне в самом общем виде.

Мы исходим из того, что метод решения задачи либо известен из практики, либо описан в литературе, либо подсказывается здравым смыслом.

Для многих задач, как правило, существует несколько способов (методов) их решения, поэтому требуется выбрать один, наиболее целесообразный.

Критерии целесообразности метода могут быть самыми различными. Выбор определяется, в частности, видом задачи, назначением будущей программы и т.д. *Мы рекомендуем выбирать такой метод решения задачи, который позволяет создать предельно простую по структуре и понятную для человека-программиста программу.*

На начальной стадии обучения программированию такой критерий наиболее оправдан. Кроме того, он очень полезен и при решении прикладных задач (в частности, инженерных), так как программы для таких задач в процессе эксплуатации постоянно изменяются, дополняются, корректируются и т.д. И выполнять все это должны, как правило, программисты с ними не знакомые. Успех дела здесь зависит от «прозрачности» программ, их ясности.

В то же время встречаются задачи, решение которых должно удовлетворять иным критериям, например таким:

- время решения задачи должно быть минимальным;
- программа решения задачи должна занимать минимальный объем памяти и т.д.

Примечание. При решении реальных задач на указанном этапе рассматривается и много иной информации. Например, указывается назначение программы и область ее применения, организация входных и выходных данных (форма представления их), требования к надежности, необходимый состав технических средств и их характеристики, требования к методу решения задачи и языку и т.д. Все эти сведения оформляются в виде документа, называемого «Техническое задание».

Примеры содержательной постановки задачи

Задача 13.1

Определить размеры металлического бидона цилиндрической формы объемом 10 л (дм^3) с минимальной площадью поверхности — это условие задачи.

Выделяем исходные данные и результаты, одновременно выполняем обобщение задачи:

исходные данные: объем бидона V (дм^3).

Результаты: параметры бидона с минимальной площадью поверхности:

- площадь поверхности s (дм^2);
- радиус r (дм);
- высота h (дм).

Метод решения задачи. Применим метод перебора вариантов[1]: задаемся некоторым минимально допустимым значением радиуса R бидона, вычисляем высоту H и площадь S поверхности, при которых обеспечивается объем V бидона. Затем несколько увеличиваем значение R и повторяем вычисления вновь. И так n раз. Из всех значений площади поверхности выбираем минимальное — s.

Шаг изменения R определяется по формуле

$$\Delta R = \frac{R_{max} - R_{min}}{n}.$$

Выбранный метод решения требует дополнительных исходных данных: $R_{min}(R_{max})$ — минимально (максимально) допустимый радиус бидона; n — число значений радиуса, при которых выполняются вычисления (определяет точность результата). Значениями этих величин задаемся сами исходя из здравого смысла.

Задача 13.2

Известны данные о выпуске продукции (ткани) ткацким цехом текстильного комбината за каждый день прошлого месяца:

30

32, 36, 38, 37, 38, 36, 35, ...

Определить дни месяца, когда выпуск ткани уменьшался по сравнению с предыдущим днем. Это условие задачи.

Выделяем исходные данные и результаты, одновременно обобщая задачу. Исходные данные: количество дней месяца— n; количество изготовленной ткани по дням месяца (в 1000 м):

$$t_1, t_2, t_3, ..., t_n.$$

Результат: числа месяца, отвечающие требованию условия задачи.

Метод решения задачи. Построить график изменения выпуска ткани в течение месяца и определить убывающие участки этого графика.

[1] Возможен и другой, точный метод решения задачи, но он требует применения более сложного математического аппарата.

13.2. Математическая постановка задачи

Математическая постановка задачи (МПЗ) — это формулировка нашей задачи как задачи некоторого раздела математики.

Фактически на этом этапе мы рассматриваем те же вопросы (*что дано? что найти? как найти?*), что и в содержательной постановке задачи, однако сформулированы они на языке математики. И поскольку при этом используются только математические термины, имеющие каждый точное определение, то ответы на них получаем строго однозначные.

Переход к математической задаче дает возможность подобрать и применить для ее решения методы, разработанные и предоставляемые в наше распоряжение математикой.

Основные понятия. Суть рассматриваемого этапа — в следующем:

- выявляем реальный объект (так будем называть объект, группу объектов, ситуацию, явление и т.д., о которых идет речь в задаче) и из множества параметров (свойств), определяющих реальный объект, выделяем те, которые являются существенными для решаемой задачи;
- далее подбираем математический объект (группу объектов) с тем же числом подобных параметров, отражающий (отражающих) суть реального объекта. Такой математический объект (группу объектов) будем называть математической моделью реального объекта задачи или для краткости «моделью объекта задачи» — это ключевое понятие настоящей главы.

Пример: в задаче 13.1 моделью объекта можно считать цилиндр объемом V, а в задаче 13.2 — график функции.

Построив модель объекта, мы можем отказаться от рассмотрения, анализа реального объекта задачи и перейти к анализу модели объекта, выявлению исходных данных и результатов как параметров модели. Выявив математические соотношения, связывающие исходные данные и результаты, тем самым приходим к той или иной математической задаче, описываемой этими соотношениями.

Рассмотрим конкретный пример. Во введении была рассмотрена задача о кошках. Каждая кошка в жизни характеризуется множеством свойств (параметров): масть, вес, длина ушей и т.д., но нас интересует только один ее параметр — длина хвоста (действительная величина). Поэтому для упрощения ситуации можно заменить реальный объект — «кошка» математическим объектом, определяемым также только одним параметром, например точкой C на числовой оси X с координатой c (*тоже действительная величина*).

Другой объект задачи — диапазон длин хвостов, определяется двумя параметрами — a и b — минимальной и максимальной длиной. Его можно представить таким математическим объектом, как отрезок числовой оси X, определяемым также *двумя параметрами* (границами), т.е. модель объекта нашей задачи будет представлена отрезком $[a, b]$ оси X и точкой C, и мы приходим к решению математической задачи — «Определить, лежит ли точка C на отрезке $[a, b]$?»

Таким образом, на языке математики мы четко сформулировали, что дано и что найти.

Процесс, порядок решения задачи (т.е. ответ на вопрос — как найти?) определяется соотношением:

$$Y = \begin{cases} \text{«Точка } C \text{ лежит на отрезке } [a,b]\text{»}, \text{если } a \le c < b; \\ \text{«Точка } C \text{ — вне отрезка } [a,b]\text{»} — \text{в остальных случаях.} \end{cases}$$

Здесь Y — текстовая переменная, которой присваивается тот или иной текст, отражающий результат решения задачи.

Подобная формулировка и является математической постановкой задачи.

Свойства МПЗ. В общем случае для одной и той же задачи может существовать несколько вариантов МПЗ, так что возникает проблема выбора.

Задача 13.3

Определить периметр крышки стола.

Здесь нас интересуют лишь размеры крышки стола, поэтому заменим реальный объект — «стол» плоской геометрической фигурой, размеры и конфигурация которой соответствуют поверхности крышки.

Это может быть такая фигура (рис. 13.1):

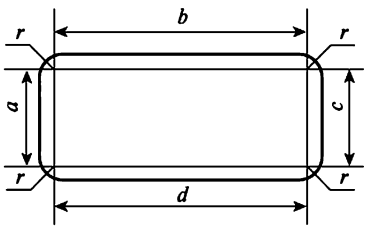


Рис. 13.1

Однако предварительно следует доказать, что радиусы закругления всех углов крышки стола с точки зрения требуемой точности вычисления результатов можно считать равными и только после этого фигура на рис. 13.1 может считаться моделью крышки стола.

Если это доказано, то математическая постановка задачи имеет такой вид:

исходные данные: r, a, b, c, d;

результат: P (периметр);

Вычислить значение P:

$$P = 2nr + a + b + c + d.$$

Если величина r такова, что при требуемой точности вычисления результатов закруглением углов можно пренебречь, то в качестве модели объекта можно взять четырехугольник общего вида (рис. 13.2).

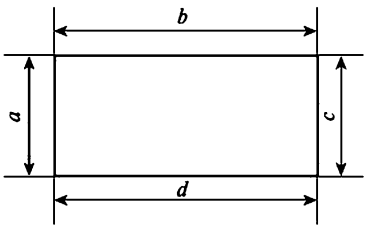


Рис. 13.2

В этом случае приходим к такой задаче:

исходные данные: a, b, c, d; результат: P.

Вычислить значение P:

$$P = a + b + c + d.$$

В качестве модели объекта задачи можно взять прямоугольник со сторонами a и b. Если окажется, что при заданной точности вычисления противоположные стороны крышки можно считать равными, как и диагонали ее, МПЗ имеет такой вид:

исходные данные: a, b;

результат: P.

Вычислить значение P:

$$P = 2\,(a + b).$$

Из рассмотрения даже такой простейшей задачи можно вывести важные *свойства математической постановки задачи*:

а) модель объекта не тождественна реальному объекту, а передает только часть его свойств и качеств, являясь приближенным описанием реального объекта;

б) модель объекта задачи не определяется однозначно реальным объектом; для одной и той же задачи можно принять разные мо-

дели в зависимости от требуемой точности вычисления результатов;

в) для любой выбранной модели объекта необходимо доказать соответствие ее реальному объекту (ситуации), т.е. доказать *адекватность* этих понятий с точки зрения требуемой точности вычисления;

г) в случае приближенной модели объекта результаты решения задачи также являются приближенными; чем точнее необходимо получить результаты, тем сложнее должна быть модель.

Научная постановка задачи. Во многих случаях математической постановке задачи предшествует формулировка ее как задачи некоторой науки, будем называть ее *научной постановкой*. Например, многие задачи, связанные с перемещением реальных предметов и живых существ, сводятся вначале к какой-либо физической задаче о равномерном или равноускоренном движении тела, т.е. вместо того чтобы рассматривать реальный процесс, определяемый множеством свойств, параметров во всей сложности его, мы изучаем физический процесс, который выражает суть реального и отражает только те его свойства, которые существенны с точки зрения нашей задачи. Этот физический процесс является моделью, *физической* (*качественной*) моделью реального процесса, так как отражает в первую очередь качество его — равномерность, равноускоренность и т.д.

Далее выделяются параметры физической модели, которые являются исходными данными и результатами задачи. Все они вместе с моделью и составят физическую постановку задачи.

Для примера возьмем такую ситуацию — кошка выпала из окна третьего этажа дома. Сколько времени она будет находиться в воздухе?

Это *содержательная* постановка задачи.

Физическая постановка может иметь такой вид: тело массой/m = = 1,5 кг движется в свободном падении с начальной скоростью $v_0 = 0$. Определить, за какое время тело преодолеет расстояние $S = 12$ м.

Исходные данные и результаты здесь очевидны.

От физической постановки задачи следует перейти к математической, которая отражает *количественные* соотношения между исходными величинами и результатами, для чего иногда достаточно подобрать известные из курса физики формулы, описывающие данную физическую задачу.

Во многих случаях такой переход однозначен.

Нужно заметить, что физическая и математическая постановка задач — необходимые этапы любого способа решения задачи, а не толь-

ко «машинного». Учащиеся сталкивались с ними и ранее, в курсе физики средней школы.

Пример.

Содержательная постановка задачи: определить, за какое время турист пройдет s км, если движется со скоростью v км/ч?

Физическая постановка задачи: определить время перемещения тела на расстояние s при равномерном прямолинейном движении его со скоростью v.

Математическая постановка задачи определяется здесь однозначно выбранной физической моделью процесса и соответствующей ей формулой $t = s/v$.

Научная постановка задачи — неотъемлемая часть процесса решения технических задач, так как многие из них сводятся к решению задач той или иной технической науки, инженерной дисциплины.

Так, задачи проектирования конструкций, механизмов, машин приводятся, как правило, первоначально к задачам таких наук, как теоретическая и строительная механика, сопротивление материалов, теория механизмов и машин и т.д., причем приводятся обычно к типовым задачам этих наук, для которых уже разработаны математические постановки.

Содержание этапа МПЗ. Обобщая сказанное, перечислим основные операции, связанные с математической постановкой задачи.

1. Формулировка научной постановки задачи, т.е. формулировка ее как задачи некоторой науки (в частности, физики). (Условно отнесем эту процедуру к этапу МПЗ, хотя она и предшествует этому этапу.)
2. Выбор модели объекта задачи, т.е. выбор совокупности математических объектов, отражающих все необходимые для нашего случая свойства реальных объектов, процессов, ситуации задачи. Модель может быть числовой, геометрической, графической, аналитической и т.д.
3. Установление адекватности (соответствия) выбранной модели реальному объекту с точки зрения требуемой точности вычисления результатов или других критериев.
4. Выделение исходных данных и результатов задачи как параметров модели.
5. Выявление математических соотношений, связывающих исходные данные и результаты решения задачи и определяемых свойствами модели. Эти соотношения могут быть выражены формулами, уравнениями, неравенствами, системами уравнений, графиками и т.д. Они могут быть выражены и в словесном виде.

Таким образом, в общем случае сформулировать задачу математически довольно сложно. В то же время в простых случаях для этого достаточно перечислить формулы, вычисление по которым приводит к решению задачи.

Следует отметить и основные особенности рассматриваемой постановки задачи:

а) в математической постановке задачи не должны присутствовать какие-либо содержательные термины;
б) сложные задачи на содержательном уровне разбиваются на подзадачи, и для каждой отдельно выполняется МПЗ;
в) математическая задача, к которой сводится наша задача, должна быть известной, типовой задачей математики. Этого добиться непросто и не всегда возможно, но к этому нужно стремиться.

Примеры типовых задач — вычисление интеграла, дифференцирование функции, решение уравнений, систем линейных уравнений и т.д. Для каждого раздела математики существуют свои типовые задачи.

Пояснение. Последнее требование к МПЗ обусловлено тем, что для большинства типовых математических задач уже составлены программы. На каждой ЭВМ такие программы, сведенные в библиотеки, хранятся в памяти. Использование их не вызывает затруднений. Поэтому если удалось свести задачу к типовой задаче математики, то отпадает необходимость в составлении программы и выполнении всех связанных с этим операций, чем достигается большой экономический эффект от использования ЭВМ. Нередко добиться этого можно небольшим изменением условия задачи (содержательной постановки).

Такой случай демонстрирует следующая задача.

Задача 13.4

Содержательная постановка:

имеется табель посещаемости занятий по химии 10-го класса за первую четверть. В классе не более 30 учеников, число занятий — не более 28. Пропуск занятия обозначается буквой «Н», определить число занятий, пропущенных каждым учеником.

Математическая постановка задачи (вариант 1).

Исходные данные: n — число учеников ($n \le 30$); k — число занятий ($k \le 28$); текстовая матрица $T(1{:}n, 1{:}k)$ — табель посещаемости, где t_{ij} =«Н» или «» (пробел) ($i = 1, 2, ..., n; j = 1, 2, ..., k$).

Результат: массив $D(1{:}n)$.

Формулировка задачи: определить в каждой строке матрицы T число элементов, равных «*Н*», и образовать массив $D(1{:}n)$.

Математическая постановка задачи (*вариант 2*).

Потребуем, чтобы пропуск занятия обозначался цифрой 1, тогда исходные данные: n, k (те же, что и выше); числовая матрица $T(1{:}n, 1{:}k)$, где $t_{ij} = 1$ или 0.

Результат — тот же массив D.

Формулировка задачи: определить сумму элементов каждой строки матрицы $T(1{:}\ n,\ 1{:}\ k)$ и образовать массив $D(1{:}n)$.

Сравнивая теперь варианты постановки задачи, можем сделать вывод, что второй приводит к известной математической задаче и, следовательно, для него может существовать готовая программа.

Вариант 1 приводит к оригинальной программе, следовательно, он менее целесообразен.

Задача 13.5[1]

1. Содержательная постановка: мальчик бросил яблоко под углом к горизонту α, с начальной скоростью υ_0. Определить, какое расстояние пролетело яблоко.

2. Научная (физическая) постановка задачи. Определить, какое расстояние преодолеет тело, брошенное под углом α к горизонту с начальной скоростью υ_0, при следующих допущениях:

а) кривизной Земли пренебречь;

б) движением Земли пренебречь;

в) считать ускорение свободного падения постоянным;

г) сопротивлением воздуха пренебречь.

Мы заменили реальную задачу известной задачей физики.

3. Математическая постановка задачи.

а) Модель задачи: задана аналитически функция

$$y = Ax - Bx^2,$$

где $A = \operatorname{tg}\alpha$; $B = \dfrac{g}{2v_0^2 \cos^2 x}$.

Исходные данные: α, v_0.

б) Формулировка задачи: решить уравнение

$$Ax - Bx^2 = 0.$$

Здесь адекватность математической модели реальной ситуации определяется физической моделью задачи, так как последняя однозначно определяет МПЗ.

13.3. Формализация задачи

Начнем с примера. Пусть математическая постановка некоторой задачи сформулирована в таком виде: заданы n отрезков прямых в си-

[1] Задача заимствована из [1], но здесь предлагается иной вариант МПЗ, на наш взгляд, более правильный.

стеме координат на плоскости. Каждый отрезок задан координатами его конечных точек $x_н$, $y_н$ и $x_к$, $y_к$.

Определить отрезки, параллельные оси X или Y.

Очевидно, что при ручном, безмашинном решении этой задачи никаких затруднений не возникает — достаточно выявить отрезки, для которых выполняется условие:

$$x_н = x_к$$

или

$$y_н = y_к.$$

Однако для решения этой задачи на ЭВМ такой постановки недостаточно. Дело в том, что в задаче используется значительное число данных. При ручном решении безразлично, где и как они записаны; человек-вычислитель в любом случае способен прочитать их и использовать.

ЭВМ же может работать с данными только в том случае, если они организованы строго определенным образом. Так, во многих современных языках программирования (ЯП) данные должны быть представлены в виде значений отдельных переменных (A, B, C, ...) либо сгруппированы в массивы, как, например в языке Бейсик.

В нашей задаче данные можно представить, например, в виде матрицы $A(1{:}n, 1{:}4)$ либо в виде четырех одномерных массивов:

$$XN(1{:}n), XK(1{:}n), YN(1{:}n), YK(1{:}n).$$

В частности, можно представить данные в виде двух матриц:

$$K(1{:}n, 1{:}2), F(1{:}n, 1{:}2).$$

При этом в i-й строке матрицы K запишем $x_н$ и $x_к$ i-го отрезка, в той же строке матрицы V — $y_н$ и $y_к$ того же отрезка ($i = 1, 2, 3, ..., n$).

Остановимся на последнем варианте «структуры данных». В этом случае решение задачи сведется к определению номеров строк матриц K и V, для которых выполняется условие: $k_{i1} = k_{i2}$ или $v_{i1} = v_{i2}$, т.е. мы перешли к новой задаче, существенно отличающейся от первоначальной. Такое преобразование не редкость в практике. Чем оно вызвано в общем случае? Дело в том, что при выполнении математической постановки задачи не накладывается каких-либо ограничений на вид задачи.

В итоге модель объекта может включать самые различные математические объекты (линии, плоскости, системы уравнений, неравенств и т.д.), над которыми могут выполняться самые различные операции.

В то же время большинство ЯП, как отмечалось, могут работать лишь с числами, переменными и элементами массивов, над которыми можно выполнять в основном лишь ограниченный набор операций: арифметические; вычисления элементарных функций; сравнения величин (логические операции).

Преобразование задачи, полученной на этапе математической постановки, к такой задаче, которая вписывается в рамки ЯП, будем называть *формализацией задачи*.

Этот процесс обычно выполняется за два шага.

Шаг 1. Выбор *структуры данных*, т.е. представление данных и результатов задачи в виде, допустимом в ЯП (как правило, в виде совокупности переменных, массивов, файлов и т.д.).

Шаг 2. Выбор способа (порядка) решения задачи, который включал бы только те операции над элементами данных, которые допустимы в языке программирования.

Пример, которым начинается параграф 11.3, показывает, как от операций с отрезками прямых, отсутствующих в ЯП, мы перешли к операциям с матрицами.

Примечания. 1. Выбор структуры данных неоднозначен. Возможны различные ее варианты. Каждому варианту обычно соответствует свой метод решения задачи, более или менее сложный, а соответственно и разные по сложности и наглядности программы. Поэтому следует стремиться к выбору более целесообразной, оптимальной структуры данных.

2. Этап формализации задачи может отсутствовать, если МПЗ ее приводит к задаче, вписывающейся в рамки ЯП.

3. Вообще говоря, при решении любой задачи можно сразу перейти от содержательной постановки к формализации задачи (так обычно и стремятся поступать начинающие программисты), но в таком случае резко уменьшается шанс свести задачу к типовой математической задаче.

Примеры. Вернемся к задаче о кошках (см. задачу В. 1). Пусть требуется проверить пригодность для эксперимента не одной, а n кошек.

В этом случае можно прийти к такой математической задаче: заданы координаты n точек числовой оси X и отрезок $[a, b]$ этой оси. Определить, какие из n точек принадлежат этому отрезку.

На этапе формализации данные можно представить в виде совокупности переменных a, b и массива $X(1{:}n)$, тогда задача будет заключаться в выборе элементов x_i массивах, отвечающих условию: $a \le x_i \le b$ ($i = 1, 2, ..., n$).

Изменим теперь условия задачи, добавив еще одно ограничение — вес кошки не должен превышать w кг (соответственно должен быть задан вес каждой кошки).

Математическая постановка этой задачи: задана система координат на плоскости и n точек этой плоскости. Определить, какие из них лежат внутри приямоугольника $ABCD$ (рис. 13.3).

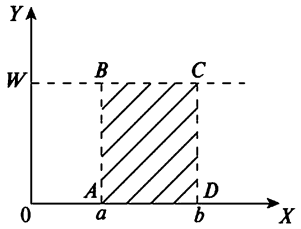


Рис. 13.3

Формализация задачи может иметь следующий вид.

1. Исходные данные: a, b, w, матрица $R(1{:}n, 1{:}2)$ (r_{i1} и r_{i2} — соответственно вес и длина хвоста i-й кошки) ($i = 1, 2, ..., n$).

Результат: $P(1{:}n)$ — порядковые номера кошек, отвечающих условиям задачи.

2. Метод решения задачи: выявить строки матрицы R, отвечающие условиям: $r_{i1} \le w$ и $a \le r_{i2} \le b$ ($i = 1, 2, 3, ..., n$), и записать их номера в массив P по порядку.

Необходимо отметить, что для многих типовых математических задач существуют методы решения, пригодные для использования их в современных языках программирования. Такие методы предлагает раздел математики, называемый «Вычислительная математика».

Этот раздел начал развиваться еще в «докомпьютерный» период и первоначально должен был обеспечивать человека-вычислителя методами решения различных математических задач с помощью только арифметических операций («численными методами»). Такие методы существуют, например, для вычисления определенного интеграла, элементарных функций, решения уравнений, систем уравнений и т.д.

В настоящее время эти методы используются для решения соответствующих задач на ЭВМ, поскольку все они полностью вписываются в рамки современных ЯП.

Как правило, для решения одной и той же задачи указанный раздел математики предлагает несколько подобных методов, различающихся точностью, трудоемкостью вычисления и т.д.

Так, для вычисления определенного интеграла существуют три основных метода: прямоугольников, трапеций и Симпсона.

Задача программиста в таком случае заключается в выборе наиболее подходящего метода. Рассмотрим еще ряд задач.

Задача 13.6

Содержательная постановка. Площадь садового кооператива разбита на отдельные участки треугольной формы, различные по площади. Определить площадь каждого участка.

Исходные данные: число участков n; длины сторон участков l_1, l_2, ..., l_m (единица измерения — 1 м).

Результаты: площади участков (м^2).

Математическая постановка задачи: заданы n произвольных треугольников длинами их сторон. Определить площадь каждого треугольника, используя формулу

$$s = \sqrt{p(p-a)(p-b)(p-c)}, \qquad (13.1)$$

где p — полупериметр треугольника; a, b, c — длины его сторон.

Если решать задачу вручную, без применения ЭВМ, то далее никаких проблем нет — достаточно для каждого треугольника вычислить полупериметр p, подставить его значение и значения длин сторон треугольника в формулу (13.1) — и все! Задача решена!

При решении задачи на ЭВМ следует организовать, представить данные в приемлемом для ЭВМ виде, т.е. этап формализации задачи необходим.

Формализация задачи. 1. Исходные данные: n, матрица $C(1{:}n, 1{:}3)$, в каждой ее строке — длины сторон одного из треугольников.

Результат: массив $S(1{:}n)$ — площади треугольников.

2. Метод решения задачи: для каждой строки матрицы S вычислить:

- сумму B ее элементов;
- величину $p = B/2$;
- величину $s_i = \sqrt{p(p-c_{i1})(p-c_{i2})(p-c_{i3})}$,

где i — номер строки ($i = 1, 2, 3, ..., n$).

Отметим, что выбранная структура данных избыточна, так как одна и та же величина может встречаться в матрице C несколько раз.

Если же представить данные иначе, например в виде массива $L(1{:}m)$, включив в него каждую величину однократно, то потребуется дополнительная информация, определяющая связь элементов массива L с номерами треугольников, т.е. выигрыша в объеме памяти для хранения данных мы не получим, но метод решения задачи будет явно сложнее.

Вернемся к задаче 13.1. Выполним математическую постановку ее.

Как отмечалось, математической моделью объекта этой задачи является цилиндр объема V, поэтому задача имеет чисто геометрический смысл:

определить радиус r и высоту h цилиндра заданного объема V, имеющего минимальную площадь поверхности s, при условии

$$R_{\min} \le r \le R_{\max}.$$

Применим известные из курса геометрии формулы:

$$V = \pi R^2 H; \tag{13.2}$$

$$S = 2\pi RH + 2\pi R^2. \tag{13.3}$$

Выразим H через V, используя (13.2), и подставим найденное для H выражение в (11.3), т.е. получим

$$H = \frac{V}{\pi R^2}; \tag{13.4}$$

$$S = \frac{V}{\pi R^2} 2\pi R + 2\pi R^2 = \frac{V}{R} + 2\pi R^2. \tag{13.5}$$

Далее, основываясь на методе решения задачи, приведенном в ее содержательной постановке, нужно вычислить значения функции $S = f_{(R)}$ по формуле (13.5) при разных значениях R, изменяя R от $R_{\min}$ до $R_{\max}$ с шагом ΔR, и выбрать наименьшее значение S (т.е. s), а также соответствующие ему значения R и H (т.е. r и h). При этом шаг определяется по формуле

$$\Delta R = (R_{\max} - R_{\min})/n.$$

Последнюю формулировку задачи можно рассматривать как формализацию ее.

Теперь рассмотрим вновь задачу 13.2. С математической точки зрения значения t_1, t_2, ..., t_n можно рассматривать как табличное задание функции $T = f_{(l)}$, где l — номер рабочего дня. Это и есть модель объекта задачи.

Решение задачи можно свести к построению графика функции T и выявлению убывающих участков графика.

В этом может и заключаться математическая постановка задачи.

Формализация задачи. 1. Исходные данные: n, массив $T(1{:}n)$.

Результат: массив $B(1{:}n-1)$, где b_i — номер рабочего дня, в котором допущено снижение выпуска ткани по сравнению с предшествующим днем ($i = 1, 2, ...; n - 1$). Очевидно, не все элементы этого массива получат числовые значения, так как число таких дней может быть меньше n — общего числа рабочих дней.

2. Метод решения задачи:
- образовать массив $D(1{:}n-1)$, где $d = t - t_{i+1}$ ($i = 1, 2, ..., n-1$), т.е. D — массив приращений производства ткани;
- образовать массив $B(1{:}n-1)$, записав в него номера отрицательных элементов массива D;
- вывести (отпечатать) все ненулевые элементы массива B.

В заключение следует еще раз подчеркнуть, что при математической постановке и формализации любой задачи возможны, как правило, варианты. В рассмотренных выше задачах приведены наиболее наглядные варианты постановок задач.

Контрольные вопросы

1. В чем суть содержательной постановки задачи?
2. Что такое математическая модель объекта задачи?
3. Выполнения каких операций требует математическая постановка задачи?
4. Почему следует сводить решение задачи к типовой математической задаче?
5. Что такое структура данных?

Задачи для самостоятельного решения

Для приведенных задач выполнить математическую постановку и формализацию.

1. Определить средний рост мальчиков и средний рост девочек одного класса. В классе 30 учеников. Задан рост каждого ученика.

2. Даны результаты экзамена по физике в 11-м классе. В классе 32 ученика. Определить:
   а) число школьников, сдавших на «хорошо» и «отлично»;
   б) составить список школьников, не сдавших экзамен.

3. Дан список участников соревнований по плаванию и их результаты в соревновании. Число участников — 20 человек. Определить:
   а) сколько из них выполнило норматив третьего разряда?
   б) лучший результат и фамилию победителя.

4. Даны сведения о количестве осадков, выпавших в каждый день месяца. Определить общее количество осадков за месяц.

5. Даны сведения о температуре воздуха за каждый день месяца. Определить:
   а) сколько дней в течение месяца температура опускалась ниже 0 °С?
   б) сколько дней температура была ниже среднемесячной.

6. Условия задачи 3. Расположить результаты и фамилии участников соревнования в соответствии с занятыми местами.

7. Даны даты рождения всех учеников класса в виде: год, номер месяца, число (например, 1969. 12. 15). Определить:

   а) число учеников, родившихся в декабре;
   б) число учеников, родившихся весной;
   в) дату рождения самого младшего ученика в классе.

8. Дан список книг с указанием фамилии автора, года издания и названия. Определить:

   а) число книг, изданных до 1980 г.;
   б) есть ли в списке книги Булгакова?
   в) сколько в списке книг Пушкина, Лермонтова и Некрасова, изданных до 1985 г.?

9. Какие существуют варианты размена купюры достоинством N руб. купюрами достоинством 100, 200 и 500 руб.?

ГЛАВА 14

ОСНОВЫ ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ ПАСКАЛЬ

Программа на языке Паскаль содержит как минимум две основные части:

- объявление данных, с которыми будут производиться некоторые действия (вычисления, вывод на экран и т.п.);
- операторную часть, которая задает последовательность операций, выполняемых с этими данными. В Паскале она начинается словом **begin**. Разделителем между операторами является символ «;»

Программа начинается с необязательного заголовка, состоящего из служебного слова **PROGRAM**, после которого следует имя программы, данное ей разработчиком.

В Паскале могут быть использованы следующие виды данных:

константа — постоянная величина, имеющая имя;

переменная — изменяющаяся величина, имеющая имя (ячейка памяти).

Имена программ, констант и переменных могут включать латинские буквы (A – Z), причем заглавные и строчные буквы не различаются, цифры, знак подчеркивания.

Имена НЕ могут включать русские буквы, пробелы, скобки, знаки +, =, !, ?

В программе на Паскале могут быть использованы:

процедура — вспомогательный алгоритм, описывающий некоторые действия (рисование окружности);

функция — вспомогательный алгоритм для выполнения вычислений (вычисление квадратного корня, **sin**).

Объявления переменных необходимы для того, чтобы во время трансляции программы выделить в памяти место для хранения переменных и закрепить за именами адреса ячеек памяти. Тип переменной указывается потому, что переменные имеют различное представление в памяти машины, разный диапазон изменения и даже машинные команды для выполнения арифметических операций с целочисленными и вещественными переменными требуются разные.

Стандартные типы данных. При объявлении переменных используются следующие типы данных:

Integer, ShortInt, LongInt, Word, Byte — целочисленные типы (значениями переменных этого типа могут быть только целые числа);

char — литерный (символьный) тип данных, предназначенный для хранения одного алфавитно-цифрового символа (буквы, цифры, знака препинания и т.п.);

boolean — булевский (логический) тип, имеющий два возможных значения true (истина) и false (ложь);

real, double, single, extended, comp — вещественные типы (в переменную такого типа можно записать вещественное число, т.е., ее значение может иметь целую и дробную часть);

string — строковый тип, предназначенный для хранения строки не более, чем из 255 символов;

pointer — указатель, переменные этого типа предназначены для хранения адреса ячейки памяти;

text — этот тип данных определяет текстовый файл (то есть, текст, записанный во внешней памяти и снабженный хранящимся в каталоге файлов именем).

Целочисленный, литерный и булевский типы данных называют также дискретными или порядковыми (*ordinal*) типами. Это связано с тем, что все возможные значения каждого из этих типов можно пронумеровать и упорядочить (расположить в порядке «возрастания»). Таким образом, для значения порядкового типа можно указать предыдущее и последующее значения.

Операторная часть программы всегда начинается словом **begin** и заканчивается словом **end.**

Она состоит из разделенных точками с запятой операторов (предложений) языка. Процесс выполнения программы заключается в том, что компьютер последовательно читает операторы (обычно в том порядке, в каком они записаны в тексте программы) и выполняет предписанные ими действия.

Программа должна взаимодействовать с пользователем: выводить на экран результаты расчетов, используя клавиатуру дисплея вводить необходимые для выполнения этих расчетов исходные данные. Выводит данные на экран оператор **write**. После слова **write** в скобках перечисляются через запятую элементы так называемого «списка вывода»: строки текста, имена переменных и констант, значения которых при выполнении оператора выведутся на экран. Таким образом, выполнение оператора **write ('Введите количество чисел ->')** приведет к появлению на экране строки текста

Введите количество чисел ->.

Оператор **write (Osn)** вывел бы на экран значение константы **Osn**. Строка текста всегда ограничивается символами ` (апостроф).

Оператор **readln (N)** позволяет ввести значение N с клавиатуры. При его выполнении программа приостанавливает свою работу. Следующий оператор не будет выполняться до тех пор, пока человек не введет с клавиатуры строку символов, закончив ее нажатием клавиши **Enter**.

Стандартные функции и процедуры. В математических функциях аргумент задается вещественным или целым числом, а результат всегда получается вещественным:

Sin(X) вычисление синуса;

Arctan(X) вычисление арктангенса;

Exp(X) —возведение числа *e* в степень X;

Ln(X) — натуральный логарифм числа X;

Sqr(X) — квадрат числа X;

Sqrt(X) — квадратный корень из числа X.

В языке Паскаль нет операции возведения числа M в степень X. Применение стандартных функции вполне позволяют решить эту задачу даже для не целых M и X. Для этого достаточно написать

Result:=exp(X*ln(M)).

Функция **Round(X)** округляет вещественное число до ближайшего целого и возвращает результат целого типа.

Функция **Trunc(X)** получает целочисленный результат отбрасывая дробную часть X, то есть Trunc(2.7) возвратит значение 2.

Функция **Int(X)** делает то же самое, но возвращает результат вещественного типа. То есть Результатом выполнения функции Trunc(2.7) будет целое число 2, а результатом выполнения Int(2.7) будет вещественное число 2.0.

Функция **Frac(X)** вычисляет дробную часть числа X.

Процедура отличается от функции тем, что результат ее работы заносится в переменную, заданную одним из параметров.

Оператор присваивания. Оператор присваивания применим к любому типу данных, кроме файлового.

Оператор имеет следующий вид: **step: =step*osn**

В правой части оператора записывается выражение — сочетание констант, переменных и функций, связанных знаками операций.

Арифметические и логические операции:

+ сложение;

– вычитание;

***** умножение целых и вещественных чисел;

/деление вещественных чисел;

div — деление целых чисел;

mod — остаток от деления целых чисел (знак остатка совпадает со знаком делимого);

and — логическое умножение;

shr — логический сдвиг вправо;

shl — логический сдвиг влево;

or — логическое сложение;

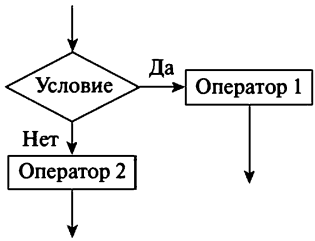
xor — сложение по модулю два.

Операции отношения:

< — меньше; > — больше; = — равно; <> — не равно; <= — меньше или равно; >= — больше или равно;

in — принадлежность множеству.

Оператор условного перехода



Выполнение оператора заключается в следующем:

вычисляется условие; если выражение истинно, выполняется оператор 1, если ложно — оператор 2.

Если используется краткая форма оператора (минуя ветвь **else**), то при ложности выражения не выполняется никаких действий.

Применим полную форму оператора для определения наибольшего из трех чисел. Конструкция, записывающая в переменную M большее из трех чисел a, b, c, имеет вид:

if a > b **then** $\underbrace{\text{if a>c then M:=a else M:=c}}_{\text{Оператор 1}}$ **else** $\underbrace{\text{if b>c then M:=b else M:=c}}_{\text{Оператор 2}}$.

Как видим, она вполне соответствует синтаксису оператора условного перехода, но ее довольно трудно понять при первом чтении. В таких случаях удобнее записывать составные части с новой строки:

```
if a>b then
if a>c then M:=a else M:=c
else
if b>c then M:=b else M:=c.
```

Заметим, что, используя краткую форму, эту задачу можно решить даже с меньшим количеством проверок:

```
M:=a; {выигрываем на том, что не раздумывая назначили
большим числом}
if b>M then M:=b;
if c>M then M:=c;.
```

Метки и безусловные переходы. Перед любым оператором программы можно поставить метку. Метка — это имя или целое число, отделенное от оператора двоеточием. Все метки, которые будут использоваться в программе, должны быть перечислены в разделе, начинающемся со слова **label**.

Метки используются в операторе безусловного перехода **goto**, предназначенном для изменения порядка выполнения операторов.

После выполнения оператора **goto** следующим выполняется оператор, возле которого помещена соответствующая метка. Нельзя одну метку ставить возле разных операторов, а вот два оператора **goto** могут передавать управление на одну метку. Заметим, что оператор перехода может передавать управление внутрь тела цикла, минуя его заголовок, а также из тела цикла. Например, следующая программа

```
PROGRAM Метка;
label 1, M1;
var i, r: integer;
begin
  i:=5; goto 1;
  for i:=1 to 10 do
    1: begin writeln (i); goto M1 end;
    M1: write ('конец цикла')
end.
```

выведет на экран одно число 5 и сообщение «конец цикла».

Операторы цикла. Оператор:

```
for i:=1 to N do
begin
  writeln (Step: 5:0);
  step:=step*osn
end;,
```

называется оператором цикла и состоит из двух частей.

```
begin
  writeln (Step: 5:0);
  step:=step*osn
end;,
```

Заключенная в операторные скобки **begin...end** называется телом цикла и выполняет следующие действия:

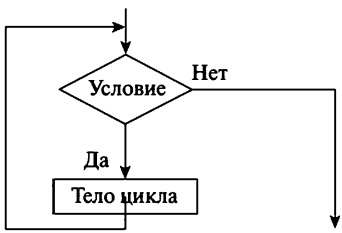
— выводит на экран значение переменной **Step**;
— вычисляет значение произведения **step*osn** и то, что при этом получилось, записывает в переменную **step**.

Первая строка оператора цикла **for** i: =1 **to** N **do** называется заголовком цикла и предписывает выполнить следующие действия:

— присвоить переменной **i** значение **1**;
— после этого многократно повторять выполнение тела цикла, увеличивая после каждого такого выполнения переменную **i** на единицу.

Выполнение оператора заканчивается, как только значение переменной **i** превысит значение также указанной в заголовке переменной **N**.

Оператор цикла с предусловием.



Выполнение оператора заключается в многократном повторении двух действий:

- вычислении логического выражения, записанного в заголовке цикла;
- выполнении оператора, являющегося телом цикла.

Эти действия повторяются, пока выражение в заголовке не станет ложным. Проверка истинности выражения производится до первого выполнения оператора, т.е., если выражение сразу ложно, то тело цикла не выполнится ни разу.

Задача.

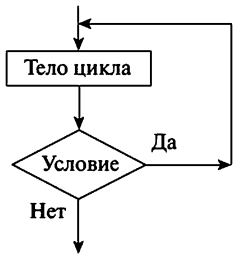
Вывести на экран все значения 2^{-i}, превышающие число MIN.

Решение.

```
PROGRAM M_min_N;
const                              {раздел объявления констант}
  Osn=2;
Var                               {раздел объявления переменных}
  Step: real;
  Min: real;
```

```
BEGIN                    {начало операторной части программы}
  Step:=1;
  write ('Введите предельное значение -> ');
  readln (Min);                {если MIN > 1 не выведется
                                              ни одного числа}
  writeln ('Отрицательные степени числа ',Osn);
  While Step>=Min do       {Цикл закончится при нарушении
                                                   условия}
    begin
      writeln(Step:5:4);
      Step:=Step/Osn
    end;
  readln
END.
```

Оператор цикла с постусловием.



Выполнение оператора заключается в многократном повторении двух действий:

- выполнении оператора, являющегося телом цикла;
- вычислении условия, записанного в заголовке цикла.

Эти действия повторяются, пока выражение в заголовке не станет истинным. Проверка истинности выражения производится после выполнения тела цикла, т.е. если выражение сразу истинно, то тело цикла выполнится один раз.

Задача

Найти значение наибольшего общего делителя.

Решение

```
var
bigger, smaller, r: integer;
BEGIN
  readln (bigger, smaller);
  if bigger<smaller then     {записываем в bigger большее
                                              из двух чисел}
    begin
      r:=smaller; smaller:=bigger; bigger:=r;
```

```
    end;
    repeat
      r:=bigger mod smaller;          {делим большее число
                                              на меньшее,}
    bigger:=smaller; {потом делитель назначаем делимым,}
    smaller:=r                          {а остаток делителем}
    until r=0;            {и так до тех пор, пока остаток
                                    не станет равным нулю}
    writeln (bigger);    {последнее участвующее в делении
                        значение делителя (его переписали
                        в переменную bigger, еще не зная,
                          что остаток r равен нулю) — это
                       и есть наибольший общий делитель.}
END.
```

Задача.

Вычисление суммы $\sum(1/2*i)$ до тех пор, пока она не станет больше заданного числа N:

Решение.

```
PROGRAM repeat1;
var
  Sum, Max: single;
  i: longint;            {Количество слагаемых может быть
                                          очень большим}
  begin
    write (',Введите предельное значение -> ');
    readln (Max);
    Sum:=0; i:=1;
    Repeat
      Sum:=Sum+1/(2*i);
      writeln (i: 7,Sum: 10:3);
      i:=i+1;                {Succ (i) не останавлиается
                               на макс. полож. значении}
    Until Sum>=Max;                    {Цикл закончится
                                  по выполнении условия}
      writeln ('Количество слагаемых ',i-1);
      writeln ('Сумма ',Sum: 7:5);
      writeln ('Последнее слагаемое ',1/(2*(i-1)):7:7);
    readln
  end.
```

Одномерные массивы. Массив в языке Паскалъ — это совокупность фиксированного числа однотипных элементов объединенных общим именем.

Индекс массива определяет число его элементов (при объявлении массива) и номер элемента (в операторной части программы), как и параметр цикла, он должен иметь порядковый тип (целый, литерный, перечисляемый, булевский или интервальный). Элементы массива могут иметь любой тип, кроме файлового (с последним мы пока не знакомы).

Примеры объявления массивов:

```
Mas: array [1..5] of real;      {Массив из 5 вещественных
                                    чисел, индекс массива
                                  имеет интервальный тип}
Ms = array [char] of integer;       {Массив из 256 целых
                                    чисел, индекс массива
                                     имеет литерный тип};
```

К массиву в целом применим оператор присваивания, при этом все элементы одного массива запишутся в соответствующие элементы другого массива.

Для обращения к отдельному элементу массива после имени нужно указать значение индекса элемента:

```
Mas1 [1] — первый элемент массива Mas1 (целое число);
```

Допустимые операции с элементом определяются типом элемента.

Рассмотрим действия, которые можно выполнять с объявленными выше массивами в операторной части программы.

```
begin
  M2:=M1;                        {К массиву в целом применим
                                     оператор присваивания,}
  Mas1:=Mas2;                 {при этом все элементы одного
                          массива запишутся в соответствующие
                                   элементы другого массива.}
  M1[5]:=M1[5]+1;             {определяются типом элемента}
end.
```

Задача

Составить программу, которая вводит в массив данные с клавиатуры, после чего определяет значение и индекс (номер) максимального элемента массива.

Решение

```
PROGRAM MASSIV1;
const
  r=10;                                        {Размер массива}
type
  M=array [1..r] of real;
```

```
var
  Mas: M; J: integer;
  Max:: real;          {Значение максимального элемента}
  I_Max: integer:;       {Позиция максимального элемента
                                              в массиве}
BEGIN
  for J:=1 to r do
  begin
    write ('Введите Mas[',J, ']'); readln (Mas[j])
  end;
  Max:=Mas[1]; I_Max:=1;            {Сначала считаем, что
                                   максимальный — первый}
  for J:=2 to r do
    if Mas[j]>Max then   {Потом проверяем все остальные}
      begin Max:=Mas[J]; I_Max:=J; end;
      writeln ('Номер максимального элемента = ',I_Max)
END.
```

Многомерные массивы.

Пусть массив Tabl содержит четыре элемента и каждый элемент в свою очередь является массивом из трех вещественных чисел.

Объявить такой массив можно следующим образом:

```
type M1 = array [1..3] of real;
  M2 = array [1..4] of M1;
var
  Tabl:M2;
  Mas1:M1;
```

К элементам массива Tabl можно обращаться указывая, как обычно, их номер: Tabl[1], ... Tabl[4]. Эти элементы можно использовать в операторе присваивания: Mas1:=Tabl[1]. Но каждый из этих элементов, например **Tabl[2]**, является массивом из трех вещественных чисел. Указывая индекс в массиве **Tabl[2]** можно обратиться к его элементу — вещественному числу: **Tabl[2]** [1] — первый элемент массива Tabl[2], **Tabl[2]** [2] — второй, и т.д. Двумерными массивами удобно представлять в программе матрицы. Обычно считается, что первый индекс задает номер строки, а второй номер столбца матрицы.

Если в рассмотренном выше примере компоненты данных типа M1 описать не как числа, а тоже как массивы, получим трехмерный массив. Аналогично можно строить массивы большей размерности.

Задача

В программе объявляется одномерный массив (вектор) Mas из 25 элементов и матрица Matr размером 5×5.

Решение

Элементы матрицы сначала построчно записываются в вектор Mas, а потом ее первая и четвертая строка меняются местами:

```
const
  r = 5;
type
  M1=array [1..r] of real;
  M=array [1..r*r] of real;
  Mtr=array [1..r] of M1;
const
  Matr:Mtr=
    ((1,2,3,4,5),                    {Matr[1,1] — Matr[1,5]}
    (0,0,0,0,0),
    (6,7,8,9,10),
    (1,1,1,1,1),
    (0,0,0,0,0));                    {Matr[5,1] — Matr[5,5]}
var
  Mas:M;
  Mas1:M1;
  I,J:integer;
BEGIN
  for I:=1 to r do
  for J:=1 to r do Mas [(I-1)*r+J]:=Matr[I,J];
  Mas1:=Matr[1];                              {Меняем местами
  Matr[4]:=Mas1;                  первую и четвертую строки}
        {Mtr(Mas):=Matr; чтобы не забыть — потом убрать}
END.
```

Задача

Вывести элементы матрицы на экран, обходя их по спирали, двигаясь начиная с верхнего левого угла по часовой стрелке.

Решение

```
{$R+}
const
  m=3;
  n=3;
  A:array [1..m, 1..n] of integer=((5,3,7),
  (5,3,-1), (-2,3,7));
var
  V:array [1..m] of integer;
  i, j, l, k: integer;
BEGIN
  writeln ('Matrix');
  for i:=1 to m do                    {вывод матрицы в виде
                                     прямоугольной таблицы}
```

```
  begin
    writeln;
    for j:=1 to n do write (A[i,j]:6);
  end;
  writeln;
    if m<n then k:=m else k:=n;
    k:= k div 2+k mod 2;
    for l:=1 to k do              {вывод элементов матрицы
                                                по спирали}
    begin
      for j:=l to n-l+1 do write (A[l,j]:5);
      for i:=l+1 to m-l+1 do write (A[i,n-l+1]:5);
      for j:=n-l downto l do write (A[m-l+1,j]:5);
      for i:=m-l downto l+1 do write (A[i,l]:5);
    end;
  readln;
END.
```

ЛИТЕРАТУРА

1. *Бройдо В.Л.* Вычислительные системы, сети и телекоммуникации. СПб. : Питер, 2002.
2. Информатика и информационные технологии : учебник для бакалавров / Гаврилов М.В., Климов В.А. 2-е изд., испр. и доп. М. : Юрайт, 2012.
3. Информатика : учеб. пособие для среднего профессионального образования (+CD) / под общ. ред. И.А. Черноскутовой. СПб. : Питер, 2005.
4. Информатика : задачник-практикум : в 2 т. / под ред. И.Г. Семакина, Е.К. Хеннера. М. : Лаборатория базовых знаний, 2003.
5. *Иопа Н.И.* Информатика для технических специальностей. М. : КНОРУС, 2011.
6. *Кузнецова О.С.* Информатика. Краткий курс (3-е изд.). М. : Окей-книга, 2011.
7. *Ляхович В.Ф.* Основы информатики. Ростов-н/Д : Феникс, 2001.
8. *Макарова Н.В., Матвеева Л.А., Бройдо В.Л.* Информатика : учеб. пособие. М. : Финансы и статистика, 2006.
9. *Михеева Е.В., Титова О.И.* Информатика : учебник для учреждений среднего проф. образования. 5-е изд., испр. М. : Academia, 2010.
10. *Молодцов В.А., Рыжикова Н.Б.* ЕГЭ — это очень просто! Информатика. Ростов н/Д : Феникс, 2009.
11. *Семакин И.Г.* и др. Информатика. 10-й, 11-й класс. М. : Лаборатория базовых знаний, 2002.
12. *Симонович С.* и др. Общая информатика : учеб. пособие для средней школы. М. : АСТ-ПРЕСС, 2006.
13. *Угринович Н.Д.* Информатика и информационные технологии : учеб. пособие. М. : Лаборатория базовых знаний, 2007.
14. *Хлебников А.А.* Информатика : учебник для СПО. 2-е изд. Ростов-н/Д : Феникс, 2010.