ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

УПРАВЛЕНИЕ ДИСТАНЦИОННОГО ОБУЧЕНИЯ И ПОВЫШЕНИЯ КВАЛИФИКАЦИИ

Кафедра «Вычислительные системы и информационная безопасность»

Учебное пособие

по дисциплине
«Информатика и ИКТ»

Краткий курс «Информатика и ИКТ» Часть 1

Авторы
Полуян А.Ю., Петренкова С.Б.,
Смирнова О.В., Басова А.В., Пурчина О.А.

Ростов-на-Дону, 2017

Аннотация

В учебном пособии кратко излагается курс дисциплины «Информатика и ИКТ», начиная с возникновения информатики как науки и заканчивая прикладными аспектами алгоритмизации и программирования. В книге выделены, с нашей точки зрения, самые основные моменты, на которые следует обратить внимание при изучении курса Информатика и ИКТ. Пособие предназначено для студентов технических вузов, колледжей и техникумов.

Авторы

к.т.н., доцент Полуян А.Ю.,
к.п.н., доцент Петренкова С.Б.,
к.т.н., доцент Смирнова О.В.,
к.т.н., доцент Басова А.В.,
ст. преподаватель Пурчина О.А.

Оглавление

1. ВВЕДЕНИЕ В ИНФОРМАТИКУ

На данный момент наука информатика является одной из самых быстро развивающихся областей знания. Любой учебник, освещающий последние достижения компьютерной техники, устаревает еще на этапе издания. Каждый лектор, преподающий эту дисциплину, знает – нельзя оставить курс лекций без изменений на очередной учебный год. Ведь за это время возникли новые периферийные устройства, увеличилось быстродействие процессоров, увеличился объем оперативной памяти выпускаемой вычислительной техники. По этой причине авторами избран путь краткого изложения основополагающих фактов, составляющих основу дисциплины «Информатика» и не изменяющихся со временем.

Начнем с того факта, что информатика – это наука, связанная не только с компьютерами и их программным обеспечением. На самом деле информатика изучает процессы, связанные с хранением и обработкой любой информации на любых носителях.

В этом контексте отмечают четыре исторических этапа в развитии информатики:

- В V тысячелетии до нашей эры появилась письменность. При этом носителями информации стали камни, глиняные таблички, папирус, пергамент, а во II веке до н.э. появилась и бумага.
- В XV веке было изобретено книгопечатание (в Европе в XV веке Гутенбергом и в XVI веке Иваном Федоровым). Как следствие этого информация стала доступной для широких масс.
- С начала XX века информация стала распространятся с помощью технических средств - телеграф, телефон, радио, телевидение.
- В середине XX столетия был изобретен компьютер и обработка информации стала автоматической.

Существует несколько определений понятия термина «информатика». Приведем некоторые из них.

Определение. *Информатика* - это название фундаментальной науки, изучающей общие свойства информации, процессы, методы и средства ее обработки (сбор, хранение, преобразование, перемещение, выдача).

Определение. *Информатика* - это наука о преобразовании информации, которая базируется на вычислительной технике. Состав информатики – это три неразрывно и существенно связанные составные части: технические средства, программные и алгоритмические.

Определение. *Информатика* – это наука о проблемах обработки различных видов информации, создании новых видов высокоэффективных ЭВМ, позволяющая представлять человеку широкий спектр информационных ресурсов.

Как следует из всех этих определений термином «информатика» обозначают совокупность научных направлений, изучающих информацию, информационные процессы в природе, обществе, технике, способы представления, накопления, обработки и передачи информации с помощью технических средств.

Условно дисциплину информатика можно разделить на несколько составных частей:

- *Теоретическая информатика* – часть информатики, занимающаяся изучением структуры и общих свойств информации и информационных процессов. Основана на использовании математических методов и включает ряд математических разделов (теория алгоритмов, теория автоматов, теория кодирования, математическая логика, исследование операций).
- *Вычислительная техника* – раздел, в котором описываются общие принципы построения вычислительных систем. Речь идет о принципиальных решениях на уровне архитектуры вычислительных систем, определяющей состав, назначение, функциональные возможности и принципы взаимодействия устройств
- *Программирование* – деятельность, связанная с разработкой систем программного обеспечения.
- *Информационные системы* – раздел информатики, связанный с решением вопросов по анализу потоков информации в различных сложных системах, их оптимизации, структурировании, принципах хранения и поиска информации.
- *Искусственный интеллект* – область информатики, в которой решаются сложнейшие проблемы, находящиеся на пересечении с психологией, физиологией, лингвистикой и другими науками. Основные направления разработок в этой области – моделирование рассуждений, компьютерная лингвистика, машинный перевод, создание экспертных систем и другие.

2. ОСНОВНОЕ ПОНЯТИЕ ИНФОРМАТИКИ – ИНФОРМАЦИЯ

Любая деятельность человека представляет собой процесс сбора и переработки информации, принятия на ее основе решений и их выполнения. Термин *информация* произошел от латинского слова *information* – разъяснение, осведомление.

Определение. *Информация* – сведения об объектах и явлениях окружающей среды, их параметрах, свойствах и состоянии, которые уменьшают имеющуюся в них степень неопределенности, неполноты знаний.

Современная наука о свойствах информации и закономерностях информационных процессов называется теорией информации.

Определение. Материальный объект или среду, которые служат для представления или передачи информации, будем называть ее *материальным носителем*.

Свойства информации

Понятие «информация» имеет большое количество разнообразных свойств, но для дисциплины информатика наиболее важными являются следующие свойства:

1. **Дуализм** характеризуется двойственностью информации. С одной стороны, информация объективна в силу объективности данных. С другой стороны – субъективна, в силу субъективности применяемых методов. Иными словами, методы могут вносить в большей или меньшей степени субъективный фактор и т.о. влиять на информацию в целом. Например, два человека читают одну и ту же книгу и получают подчас весома разную информацию, хотя прочитанный текст, т.е. данные, были одинаковы. Более объективная информация применяет методы с меньшим субъективным элементом.

2. **Полнота** информации характеризует степень достаточности данных для принятия решения или создания новых данных на основе имеющихся. Неполный набор данных оставляет большую долю неопределенности, т.е. большое число вариантов выбора, а это потребует применения дополнительных методов, например, экспертных оценок, бросания жребия и т.п. Избыточный набор данных затрудняет доступ к нужным данным, создает повышенный информационный шум, что также вызывает необходимость дополнительных методов, например, фильтрацию, сортировку. И неполный и избыточный наборы данных затрудняют получение информации и принятие адекватного решения.

3. **Достоверность** – это свойство, характеризующее степень соответствия информации реальному объекту с необходимой точностью. Измеряется достоверность информации *доверительной вероятностью* необходимой точности, т.е. вероятностью того, что отображаемое информацией значение параметра отличается от истинного значения этого параметра в пределах необходимой точности.

4. **Адекватность** информации выражает степень соответствия создаваемого на основе информации образа реальному объекту, процессу, явлению. Адекватность информации может выражаться в трех формах:

– *Синтаксическая* адекватность отображает формально-структурные характеристики информации и не затрагивает ее смыслового содержания. На синтаксическом уровне учитываются тип носителя и способ представления информации, скорость передачи и обработки, размеры кодов представления информации, надежность и точность преобразования этих кодов и т.п. Информацию, рассматриваемую только с синтаксических позиций, обычно называют данными, т.к. при этом не имеет значения смысловая сторона. Эта форма способствует восприятию внешних, структурных характеристик;

– *Семантическая* (смысловая) адекватность определяет степень соответствия образа объекта и самого объекта и предполагает учет смыслового содержания информации. Эта форма служит для формирования понятий и представлений, выявления смысла, содержания информации и ее обобщения;

– *Прагматическая* (потребительская) адекватность отражает отношение информации и ее потребителя, соответствие информации цели управления, которая на ее основе реализуется. Прагматические свойства информации проявляются только при наличии единства информации (объекта), пользователя и цели управления. Прагматический аспект рассмотрения связан с ценностью, полезностью использования информации при выработке потребителем решения для достижения своей цели.

5. **Доступность** информации, Информация должно быть доступна восприятию пользователя. Доступность обеспечивается выполнением соответствующих процедур ее получения и преобразования. Например, в ИС информация преобразовывается к доступной и удобной для восприятия пользователя форме.

6. **Актуальность информации**. Информация существует во времени, т.к. существуют во времени все информационные процессы. Информация, актуальная сегодня, может стать совер-

шенно ненужной по истечении некоторого времени. Например, программа телепередач на нынешнюю неделю будет неактуальна для многих телезрителей на следующей неделе.

Определение. Изменение характеристики носителя, которое используется для представления информации, называется *сигналом*, а значение этой характеристики, отнесенное к некоторой шкале измерений, называется *параметром сигнала.*

Определение. Последовательность сигналов называется *сообщением.*

Сообщение, таким образом, служит переносчиком информации, а информация является содержанием сообщения. При этом изменение с течением времени содержания информации или представляющего его сообщения будет представлять собой информационный процесс. К основным видам информационных процессов можно отнести создание, преобразование, уничтожение и передачу информации.

Понятие «информация» обычно предполагает наличие двух объектов – источника и приемника информации. Информация передается от источника к приемнику в материально-энергетической форме в форме сигналов, распространяющихся в определенной среде.

Определение. *Источник информации* – это субъект или объект, порождающий информацию и представляющий ее в виде сообщения.

Определение. *Получатель информации* - это субъект или объект, принимающий сообщение и способный правильно его интерпретировать.

Итак, информация передается в форме сообщений от некоторого источника информации к получателю посредством системы связи между ними.

Совокупность технических средств, используемых для передачи сообщений от источника к получателю, называется *системой связи***.**

Определение. *Канал связи* – совокупность технических устройств, обеспечивающих передачу сигнала от передатчика к приемнику.

Определение. *Кодирующее устройство* предназначено для кодирования информации (преобразования исходного сообщения от источника к виду, удобному для передачи информации).

Определение. *Декодирующее устройство* предназначено для преобразования полученного сообщения в исходное.

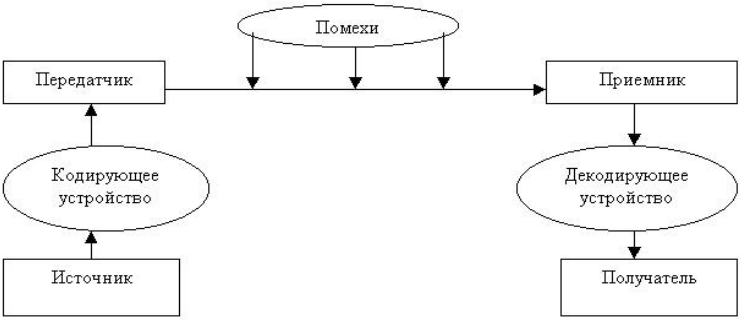


Рис. 1. Схема передачи информации.

Выше говорилось о том. что передача информации производится с помощью сигналов. Их различают два вида: аналоговые и дискретные.

Определение. Сигнал называется *непрерывным (или аналоговым)*, если его параметр может принимать любое значение в пределах некоторого интервала (речь, музыка).

Определение. Сигнал называется *дискретным,* если его параметр может принимать конечное число значений в пределах некоторого интервала.

Пример дискретного сообщения – процесс чтения книги, информация в которой представлена текстом, то есть дискретной последовательностью отдельных букв. Примеров непрерывной информации служит человеческая речь, передаваемая модулированной звуковой волной, параметром сигнала в этом случае является давление, создаваемое этой волной в точке нахождения приемника - человеческого уха.

3. ИЗМЕРЕНИЕ ИНФОРМАЦИИ

При реализации информационных процессов всегда происходит перенос информации в пространстве и времени от источника информации к приемнику. При этом для передачи информации используют различные знаки или символы, например естественного или искусственного (формального) языка, позволяющие выразить ее в форме сообщения. Для измерения информации вводятся два параметра: объем информации и количество информации.

Если информация передается в виде сообщения, представляющего собой совокупность символов какого-либо алфавита, то каждый новый символ в сообщении увеличивает количество информации в нем. Если теперь количество информации, содержащейся в сообщении из одного символа, принять за единицу, то объем информации (данных) *V* в любом другом сообщении будет равен количеству символов (разрядов) в этом сообщении. Так как одна и та же информация может быть представлена многими разными способами (с использованием разных алфавитов), то и единица измерения информации (данных) соответственно будет меняться.

В компьютерной технике наименьшей единицей измерения информации является 1 бит. Таким образом, объем информации, записанной двоичными знаками (0 и 1) в памяти компьютера или на внешнем носителе информации подсчитывается просто по количеству требуемых для такой записи двоичных символов. Например, восьмиразрядный двоичный код 11001011 имеет объем данных V= 8 бит.

В современной вычислительной технике наряду с минимальной единицей измерения данных «бит» широко используется укрупненная единица измерения «байт», равная 8 бит. При работе с большими объемами информации для подсчета ее количества применяют более крупные единицы измерения, такие как килобайт (Кбайт), мегабайт (Мбайт), гигабайт (Гбайт), терабайт (Тбайт):

1 Кбайт = 1024 байт = 2^{10} байт;

1 Мбайт = 1024 Кбайт = 2^{20} байт = 1 048 576 байт;

1 Гбайт = 1024 Мбайт = 2^{30} байт = 1 073 741 824 байт;

1 Тбайт = 1024 Гбайт = 2^{40} байт = 1 099 511 627 776 байт.

Используя методику такого рода, можно достаточно легко вычислить объем информации, содержащейся в некотором сообщении. Однако, к сожалению, этот подход удается использовать не всегда.

Часто приходится иметь дело с явлениями, исход которых неоднозначен и зависит от факторов, которые мы не знаем или не можем учесть. Например – результат бросания игральной кости. В этом случае измерить количество информации, полученное в результате эксперимента, несколько сложнее.

Прежде, чем раскрыть существо вероятностного метода вычисления количества информации дадим несколько определений.

Определение. События, о которых нельзя сказать произойдут они или нет, пока не будет осуществлен эксперимент, называются *случайными.*

Раздел математики, в котором строится понятийный и математический аппарат для описания случайных событий, называется теорией вероятности.

Определение. Осуществление некоторого комплекса условий называется *опытом*, а интересующий нас исход этого опыта – *благоприятным событием*.

Определение. *Вероятностью события A* называется отношение числа равновозможных исходов, благоприятствующих событию *A*, к общему числу всех равновозможных исходов.

$$P(A) = \frac{m}{n}.$$

Определение. *Энтропия H* – мера неопределенности опыта, в котором проявляются случайные события.

Очевидно, что величины *H* и *n* (число возможных исходов опыта) связаны функциональной зависимостью: *H=f(n)*, то есть мера неопределенности есть функция числа исходов.

Некоторые **свойства** этой функции:

1. $f(1) = 0$, так как при $n=1$ исход не является случайным и неопределенность отсутствует.

2. $f(n)$ возрастает с ростом n, так как чем больше возможных исходов, тем труднее предсказать результат, и, следовательно, больше неопределенность.

3. если α и β два независимых опыта с количеством равновероятных исходов n_α и n_β, то мера их суммарной неопределенности равна сумме мер неопределенности каждого из опытов:

$$f(n_\alpha n_\beta) = f(n_\alpha) + f(n_\beta)$$

Всем трем этим свойствам удовлетворяет единственная функция – $\log(n)$. То есть за меру неопределенности опыта с n равновероятными исходами можно принять число $\log(n)$. В силу известной формулы можно перейти от логарифма по одному основанию к логарифму по любому другому основанию. Таким об-

разом, выбор основания значения не имеет. Произвольно выбрав основание, равное 2, получим формулу Хартли:

$H=\log_2 n$.

Если исходы опыта не равновероятны, справедлива формула Шеннона:

$$H = \sum_{i=1}^{n} p_i \log_2 \frac{1}{p_i},$$

где p_i-вероятность i-того исхода.

Теперь перейдем к нашей цели – определению количества информации. Из определения энтропии следует, что энтропия это числовая характеристика, отражающая степень неопределенности некоторой системы.

После проведения опыта, то есть после получения информации, естественно, какая-то часть неопределенности исчезнет. И чем больше информации мы получим при проведении опыта, тем меньше неопределенности останется. Таким образом, уменьшение энтропии является количественной мерой информации:

$I=H1-H2=\log_2 n_1-\log_2 n_2=\log_2 (n_1/n_2)$,

где $H1$ – энтропия до проведения опыта, $H2$ – энтропия после проведения опыта.

Ситуация напоминает закон Архимеда: тело, погруженное в жидкость, выталкивает ровно тот ее объем, который занимает само. Так и информация вытеснит энтропию в количестве, равном своему.

Очевидно, что в случае, когда получен конкретный результат, $H2=0$, и, таким образом, количество полученной информации совпадает с начальной энтропией и подсчитывается при помощи формулы Хартли.

Итак, мы ввели меру неопределенности – энтропию и показали, что начальная энтропия (или уменьшение энтропии) равна количеству полученной в результате опыта информации. Важным при введении какой-либо величины является вопрос о том, что принимать за единицу ее измерения. Очевидно, значение H будет равно 1 при $n=2$. Иначе говоря, в качестве единицы принимается количество информации, связанное с проведением опыта, состоящего в получении одного из двух равновероятных исходов (например, бросание монеты). Такая единица количества информации называется "бит".

Общая характеристика процессов сбора, передачи, обработки и накопления информации

Получение информации тесно связано с информационными процессами. Рассмотрим их виды.

Сбор данных – это деятельность субъекта по накоплению данных с целью обеспечения достаточной полноты. Соединяясь с адекватными методами, данные рождают информацию, способную помочь в принятии решения. Например, интересуясь ценой товара, его потребительскими свойствами, мы собираем информацию для того, чтобы принять решение: покупать или не покупать его.

Передача данных – это процесс обмена данными. Предполагается, что существует источник информации, канал связи, приемник информации, и между ними приняты соглашения о порядке обмена данными, эти соглашения называются протоколами обмена. Например, в обычной беседе между людьми негласно принимается соглашение, не перебивать друг друга во время разговора.

Хранение данных – это поддерживание данных в форме, постоянно готовой к выдаче их потребителю. Одни и те же данные могут быть востребованы не однажды, поэтому разрабатывается способ их хранения (обычно на материальных носителях) и методы доступа к ним по запросу потребителя.

Обработка данных – это процесс преобразования информации от исходной ее формы до определенного результата. Сбор, накопление, хранение информации часто не являются конечной целью информационного процесса. Чаще первичные данные привлекаются для решения какой-либо проблемы, затем они преобразуются шаг за шагом в соответствии с алгоритмом решения задачи до получения выходных данных, которые после анализа пользователем предоставляют необходимую информацию.

4. КОДИРОВАНИЕ СИМВОЛЬНОЙ ИНФОРМАЦИИ

Кодирование информации в ЭВМ - одна из задач теории кодирования. Теория кодирования **-** один из разделов теоретической информатики. Одна из задач - разработка принципов наиболее экономичного кодирования. Эта задача касается передачи, обработки, хранения информации. Частное ее решение – представление информации в компьютере.

Определение. Алфавит, в котором источник информации представляет сообщение, называется *первичным*.

Определение. Алфавит, в котором представлено сообщение после обработки в кодирующем устройстве, называется *вторичным*.

Кодирование – перевод сообщения из первичного алфавита во вторичный. *Декодировани*е **–** операция обратная кодированию.

Операции кодирования и декодирования называются обратимыми**,** если их последовательное применение обеспечит возврат к исходной информации без каких-либо ее потерь.

Утверждение. Пусть первичный алфавит A сост. из N знаков со средней информацией на знак I^A , вторичный B – из M знаков со средней информацией на знак I^B. Пусть сообщение в первичном алфавите содержит n знаков, а закодированное – m знаков. Если исходное сообщение содержит I_S (A) информации, а закодированное I_f (B), то условие обратимости кодирования (то есть неисчезновения информации при кодировании) очевидно может быть записано так:

$$I_S(A) \le I_f(B)$$

или

$$n * I^{(A)} \le m * I^{(B)}$$

То есть операция обратимого кодирования может увеличить количество информации в сообщении, но не может уменьшить ее.

Определение. Среднее число знаков вторичного алфавита, который используется для кодирования одного знака первичного называется *длиной кода*.

$$K(A, B) = m/n$$

Тогда

$$K(A, B) \ge I^{(A)} / I^{(B)}$$

Минимально возможная длина кода:

$$K^{min}(A, B) = I^{(A)} / I^{(B)}.$$

Теорема (*первая теорема Шеннона).* При отсутствии помех всегда возможен такой вариант кодирования сообщения, при котором средняя длина кода будет сколь угодно близкой к

отношению средних информаций на знак первичного и вторичного алфавитов.

При кодировании длина кода может быть одинаковой для всех знаков первичного алфавита (код равномерный) или различной (неравномерный код). Коды могут строиться для отдельного знака первичного алфавита (алфавитное кодирование) или для их комбинаций (кодирование блоков, слов).

Пример. Представление символьной информации в компьютере.

Определим, какой должна быть длина кода. Компьютерный алфавит С включает 52 буквы латинского алфавита, 66 букв русского (прописные и строчные), цифры 0…9, 20 символов для обозначения знаков математических операций, препинания и т.д. В сумме получается 148 символов.

Вычислим длину кода: $K\ (C,\ 2) \geq log_2 148 \geq 7{,}21$, но длина кода – целое число, следовательно, $K\ (C,2) = 8$. Именно такой способ кодирования принят в компьютерных системах. Называют 8 бит =1 байт, а кодирование байтовым. Один байт соответствует количеству информации в одном знаке алфавита при их равновероятном распределении.

Символы в компьютере хранятся в виде числового кода, причем каждому символу ставиться в соответствии своя уникальная комбинация двоичных разрядов. В этом случае текст будет представлен как длинный ряд битов, в котором следующее друг за другом комбинации битов отражают последовательность символов в исходном тексте. Присвоение символу конкретного двоичного кода это вопрос соглашения, которое фиксируется в кодовой таблице – их существует несколько.

Таблица, в которой устанавливается однозначное соответствие между символами и их порядковыми номерами, называется таблицей кодировки.

С распространением ПК типа IBM PC международным стандартом стала таблица кодировки под названием American Standard Cod for Information Interchange – ASCII. Системы кодирования текстовых данных были разработаны и в других странах. Так, например, в СССР в этой области действовала система кодирования КОИ-7 и КОИ-8. Система, основанная на 16-разрядном кодировании символов, получила название универсальной Unicode. Шестнадцать разрядов позволяют обеспечить уникальные коды для 65536 различных символов – этого поля достаточно для размещения в одной таблице символов большинства языков планеты.

5. СИСТЕМЫ СЧИСЛЕНИЯ

Определение. *Системой счисления* называется представление чисел с помощью комбинации знаков. Количество этих знаков образуют основание системы счисления.

Система счисления называется *позиционной*, если значение знака изменяется в зависимости от его места в числе.

Система счисления называется не *позиционной*, если значение знака не зависит от его места в числе.

Например, в десятичной системе счисления в числе 118, значение знака «единица» разное. В римской системе счисления это число представляется в виде: CXVIII, здесь знак «единица» имеет одно и то же значение, поэтому десятичная система счисления является позиционной, а римская система счисления не является позиционной. В дальнейшем будем рассматривать только позиционные системы счисления.

В десятичной системе счисления для изображения числа используются следующие десять знаков: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. Числа большей величины, как, например, триста семьдесят два, представляются в виде:

$$300 + 70 + 2 = 3 \cdot 10^2 + 7 \cdot 10 + 2$$

и в десятичной системе записываются 372. Существенно в данном случае то, что смысл каждой из цифр — 3, 7, 2 — зависит от ее положения — от того, стоит ли она на месте единиц, десятков или сотен. Общее правило такого изображения дается схемой, которая иллюстрируется примером:

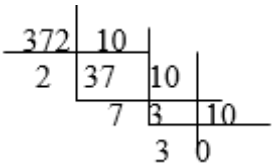
$$z = a \cdot 10^3 + b \cdot 10^2 + c \cdot 10 + d,$$

где, a, b, c, d — целые числа в пределах от нуля до девяти. Число z в этом случае сокращенно обозначается символом

$a\ b\ c\ d$.

Заметим, между прочим, что коэффициенты d, c, b, a являются не чем иным, как остатками при последовательном делении числа z на 10.

Так, например,



С помощью написанного выше выражения для числа z

можно изображать только те числа, которые меньше десяти тысяч, так как числа, большие, чем десять тысяч, требуют пяти или большего числа цифр. Если z есть число, заключенное между десятью тысячами и ста тысячами, то можно представлять его в виде:

$$z = a \cdot 10^4 + b \cdot 10^3 + c \cdot 10^2 + d \cdot 10 + e$$

и тогда записать символически *abcde*. Подобное же утверждение справедливо относительно чисел, заключенных между ста тысячами и одним миллионом и т. д. Чрезвычайно важно располагать способом, позволяющим получить результат посредством одной единственной формулы. Этой цели можно добиться, если обозначить различные коэффициенты *e*, *d*, *c* … одной и той же буквой a с различными значками (индексами) a_0, a_1, a_2,…, а степени числа 10 выразим как 10^n, понимая под n произвольное натуральное число. В таком случае любое целое число z в десятичной системе может быть представлено в виде:

$$z = a_n \cdot 10^n + a_{n-1} \cdot 10^{n-1} + \ldots + a_1 \cdot 10 + a_0$$

и записано посредством символа

$$a_n\ a_{n-1}\ a_{n-2} \ldots a_2\, a_1.$$

Как и в частном, рассмотренном выше, мы обнаруживаем, что a_0, a_1, a_2,…, a_n являются остатками при последовательном делении z на 10.

В десятичной системе число десять играет особую роль как «основание» системы. Тот, кому приходится встречаться лишь с практическими вычислениями, может не отдавать себе отчет в том, что такое выделение числа десять не является существенным и что роль основания способно было бы играть любое целое число, большее единицы. Например, была бы вполне возможна семеричная (септимальная) система с основанием семь. В такой системе целое число представлялось бы в виде:

$$b_n \cdot 7^n + b_{n-1} \cdot 7^{n-1} + \ldots + b_1 \cdot 7 + b_0,$$

где b – коэффициенты, обозначающие числа в пределах от нуля до шести. Тогда такое целое число записывалось бы посредством символа $b_n\ b_{n-1} \ldots b_1\ b_0$.

Так, число «сто девять» в семеричной системе обозначалось бы символом 214, потому что

$109 = 2 \cdot 7^2 + 1 \cdot 7 + 4.$

В качестве упражнения можно вывести общее правило для перехода от основания 10 к любому основанию *p*: нужно выполнять последовательное деление на *p*, начиная с данного числа *z*; остатки и будут «цифрами» при записи числа в системе с основанием *p*. Например,

109	7		
4	15	7	
	3	2	7
	1	2	0

109 (в десятичной системе) = 214 (в семеричной системе).

Позиционные системы счисления

Изображение целых значений в позиционных системах счисления: десятичная, двоичная и шестнадцатеричная системы

Введем для общности понятие об изображении целых числовых значений в системе счисления с основанием *p*, где *p* — целое число, $p \geq 2$. Цифрами в системе счисления с основанием *p* называют *p* символов, обозначающих все целые значения от 0 до $p - 1$.

В десятичной системе счисления ($p = 10$) такими символами являются

0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

В двоичной системе счисления ($p = 2$) в качестве цифр употребляются символы 0 и 1.

В шестнадцатеричной или *hex*-системе счисления в качестве цифр используются символы

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, *A*, *B*, *C*, *D*, *E*, *F*.

Изображение целого числового значения *N* в виде строки

$$c_n c_{n-1} \ldots c_0 \quad (n \geq 0) \qquad (1)$$

в *p*-ичной системе счисления (c_i — цифры этой системы счисления) значение *N* определяют формулой:

$$N = c_n p^n + c_{n-1} p^{n-1} + \ldots + c_0 p^0. \qquad (2)$$

Из формулы (2) следует, в частности, что p^k в *p*-ичной системе счисления изображается последовательностью *k*+1 символов, из которых первый — 1, а остальные *k* — нули, например:

$10^3 = (1000)_{16}$; $16 = (10)_{16}$; $16^2 = (100)_{16}$;

$2 = (10)_2$; $2^4 = (10000)_2$ и т. п.

Индексом справа внизу обозначено основание системы счисления, в которой представлено число в скобках.

С помощью формулы (2), зная изображение (1) в p-ичной системе счисления, можно получить изображение значения N в q-ичной системе счисления. Для этого нужно изобразить c_i и p в q-ичной системе счисления и выполнить действия, предписанные формулой (2) по правилам «q-ичной арифметики».

Если $q = 10$, то эти действия — перевод изображения из p-ичной системы счисления в его изображение в десятичной системе — выполняются с помощью привычной нам десятичной арифметики. Рассмотрим примеры.

Пусть $p = 2$ и задано изображение некоторого значения в двоичной системе счисления :

1111011 (3)

Тогда в соответствии с формулой (2) десятичное изображение этого значения определяется выполнением действий:

$$1 \cdot 2^6 + 1 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 \quad (4)$$

Вычислив выражение (4), получим десятичное изображение значения, представленного в двоичной системе в виде (3): 123, т. е.:

$$(1111011)_2 = (123)_{10}.$$

Пусть теперь $p = 16$ и задано изображение некоторого значения в шестнадцатеричной системе счисления *AF*01.

Для получения десятичного изображения этого значения подставим в формулу (2) десятичные представления шестнадцати шестнадцатеричных цифр и показателей степеней $10 \cdot 16^3 + 15 \cdot 16^2 + 0 \cdot 16^1 + 1 \cdot 16^0 = 44801$,т. е.:

$$(AF01)_{16} = (44801)_{10}.$$

Для перевода десятичного ($p = 10$) изображения значения в его изображение в какой-нибудь «чужой» системе счисления (например, $q = 16$) следовало бы, пользуясь формулой (2), подставлять в нее шестнадцатеричные изображения степеней десяти и проводить умножение и сложение в шестнадцатеричной системе. Однако человеку действовать по правилам шестнадцатеричной системы неудобно. Поэтому для перевода вручную изображе-

ний числовых значений из десятичной системы счисления в шестнадцатеричную используется другой прием.

Пусть известно десятичное изображение *N*. Из формулы (2) следует: значение младшей шестнадцатеричной цифры c_0 является остатком от деления *N* на 16, а частное этого деления $N_1 = c_1 + c_2 16^1 + \ldots c_n 16^{n-1}$. Таким образом, c_1, c_2, c_3 и т. д. будут являться остатками от деления *N*, N_1, N_2, … и т. д. на 16. Деление продолжается до тех пор, пока очередное частное не окажется равным нулю.

Например, получение десятичного значения *N* = 44801 в шестнадцатеричной системе счисления выполняется так:

```
44801|16
остаток:    1|2800| 16
остаток:       0  | 175|16
остаток:          15 | 10 | 16
остаток:             10 | 0
```

Таким образом, значения шестнадцатеричных цифр равны соответственно значениям остатков 1, 0, 15, 10 (в десятичном представлении). Заменяя эти значения их шестнадцатеричным изображением, получим:

$$(44801)_{10} = (AF01)_{16}.$$

Можно доказать следующее простое правило перехода от двоичного изображения числового значения к его шестнадцатеричному изображению и обратно. Для этого достаточно (дополнив, если надо, двоичное изображение незначащими нулями слева) заменить каждую четверку двоичных цифр (тетраду) шестнадцатеричной цифрой, изображающей значение этого четырехразрядного двоичного числа (табл.1).

Таблица 1

Соответствие десятичных цифр (чисел) шестнадцатеричным цифрам и двоичным тетрадам

P = 10	*P* = 16	*P* = 2	*P* = 10	*P* = 16	*P* = 2
0	0	0000	8	8	1000
1	1	0001	9	9	1001
2	2	0010	10	10	1010
3	3	0011	11	11	1011
4	4	0100	12	12	1100

5	5	0101	13	13	1101
6	6	0110	14	14	1110
7	7	0111	15	15	1111

Например:

$$(1101011111101)_2 = (1AFD)_{16}.$$

Для обратного перехода – от шестнадцатеричного изображения значения к двоичному – достаточно заменить каждую шестнадцатеричную цифру соответствующей двоичной тетрадой:

$$(B7C2F)_{16} = (10110111110000101111)_2.$$

Арифметические и логические действия над двоичными и шестнадцатеричными изображениями целых значений выполняются по правилам, полностью совпадающими с теми, которые применяются к десятичным изображениям чисел.

Приведем несколько примеров сложения и вычитания двоичных и шестнадцатеричных чисел:

а) $(78)_{10} = (4E)_{16} = (100\ 1110)_2$

$+ \qquad + \qquad +$

$(47)_{10} = (2F)_{16} = (10\ 1111)_2$

$(125)_{10} = (7D)_{16} = (111\ 1101)_2$

б) $(1450)_{10} = (5AA)_{16} = (101\ 1010\ 1010)_2$

$- \qquad - \qquad -$

$(427)_{10} = (1AB)_{16} = (1\ 1010\ 1011)_2$

$(1023)_{10} = (3FF)_{16} = (11\ 1111\ 1111)_2$

в) $(4095)_{10} = (FFF)_{16} = (1111\ 1111\ 1111)_2$

$+1 \qquad +1 \qquad + \qquad 1$

$(4096)_{10} = (1000)_{16} = (1\ 0000\ 0000\ 0000)_2$

Изображение дробных чисел в двоичной и шестнадцатеричной системах счисления

Изображением дроби в какой-либо системе счисления называется последовательность вида:

$$0.a_1 a_2 \ldots a_m . \qquad (5)$$

Ноль и точка выполняют роль признака правильной дроби; a — цифры в соответствующей системе счисления. Приписывание любого числа незначащих нулей слева от 0 или справа от a_m не меняет изображаемого значения. Значение дроби (5) есть значение выражения:

$$Q = a_1 p^{-1} + a_2 p^{-2} + \ldots + a_m p^{-m}, \qquad (6)$$

где p – целое значение – основание системы счисления.

Для перевода двоичного или шестнадцатеричного изображения дроби в десятичное нужно подставить в выражение (6) десятичное изображение a_i и p и выполнить действия по правилам десятичной арифметики. Например:

$$(0.1001101)_2 =$$

$$1\cdot 2^{-1}+0\cdot 2^{-2}+0\cdot 2^{-3}+1\cdot 2^{-4}+1\cdot 2^{-5}+0\cdot 2^{-6}+1\cdot 2^{-7} =$$

$$(0.6015625)_{10};$$

$$(0.9A)_{16} = 9\cdot 16^{-1} + 10\cdot 16^{-2} = (0.6015625)_{10}.$$

Чтобы получить двоичное или шестнадцатеричное изображение значения дроби Q, заданной десятичным представлением, умножают Q по правилам десятичной арифметики на 2 (или 16); значение целой части произведения a_1 изображают цифрой двоичной (шестнадцатеричной) системы. Если дробная часть Q_1 произведения не ноль, то Q_1 умножают на 2 (16) и тем же путем получают значение цифры a_2 и т. д.

Переведем, например, десятичную дробь 0.6015625 в двоичную и шестнадцатеричную системы счисления:

$0.6015625 \cdot 2 = 1.203125,$ $\quad a_1 = 1;$

$0.203125 \cdot 2 = 0.40625,$ $\quad a_2 = 0;$

$0.40625 \cdot 2 = 0.8125,$ $\quad a_3 = 0;$

$0.8125 \cdot 2 = 1.625,$ $a_4 = 1;$

$0.625 \cdot 2 = 1.25,$ $a_5 = 1;$

$0.25 \cdot 2 = 0.5,$ $a_6 = 0;$

$0.5 \cdot 2 = 1.0,$ $a_7 = 1.$

Искомое двоичное представление 0.1001101.

$0.6015625 \cdot 16 = 9.625,$ $a_1 = (9)_{10} = (9)_{16};$

$0.625 \cdot 16 = 10.00,$ $a_2 = (10)_{10} = (A)_{16}.$

Шестнадцатеричное представление 0.9A.

Между цифрами двоичных и шестнадцатеричных изображений дробей существует такая же простая связь, как между цифрами двоичных и шестнадцатеричных представлений целых значений, в чем можно убедиться на предыдущем примере.

Арифметические действия в двоичной и шестнадцатеричной системах счисления

Исключительная роль десятка восходит к истокам цивилизации и без всякого сомнения связана со счетом по пальцам на двух руках. Но наименование в числительных в разных языках указывает и на наличие – в былые времена – иных систем счисления, а именно: с основаниями двадцать и двенадцать. В английском и немецком языках слова, обозначающие 11 и 12, построены не по десятичному принципу, сочетающему десятки с единицами: они лингвистически независимы от слов, обозначающих число 10. Во французском языке слова, обозначающие 20 или 80, позволяют предполагать о первоначальном существовании системы с основанием 20, используемой для тех или иных надобностей. В датском языке слово halvfirsindstyve, обозначающее 70, буквально переводится «полпути от трижды двадцать до четырежды двадцать». Вавилонские астрономы пользовались системой частично секзагезимальной структуры (с основанием 60), и предполагается, что именно в этом обстоятельстве следует искать объяснение того факта, что час и угловой радиус разделены на 60 минут.

Как правило, люди считают в десятичной системе, где для изображения чисел используются 10 цифр: 0, 1, …, 9. Основой других систем счисления являются иные символы, число которых может быть меньше или больше 10. Рассмотрим две такие системы, в одной из которых (двоичной) для представления чисел используются два символа (две двоичные цифры) 0 и 1, а в

другой (шестнадцатеричной) — 16 символов. Прежде чем приступить к рассмотрению этих систем, опишем правила, по которым происходит процесс счета в десятичной системе.

В десятичной системе при изображении чисел, больших девяти, различные символы (т. е. цифры 0, 1, …, 9) располагаются друг за другом, например 365. Комбинации этих символов могут быть получены сложением 1 и 0 с последующим добавлением 1 к каждой получаемой сумме: 1 + 0 = 1, 1+ 1 = 2, 1 + 2 = 3 и т. д. Поскольку при операции сложения 1 + 9 сумму невозможно изобразить одним символом, поэтому слева от цифры девять, к которой прибавляется 1, ставится 1, а саму цифру девять заменяют цифрой 0, иначе говоря, осуществляют перенос в старший разряд. После этого можно продолжать операцию добавления 1 к сумме. Описанный процесс, называемый счетом, позволяет получить все комбинации цифр, используемых для изображения чисел в десятичной системе.

• Заметим, что при счете в десятичной системе особое внимание следует обращать на выполнение переноса и замену наибольшей цифры наименьшей.

Рассмотрим теперь, как происходит счет в двоичной системе счисления. Напомним, что в этой системе для изображения чисел используются только два символа: 0 и 1. Начнем счет также, как и в десятичной системе, складывая 1 и 0. Естественно, что 1 + 0 = 1. Добавим к полученной сумме 1. Поскольку сумму в двоичной системе невозможно представить одной цифрой, как и раньше, выполним перенос: припишем слева к первой сумме 1, а ее значение заменим на 0.

• Заметим, что операция переноса выполняется также, как при счете в десятичной системе с той лишь разницей, что в двоичной системе используются только два символа (две двоичные цифры).

Процесс счета в десятичной и двоичной системах счисления показан в табл. 2, из которой следует, что десятичное число 7 эквивалентно двоичному числу (0111).

Нетрудно также установить, что в обеих системах процесс счета можно продолжать бесконечно, при этом каждому десятичному числу будет соответствовать некоторое двоичное число.

Таблица 2

Счет в двоичной и десятичной системах счисления

Двоичная система	Десятичная система
0000 + 1 -------- 0001 + 1 -------- 0010 + 1 -------- 0011 + 1 -------- 0100 + 1 -------- 0101 + 1 -------- 0110 + 1 -------- 0111 + 1 -------- 1000 + 1 -------- 1001 + 1 -------- 1010 + 1 -------- 1011 + 1 -------- 1100	0000 + 1 ------ 0001 + 1 ------ 0002 + 1 ------ 0003 + 1 ------ 0004 + 1 ------ 0005 + 1 ------ 0006 + 1 ------ 0007 + 1 ------ 0008 + 1 ------ 0009 + 1 ------ 0010 + 1 ------ 0011 + 1 ------ 0012

Обратимся теперь к шестнадцатеричной системе счисления. В этой системе используются 16 символов: цифры от 0 до 9 и буквы от *A* до *F*. Поэтому шестнадцатеричное число может иметь вид 03*FA*. Чтобы определить шестнадцатеричные числа, можно вновь повторить процесс счета подобному тому, как это делалось в случае десятичной и двоичной систем.

В табл. 3 изображены числа, получаемые при счете в каждой из трех систем счисления. Из нее следует, что числа 15, *F* и 1111 эквивалентны, как эквивалентны числа 12, *C* и 1100.

Заметим, что одной и той же комбинацией цифр можно изобразить числа из различных систем счисления. Например, число 110 может принадлежать двоичной, десятичной и шестнадцатеричной системам. Во избежание путаницы двоичные числа помечают буквой *B* (например, 00101000*B*), шестнадцатеричные – буквой *H* (например, (043*A*) *H*). Иногда в документации или в таблицах появляются числа без буквенных указателей, несмотря на то, что они являются двоичными или шестнадцатеричными. В подобных ситуациях их принадлежность той или иной системе счисления определяется из контекста.

Таблица 3

Система счисления					
Двоичная	Десятичная	Шестнадцатеричная	Двоичная	Десятичная	Шестнадцатеричная
0000	0000	00	1000	0008	08
0001	0001	01	1001	0009	09
0010	0002	02	1010	0010	0A
0011	0003	03	1011	0011	0B
0100	0004	04	1100	0012	0C
0101	0005	05	1101	0013	0D
0110	0006	06	1110	0014	0E
0111	0007	07	1111	0015	0F

6. КРАТКАЯ ИСТОРИЯ ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

В начале 20 века сложилась ситуация, что, сколько бы людей не привлекалось дополнительно в сферу информации, они все равно не могли справиться с все растущими потребностями по подготовке и обработке информации, генерируемой человеческим обществом – возникла потребность в создании электронных машин.

Совокупность устройств, предназначенных для автоматической или автоматизированной обработки информации называют вычислительной техникой. Конкретный набор, связанных между собою устройств, называют вычислительной системой. Центральным устройством большинства вычислительных систем является электронная вычислительная машина или компьютер.

Определение. *Компьютер* – это электронное устройство, которое выполняет операции ввода информации, хранения и обработки ее по определенной программе, вывод полученных результатов в форме, пригодной для восприятия человеком.

Вообще же слово компьютер означает «вычислитель», то есть устройство для вычислений. Как уже было сказано, потребность в автоматизации обработки данных, в том числе вычислений, возникла очень давно. Много тысячи лет назад для счета использовались счетные палочки, камешки и т.д. Более 1500 лет назад для облегчения вычислений стали использовать счеты. В 1642 году Блез Паскаль изобрел устройство, механически выполняющее сложение чисел, а в 1673 году Готфрид Вильгельм Лейбниц сконструировал арифмометр, позволяющий механически выполнять четыре арифметических операции. В 1943 году американец Говард Эйкен на основе уже техники 20 века (электромеханических реле) смог построить на одном из предприятий фирмы IBM такую машину под названием «Марк-1». В 1945 году к работе был привлечен знаменитый математик Джон фон Нейман, который подготовил доклад об этой машине. В нем Нейман ясно и четко сформулировал общие принципы функционирования универсальных вычислительных устройств, т.е. компьютеров.

Первый компьютер, в котором были воплощены принципы фон Неймана, был построен в 1949 году английским исследователем Морисом Уилксом. С той поры компьютеры стали гораздо более мощными, но подавляющее большинство из них сделано в соответствии с теми принципами, которые изложил в своем докладе в 1945 году Джон фон Нейман.

В 1951-1954 гг. элементная базой ЭВМ были электронные лампы. Их быстродействие (кол-во операций в секунду) составляло 10^4. Затем, в 1958 – 1960 гг. элементной базой стали транзисторы (полупроводниковые схемы). Быстродействие повысилось до 10^6. С 1965 г. появились интегральные схемы (позже – сверх большие интегральные схемы). К 1985 г. быстродействие повысилось до 10^9. В настоящее время технологии изготовления компьютерной техники совершенствуются быстрее, чем технологии в какой-либо другой отрасли.

Однако классическая архитектура ЭВМ, предложенная Нейманом, до настоящего времени лежит в основе устройства компьютеров.

Определение. *Архитектура* - это наиболее общие принципы построения ЭВМ, реализующие программное управление работой и взаимодействием основных ее функциональных узлов.

Определение. *Запоминающее устройство* - это блок ЭВМ, предназначенный для временного (оперативная память) и продолжительного (постоянная память) хранения программ, входных и результирующих данных, а также промежуточных результатов.

Информация в оперативной памяти сохраняется временно, лишь при включенном питании, но оперативная память имеет большее быстродействие. В постоянной памяти данные могут сохраняться даже при отключенном компьютере, но скорость обмена данными между постоянной памятью и центральным процессором, в подавляющем большинстве случаев, значительно меньше.

Фон Нейман с соавторами выдвинули основные принципы логического устройства ЭВМ и предложили ее структуру, которая полностью воспроизводилась в течение первых двух поколений ЭВМ:

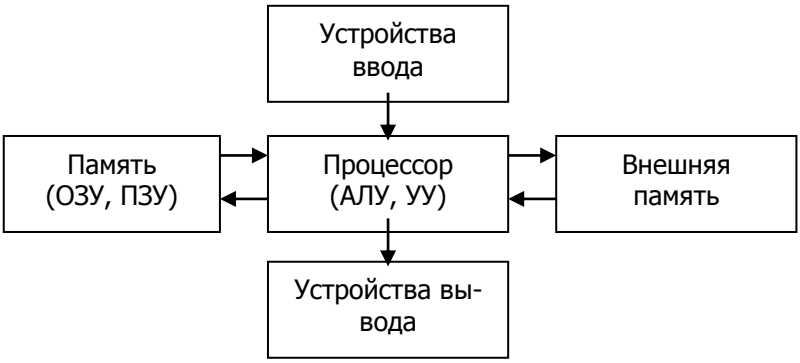


Рис. 2. Классическая архитектура ЭВМ.

Определение. *Арифметико-логическое устройство* - это блок ЭВМ, в котором происходит преобразование данных по командам программы: арифметические действия над числами, преобразование кодов и др.

Управляющее устройство координирует работу всех блоков компьютера. В определенной последовательности он выбирает из оперативной памяти команду за командой. Каждая команда декодируется, по потребности элементы данных из указанных в команде ячеек оперативной памяти передаются в АЛУ; АЛУ настраивается на выполнение действия, указанной текущей командой (в этом действии могут принимать участие также устройства ввода-вывода); дается команда на выполнение этого действия. Этот процесс будет продолжаться до тех пор, пока не возникнет одна из следующих ситуаций: исчерпаны входные данные, от одного из устройств поступила команда на прекращение работы, выключено питание компьютера.

Кроме архитектуры ЭВМ Нейман предложил основополагающие принципы логического устройства ЭВМ.

0. Для представления данных в ЭВМ необходимо использование двоичной системы (преимущественно для технической реализации, простота выполнения арифметических и логических операций).

1. Принцип программного управления. Программа состоит из набора команд, которые выполняются процессором автоматически друг за другом в определенной последовательности. Регистр – специализированная дополнительная ячейка памяти в процессоре. Регистр выполняет функцию кратковременного хранения числа или команды. Счетчик команд – регистр УУ, содержимое которого соответствует адресу очередной выполняемой команды, он служит для автоматической выборки программы из последовательных ячеек памяти. То есть, с его помощью осуществляется выборка программы из памяти. Этот регистр последовательно увеличивает хранимый в нем адрес очередной команды на длину команды. А так как, команды программы расположены в памяти друг за другом, то тем самым осуществляется выборка цепочки команд из последовательно расположенных ячеек памяти. Если же нужно после выполнения команды перейти не к следующей, а к какой – то другой, используются команды условного или безусловного переходов. Таким образом, процессор исполняет программу автоматически, без вмешательства человека.

2. Принцип однородности памяти. Программы и данные

хранятся в одной и той же памяти. Поэтому компьютер не различает, что храниться в данной ячейке памяти – число, текст или команда. Над командами можно выполнять такие же действия, как и над данными.

3. Принцип адресности. Структурно основная память состоит из пронумерованных ячеек; процессору в произвольный момент времени доступна любая ячейка. Это позволяет обращаться к произвольной ячейке (адресу) без просмотра предыдущих ячеек.

7. КЛАССИФИКАЦИЯ И АРХИТЕКТУРА ВЫЧИСЛИТЕЛЬНОЙ СИСТЕМЫ

Классификация ЭВМ по принципу действия

Электронная вычислительная машина, компьютер - комплекс технических средств, предназначенных для автоматической обработки информации в процессе решения вычислительных и информационных задач.

По принципу действия вычислительные машины делятся на три больших класса (рис.2): аналоговые (АВМ), цифровые (ЦВМ) и гибридные (ГВМ).

Рис.2. Классификация вычислительных машин по принципу действия.

Критерием деления вычислительных машин на эти три класса является форма представления информации, с которой они работают (рис.3).

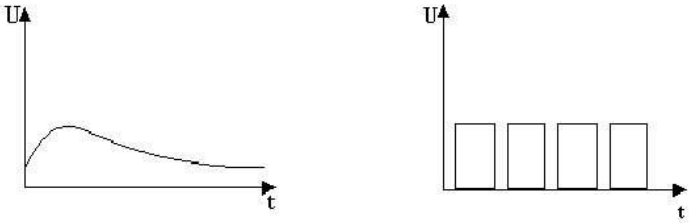


Рис.3. Две формы предоставления информации в машинах: *а*- аналоговая; *б*- цифровая импульсная.

Цифровые вычислительные машины (ЦВМ) - вычислительные машины дискретного действия, работают с информацией, представленной в дискретной, а точнее, в цифровой форме.

Аналоговые вычислительные машины (АВМ) - вычислительные машины непрерывного действия, работают с информацией, представленной в непрерывной (аналоговой) форме, т.е. в виде непрерывного ряда значений какой-либо физической величины (чаще всего электрического напряжения).

Аналоговые вычисли- тельные машины весьма просты

и удобны в эксплуатации; программирование задач для решения на них, как правило, нетрудоемкое; скорость решения задач изменяется по желанию оператора и может быть сделана сколь угодно большой (больше, чем у ЦВМ), но точность решения задач очень низкая (относительная погрешность 2-5 %). На АВМ наиболее эффективно решать математические задачи, содержащие дифференциальные уравнения, не требующие сложной логики.

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

Наиболее широкое применение получили ЦВМ с электрическим представлением дискретной информации - электронные цифровые вычислительные машины**,** обычно называемые просто *электронными вычислительными машинами* **(ЭВМ),** без упоминания об их цифровом характере**.**

Классификация ЭВМ по назначению

По назначению ЭВМ можно разделить на три группы**:** универсальные (общегоназначения), проблемно-ориентированные и специализированные (рис.4).

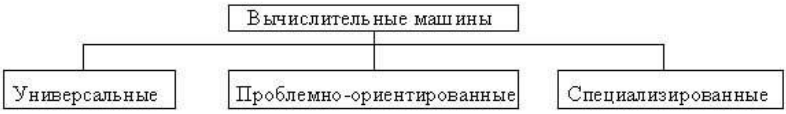


Рис.4. Классификация ЭВМ по назначению.

Универсальные ЭВМ предназначены для решения самых различных инженерно-технических задач: экономических, математических, информационных и других задач, отличающихся сложностью алгоритмов и большим объемом обрабатываемых данных. Они широко используются в вычислительных центрах коллективного пользования и в других мощных вычислительных комплексах.

Характерными чертами универсальных ЭВМ являются:

- высокая производительность;
- разнообразие форм обрабатываемых данных: двоичных, десятичных, символьных, при большом диапазоне их изменения и высокой точности их представления;
- обширная номенклатура выполняемых операций, как арифметических, логических, так и специальных;

- большая емкость оперативной памяти;
- развитая организация системы ввода-вывода информации, обеспечивающая подключение разнообразных видов внешних устройств.

Проблемно-ориентированные ЭВМ служат для решения более узкого круга задач, связанных, как правило, с управлением технологическими объектами; регистрацией, накоплением и обработкой относительно небольших объемов данных; выполнением расчетов по относительно несложным алгоритмам; они обладают ограниченными по сравнению с универсальными ЭВМ аппаратными и программными ресурсами.

К проблемно-ориентированным ЭВМ можно отнести, в частности, всевозможные управляющие вычислительные комплексы.

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

К специализированным ЭВМ можно отнести, например, программируемые микропроцессоры специального назначения; адаптеры и контроллеры, выполняющие логические функции управления отдельными несложными техническими устройствами, агрегатами и процессами; устройства согласования и сопряжения работы узлов вычислительных систем.

Классификация ЭВМ по размерам и функциональным возможностям

По размерам и функциональным возможностям ЭВМ можно разделить (рис.5) на сверхбольшие (суперЭВМ), большие, малые, сверхмалые (микроЭВМ).

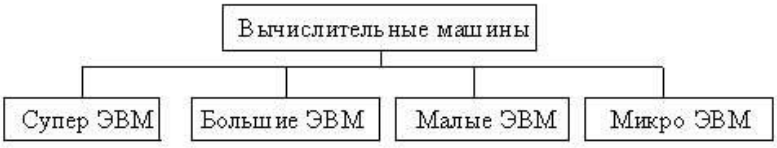


Рис. 5. Классификация ЭВМ по размерам и вычислительной мощности

Функциональные возможности ЭВМ обусловливают важнейшие технико-эксплуатационные характеристики**:**

- быстродействие, измеряемое усредненным количеством

операций, выполняемых машиной за единицу времени;

- разрядность и формы представления чисел, с которыми оперирует ЭВМ;
- номенклатура, емкость и быстродействие всех запоминающих устройств;
- номенклатура и технико-экономические характеристики внешних устройств хранения, обмена и ввода-вывода информации;
- типы и пропускная способность устройств связи и сопряжения узлов ЭВМ между собой (внутримашинного интерфейса);
- способность ЭВМ одновременно работать с несколькими пользователями и выполнять одновременно несколько программ (многопрограммность);
- типы и технико-эксплуатационные характеристики операционных систем, используемых в машине;
- наличие и функциональные возможности программного обеспечения;
- способность выполнять программы, написанные для других типов ЭВМ (программная совместимость с другими типами ЭВМ);
- система и структура машинных команд;
- возможность подключения к каналам связи и к вычислительной сети;
- эксплуатационная надежность ЭВМ;
- коэффициент полезного использования ЭВМ во времени, определяемый соотношением времени полезной работы и времени профилактики.

Архитектура вычислительных систем, назначение основных элементов ПК

Архитектура системы – совокупность свойств системы, существенных для пользования.

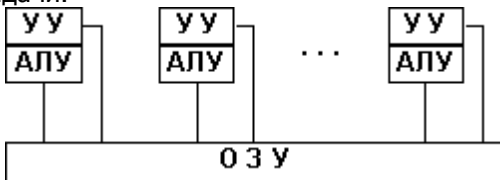
Под *архитектурой ЭВМ* понимают, основные устройства и блоки ЭВМ, а также структуру связей между ними. С точки зрения архитектуры представляют интерес лишь те связи и принципы, которые являются наиболее общими, присущими многим конкретным реализациям вычислительных систем. Именно, то общее, что есть в строении ЭВМ, и относится к понятию архитектуры. С точки зрения архитектуры важны только те сведения о построении ЭВМ, которые могут как-то использоваться при программировании и «пользовательской» работе с ЭВМ.

Классическая архитектура (архитектура фон Неймана) – это *однопроцессорный компьютер*. К этому типу архитектуры относится и архитектура персонального компьютера с *общей шиной*. Все функциональные блоки здесь связаны между собой общей шиной, называемой также *системной магистралью*.

Физически магистраль представляет собой многопроводную линию с гнездами для подключения электронных схем. Совокупность проводов магистрали разделяется на отдельные группы: шину адреса, шину данных и шину управления.

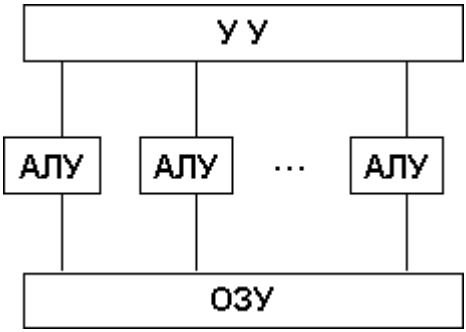
Периферийные устройства (принтер и др.) подключаются к аппаратуре компьютера через специальные *контроллеры — устройства управления периферийными устройствами*.

Многопроцессорная архитектура. Наличие в компьютере нескольких процессоров означает, что *параллельно может быть организовано много потоков данных и много потоков команд*. Таким образом, параллельно могут выполняться несколько фрагментов одной задачи.



Многомашинная вычислительная система. Здесь несколько процессоров, входящих в вычислительную систему, не имеют общей оперативной памяти, а имеют каждый свою (локальную). Каждый компьютер в многомашинной системе имеет классическую архитектуру, и такая система применяется достаточно широко. Однако эффект от применения такой вычислительной системы может быть получен только при решении задач, имеющих очень специальную структуру: она должна разбиваться на столько слабо связанных подзадач, сколько компьютеров в системе. Преимущество в быстродействии многопроцессорных и многомашинных вычислительных систем перед однопроцессорными очевидно.

Архитектура с параллельными процессорами. Здесь несколько АЛУ работают под управлением одного УУ. Это означает, что множество данных может обрабатываться по одной программе — то есть по одному потоку команд. Высокое быстродействие такой архитектуры можно получить только на задачах, в которых одинаковые вычислительные операции выполняются одновременно на различных однотипных наборах данных.



Конфигурацию персонального компьютера можно изменять по мере необходимости. Но, существует понятие базовой конфигурации, которую можно считать типичной:

- системный блок;
- монитор;
- клавиатура;
- мышь.

Системный блок - основная составляющая ПК, в котором находятся его важнейшие компоненты. Устройства, находящиеся в системном блоке называют внутренними, а устройства, подсоединенные извне, называют внешними.

Современный системный блок состоит из нескольких функциональных узлов: процессор, память, контроллеры устройств, блок питания, системная плата, звуковая карта, видеокарта и т.д. Каждый узел представляет собой сложное электронное устройство, в состав которого могут входить миллионы логических элементов.

Основным компонентом ПК является материнская плата с главным процессором и поддерживающими его микросхемами. Материнские платы различают двух видов:

- одноплатовые (плата содержит всю схему компьютера)
- системная плата минимальной конфигурацией (в шиноориентированных компьютерах реализует схему минимальной конфигурации, остальные функции реализуются с добавлением дополнительных плат).

Материнская плата – это комплекс различных устройств, поддерживающий работу системы в целом. Обязательной частью материнской платы являются базовый процессор, оперативная память, системный BIOS, контролер клавиатуры, разъемы расширения.

Процессор - это главная микросхема компьютера, его "мозг". Он разрешает выполнять программный код, находящийся в памяти и руководит работой всех устройств компьютера. Скорость его работы определяет быстродействие компьютера. Сегодня процессоры изготавливаются в виде микропроцессоров. Визуально микропроцессор – это тонкая пластинка кристаллического кремния в форме прямоугольника. Площадь пластины несколько квадратных миллиметров, на ней расположены схемы, которые обеспечивают функциональность процессора ПК. Как правило, пластинка защищена керамическим или пластмассовым плоским корпусом, к которому подсоединена посредством золотых проводков с металлическими наконечниками. Такая конструкция позволяет подсоединить процессор к системной плате компьютера.

Архитектура процессора – это способность процессора выполнять набор машинных кодов. С другой стороны можно сказать, что архитектура процессора – это отражение основных принципов внутренней организации определенных типов процессоров.

Строение процессора

- шины адресов и шины данных;
- арифметико-логическое устройство;
- регистры;
- кэш (быстрая память небольшого объема 8-512 Кбайт);
- счетчики команд; математический сопроцессор.

Центральный процессор (Central Rpocessing Unit - CPU) выполняет все основные операции. Часто ПК оснащен дополнительными сопроцесорами, ориентированными на эффективное выполнение специфических функций. Это, например, математический сопроцесор для обработки числовых данных в формате с плавающей точкой, графический сопроцесор для обработки графических изображений, сопроцесор ввода/вывода для выполнения операции взаимодействия с периферийными устройствами.

Основными параметрами процессоров являются:

- тактовая частота,
- разрядность,
- рабочее напряжение,
- коэффициент внутреннего умножения тактовой частоты,
- размер кеш памяти.

Тактовая частота определяет количество элементарных операций (тактов), выполняемые процессором за единицу времени. Чем больше тактовая частота, тем больше команд может выполнить процессор, и тем больше его производительность.

Разрядность процессора показывает, сколько бит данных он

может принять и обработать в своих регистрах за один такт. Разрядность процессора определяется разрядностью командной шины, то есть количеством проводников в шине, по которой передаются команды.

Рабочее напряжение процессора обеспечивается материнской платой, поэтому разным маркам процессоров отвечают разные материнские платы. Снижение рабочего напряжения разрешает уменьшить размеры процессоров, а также уменьшить тепловыделение в процессоре, что разрешает увеличить его производительность без угрозы перегрева.

Коэффициент внутреннего умножения тактовой частоты - это коэффициент, на который следует умножить тактовую частоту материнской платы, для достижения частоты процессора. Тактовые сигналы процессор получает от материнской платы, которая из чисто физических причин не может работать на таких высоких частотах, как процессор.

Ядро процессора. *Даже процессоры с одинаковой архитектурой могут существенно отличаться друг от друга. Эти различия обусловлены разнообразием процессорных ядер, которые обладают определенным набором характеристик. Наиболее частым отличием является различные частоты системной шины, а также размеры кэша второго уровня и технологическим характеристикам, по которым изготовлены процессоры. Очень часто смена ядра в процессорах из одного и того же семейства, требует также замены процессорного разъема. А это влечет за собой проблемы с совместимостью материнских плат. Но производители постоянно совершенствуют ядра и вносят постоянные, но не значительные изменения в ядре. Такие нововведения называют ревизией ядер и, как правило, обозначаются цифробуквенными комбинациями.*

Системная шина или процессорная шина (FSB – Front Side Bus) – это совокупность сигнальных линий, которые объединены по назначению (адреса, данные и т.д.). Каждая линия имеет определенный протокол передачи информации и электрическую характеристику. То есть системная шина – это связующее звено, которое соединяет сам процессор и все остальные устройства ПК (жесткий диск, видеокарта, память и многое другое). К самой системной шине подключается только CPU, все остальные устройства подключаются через контроллеры, которые находятся в северном мосте набора системной логики (чипсет) материнской платы. Хотя в некоторых процессорах контролер памяти подключен непосредственно в процессор, что обеспечивает более эффективный интерфейс памяти CPU.

Кэш процессора. *Кеш или быстрая память – это обязательная составляющая всех современных процессоров. Кеш является буфером между процессором и контроллером достаточно медленной системной памяти. В буфере хранятся блоки данных, отрабатываемых в данный момент, и процессору не нужно постоянно обращаться к медленной системной памяти. Естественно, это значительно увеличивает общую производительность самого процессора.*

В процессорах, используемых сегодня, кэш поделен на несколько уровней. Самый быстрый – первый уровень L1, который производит работу с ядром процессора. Он обычно разделен на две части – это кэш данных и кэш инструкций. С L1 взаимодействует L2 – кэш второго уровня. Он намного больше по объему и не разделен на кэш инструкций и кэш данных. У некоторых процессоров существует L3 – третий уровень, он еще больше второго уровня, но на порядок медленнее, так как шина между вторым и третьим уровнем уже, чем между первым и вторым. Тем не менее, скорость третьего уровня все равно гораздо выше, нежели скорость системной памяти.

С другими устройствами, и в первую очередь с оперативной памятью, процессор связан группами проводников, которые называются шинами.

Под внутренней памятью понимают все виды запоминающих устройств, расположенные на материнской плате. К ним относятся оперативная память, постоянная память и энергонезависимая память.

Оперативная память RAM *(Random Access Memory) - это массив кристаллических ячеек, способных сохранять данные. Она используется для оперативного обмена информацией (командами и данными) между процессором, внешней памятью и периферийными системами. Из нее процессор берет программы и данные для обработки, в нее записываются полученные результаты. Название "оперативная" происходит от того, что она работает очень быстро и процессору не нужно ждать при считывании данных из памяти или записи. Однако, данные сохраняются лишь временно при включенном компьютере, иначе они исчезают.*

Оперативная память в компьютере размещена на стандартных панельках, которые называются модулями. Модули оперативной памяти вставляют в соответствующие разъемы на материнской плате. Конструктивно модули памяти имеют два выполнения - однорядные (SIMM - модули) и двурядные (DIMM - модули). На компьютерах с процессорами Pentium однорядные модули можно

применять лишь парами (количество разъемов для их установления на материнской плате всегда четное). DIMM - модули можно устанавливать по одному. Комбинировать на одной плате разные модули нельзя.

В момент включения компьютера в его оперативной памяти отсутствуют любые данные, поскольку оперативная память не может сохранять данные при отключенном компьютере. Но процессору необходимы команды, в том числе и сразу после включения. Поэтому процессор обращается по специальному стартовому адресу, который ему всегда известен, за своей первой командой. Этот адрес указывает на память, которую принято называть постоянной памятью ROM или *постоянным запоминающим устройством (ПЗУ).* Микросхема ПЗУ способна продолжительное время сохранять информацию, даже при отключенном компьютере. Говорят, что программы, которые находятся в ПЗУ, "зашиты" в ней - они записываются туда на этапе изготовления микросхемы. Комплект программ, находящийся в ПЗУ образовывает базовую систему ввода/вывода BIOS (Basic Input Output System).

Основное назначение этих программ состоит в том, чтобы проверить состав и трудоспособность системы и обеспечить взаимодействие с клавиатурой, монитором, жесткими и гибкими дисками.

Работа таких стандартных устройств, как клавиатура, может обслуживаться программами BIOS, но такими средствами невозможно обеспечить роботу со всеми возможными устройствами (в связи с их огромным разнообразием и наличием большого количества разных параметров). Но для своей работы программы BIOS требуют всю информацию о текущей конфигурации системы. По очевидной причине эту информацию нельзя сохранять ни в оперативной памяти, ни в постоянной. Специально для этих целей на материнской плате есть микросхема энергонезависимой памяти, которая называется CMOS. От оперативной памяти она отличается тем, что ее содержимое не исчезает при отключении компьютера, а от постоянной памяти она отличается тем, что данные можно заносить туда и изменять самостоятельно, в соответствии с тем, какое оборудование входит в состав системы.

Микросхема памяти CMOS постоянно питается от небольшой батарейки, расположенной на материнской плате. В этой памяти сохраняются данные про гибкие и жесткие диски, процессоры и т.д. Тот факт, что компьютер четко отслеживает дату и время, также связан с тем, что эта информация постоянно хранится (и обновляется) в памяти CMOS. Таким образом, программы BIOS

считывают данные о составе компьютерной системы из микросхемы CMOS, после чего они могут осуществлять обращение к жесткому диску и другим устройствам.

8. ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ КОМПЬЮТЕРОВ

В основу работы компьютеров положен программный принцип управления, состоящий в том, что компьютер выполняет действия по заранее заданной программе.

Определение. *Программа* - это запись алгоритма решения задачи в виде последовательности команд на языке, который понимает компьютер.

Определение. *Программное обеспечение* - совокупность программ, процедур и правил, а также документации, касающихся функционирования системы обработки данных.

Состав программного обеспечения вычислительной системы называется программной конфигурацией. Между программами существует взаимосвязь, то есть работа множества программ базируется на программах низшего уровня.

Междупрограммный *интерфейс* - это распределение программного обеспечения на несколько связанных между собою уровней: прикладной, служебный, системный и базовый.

Базовый уровень является низшим уровнем программного обеспечения. Отвечает за взаимодействие с базовыми аппаратными средствами. Базовое программное обеспечение содержится в специальных микросхемах постоянного запоминающего устройства (ПЗУ), образуя базовую систему ввода-вывода BIOS. Программы и данные записываются в ПЗУ на этапе производства и не могут быть изменены во время эксплуатации.

Системный уровень ПО — переходный. Программы, работающие на этом уровне, обеспечивают взаимодействие прочих программ компьютерной системы с программами базового уровня и непосредственно с аппаратным обеспечением, то есть выполняют "посреднические" функции.

От программного обеспечения этого уровня во многом зависят эксплуатационные показатели всей вычислительной системы в целом. Так, например, при подключении к вычислительной системе нового оборудования на системном уровне должна быть установлена программа, обеспечивающая для других программ взаимосвязь с этим оборудованием. Конкретные программы, отвечающие за взаимодействие с конкретными устройствами, называются *драйверами устройств* — они входят в состав программного обеспечения системного уровня.

Другой класс программ системного уровня отвечает за взаимодействие с пользователем. Именно благодаря им он получает возможность вводить данные в вычислительную систему, управ-

лять ее работой и получать результат в удобной для себя форме. Эти программные средства называют *средствами обеспечения пользовательского интерфейса.* От них напрямую зависит удобство работы с компьютером и производительность труда на рабочем месте.

Совокупность программного обеспечения системного уровня образует *ядро операционной системы компьютера.* Если компьютер оснащен программным обеспечением системного уровня, то он уже подготовлен к установке программ более высоких уровней, к взаимодействию программных средств с оборудованием и, самое главное, к взаимодействию с пользователем. То есть *наличие ядра операционной системы* — непременное условие для возможности практической работы человека с вычислительной системой.

Служебное программное обеспечение

Общая характеристика программ служебного уровня.

Программное обеспечение этого уровня взаимодействует как с программами базового уровня, так и с программами системного уровня. Основное назначение служебных программ (их также называют *утилитами)* состоит в автоматизации работ по проверке, наладке и настройке компьютерной системы. Во многих случаях они используются для расширения или улучшения функций системных программ. Некоторые служебные программы (как правило, это программы обслуживания) изначально включаются в состав операционной системы, но большинство служебных программ являются для операционной системы внешними и служат для расширения ее функций.

В разработке и эксплуатации служебных программ существует два альтернативных направления: *интеграция с операционной системой* и *автономное функционирование.* В первом случае служебные программы могут изменять потребительские свойства системных программ, делая их более удобными для практической работы. Во втором случае они слабо связаны с системным программным обеспечением, но предоставляют пользователю больше возможностей для персональной настройки их взаимодействия с аппаратным и программным обеспечением.

Классификация служебного программного обеспечения

Диспетчеры файлов (файловые менеджеры). С помощью программ данного класса выполняется большинство операций,

связанных с обслуживанием файловой структуры: копирование, перемещение и переименование файлов, создание каталогов (папок), удаление файлов и каталогов, поиск файлов и навигация в файловой структуре. Базовые программные средства, предназначенные для этой цели, обычно входят в состав программ системного уровня и устанавливаются вместе с операционной системой (Например, программа «Проводник»). Для повышения удобства работы с компьютером большинство пользователей устанавливают дополнительные служебные программы.

Средства сжатия данных (архиваторы)- предназначены для создания архивов. Архивирование данных упрощает их хранение за счет того, что большие группы файлов и каталогов сводятся в один архивный файл. При этом повышается и эффективность использования носителя за счет того, что архивные файлы обычно имеют повышенную плотность записи информации. Архиваторы часто используют для создания резервных копий ценных данных.

Средства просмотра и воспроизведения. Обычно для работы с файлами данных необходимо загрузить их в "родительскую" прикладную систему, с помощью которой они были созданы. Это дает возможность просматривать документы и вносить в них изменения. Но в тех случаях, когда требуется только просмотр без редактирования, удобно использовать более простые и более универсальные средства, позволяющие просматривать документы разных типов.

В тех случаях, когда речь идет о звукозаписи или видеозаписи, вместо термина *просмотр* применяют термин воспроизведение документов.

Средства диагностики. Предназначены для автоматизации процессов диагностики программного и аппаратного обеспечения. Они выполняют необходимые проверки и выдают собранную информацию в удобном и наглядном виде. Их используют не только для устранения неполадок, но и для оптимизации работы компьютерной системы.

Средства контроля (мониторинги). Программные средства контроля иногда называют мониторинг. Они позволяют следить за процессами, происходящими в компьютерной системе. При этом возможны два подхода: наблюдение в реальном режиме времени или контроль с записью результатов в специальном протокольном файле. Первый подход обычно используют при изыскании путей для оптимизации работы вычислительной системы и повышения ее эффективности. Второй подход используют в тех случаях, когда мониторинг выполняется автоматически и (или)

дистанционно. В последнем случае результаты мониторинга можно передать удаленной службе технической поддержки для установления причин конфликтов в работе программного и аппаратного обеспечения.

Средства мониторинга, работающие в режиме реального времени, особенно полезны для практического изучения приемов работы с компьютером, поскольку позволяют наглядно отображать те процессы, которые обычно скрыты от глаз пользователя.

Мониторы установки. Программы этой категории предназначены для контроля за установкой программного обеспечения. Необходимость в данном программном обеспечении связана с тем, что между различными категориями программного обеспечения могут устанавливаться связи. Вертикальные связи (между уровнями) являются необходимым условием функционирования всех компьютеров. Горизонтальные связи (внутри уровней) характерны для компьютеров, работающих с операционными системами, поддерживающими принцип совместного использования одних и тех же ресурсов разными программными средствами. И в тех и в других случаях при установке или удалении программного обеспечения могут происходить нарушения работоспособности прочих программ.

Мониторы установки следят за состоянием и изменением окружающей программной среды, отслеживают и протоколируют образование новых связей и позволяют восстанавливать связи, утраченные в результате удаления ранее установленных программ.

Простейшие средства управления установкой и удалением программ обычно входят в состав операционной системы и размещаются на системном уровне программного обеспечения, однако они редко бывают достаточны. Поэтому в вычислительных системах, требующих повышенной надежности, используют дополнительные служебные программы.

Средства коммуникации (коммуникационные программы). С появлением электронной связи и компьютерных сетей программы этого класса приобрели очень большое значение. Они позволяют устанавливать соединения с удаленными компьютерами, обслуживают передачу сообщений электронной почты, работу с телеконференциями (группами новостей), обеспечивают пересылку факсимильных сообщений и выполняют множество других операций в компьютерных сетях.

Средства обеспечения компьютерной безопасности. К этой весьма широкой категории относятся средства пассивной и актив-

ной защиты данных от повреждения, а также средства защиты от несанкционированного доступа, просмотра и изменения данных.

В качестве *средств пассивной защиты,* используют служебные программы, предназначенные для резервного копирования. Нередко они обладают и базовыми свойствами диспетчеров архивов (архиваторов). В качестве *средств активной защиты* применяют *антивирусное программное обеспечение.* Для защиты данных от несанкционированного доступа, их просмотра и изменения служат специальные системы, основанные на криптографии.

Программное обеспечение **прикладного** уровня представляет собой комплекс прикладных программ, с помощью которых выполняются конкретные задачи (производственных, творческих, развлекательных и учебных). Между прикладным и системным программным обеспечением существует тесная взаимосвязь.

Классификация прикладного программного обеспечения:

1. *Текстовые редакторы.* Основные функции - это ввод и редактирование текстовых данных. Для операций ввода, вывода и хранения данных текстовые редакторы используют системное программное обеспечение. С этого класса прикладных программ начинают знакомство с программным обеспечением и на нем приобретают первые привычки работы с компьютером.

2. *Графические редакторы.* Широкий класс программ, предназначенных для создания и обработки графических изображений. Различают три категории:

- растровые редакторы;
- векторные редакторы;
- 3-D редакторы (трехмерная графика).

В *растровых редакторах* графический объект представлен в виде комбинации точек (растров), которые имеют свою яркость и цвет. Такой подход эффективный, когда графическое изображение имеет много цветов и информация про цвет элементов намного важнее, чем информация про их форму. Это характерно для фотографических и полиграфических изображений. Применяют для обработки изображений, создания фотоэффектов и художественных композиций.

Векторные редакторы отличаются способом представления данных изображения. Объектом является не точка, а линия. Каждая линия рассматривается, как математическая кривая III порядка и представлена формулой. Такое представление компактнее, чем растровое, данные занимают меньше места, но построение объекта сопровождается пересчетом параметров кривой в коор-

динаты экранного изображения, и соответственно, требует более мощных вычислительных систем. Широко применяются в рекламе, оформлении обложек полиграфических изданий.

Редакторы трехмерной графики используют для создания объемных композиций. Имеют две особенности: разрешают руководить свойствами поверхности в зависимости от свойств освещения, а также разрешают создавать объемную анимацию.

3. *Системы управления базами данных (СУБД).* Базой данных называют большие массивы данных, организованные в табличные структуры. Основные функции СУБД:

- создание пустой структуры базы данных;
- наличие средств ее заполнения или импорта данных из таблиц другой базы;
- возможность доступа к данных, наличие средств поиска и фильтраци.

В связи с распространением сетевых технологий, от современных СУБД требуется возможность работы с отдаленными и распределенными ресурсами, которые находятся на серверах Интернета.

4. *Электронные таблицы.* Предоставляют комплексные средства для хранения разных типов данных и их обработки. Основной акцент смещен на преобразование данных, предоставлен широкий спектр методов для работы с числовыми данными. Основная особенность электронных таблиц состоит в автоматическом изменении содержимого всех ячеек при изменении отношений, заданных математическими или логическими формулами.

Широкое применение находят в бухгалтерском учете, анализе финансовых и торговых рынков, средствах обработки результатов экспериментов, то есть в автоматизации регулярно повторяемых вычислений больших объемов числовых данных.

5. *Системы автоматизированного проектирования (CAD-системы).* Предназначены для автоматизации проектно-конструкторских работ. Применяются в машиностроении, приборостроении, архитектуре. Кроме графических работ, разрешают проводить простые расчеты и выбор готовых конструктивных элементов из существующей базы данных.

Особенность CAD-систем состоит в автоматическом обеспечении на всех этапах проектирования технических условий, норм и правил. САПР являются необходимым компонентом для гибких производственных систем (ГВС) и автоматизированных систем управления технологическими процессами (АСУ ТП).

6. *Настольные издательские системы.* Автоматизируют про-

цесс верстки полиграфических изданий. Издательские системы отличаются расширенными средствами управления взаимодействия текста с параметрами страницы и графическими объектами, но имеют более слабые возможности по автоматизации ввода и редактирования текста. Их целесообразно применять к документам, которые предварительно обработаны в текстовых процессорах и графических редакторах.

7. *Web-редакторы.* Особый класс редакторов, объединяющих в себе возможности текстовых и графических редакторов. Предназначены для создания и редактирования Web-страниц Интернета. Программы этого класса можно использовать при подготовке электронных документов и мультимедийных изданий.

8. *Браузеры* (средства просмотра Web-документов). Программные средства предназначены для просмотра электронных документов, созданных в формате HTML. Воспроизводят, кроме текста и графики, музыку, человеческий язык, радиопередачи, видеоконференции и разрешают работать с электронной почтой.

9. *Системы автоматизированного перевода.* Различают электронные словари и программы перевода языка.

Электронные словари - это средства для перевода отдельных слов в документе. Используются профессиональными переводчиками, которые самостоятельно переводят текст.

Программы автоматического перевода используют текст на одном языке и выдают текст на другом, то есть автоматизируют перевод. При автоматизированном переводе невозможно получить качественный исходный текст, поскольку все сводится к переводу отдельных лексических единиц.

10. *Интегрированные системы делопроизводства.* Средства для автоматизации рабочего места руководителя. В частности, это функции создания, редактирования и форматирования документов, централизация функций электронной почты, факсимильной и телефонной связи, диспетчеризация и мониторинг документооборота предприятия, координация работы подразделов, оптимизация административно-хозяйственной деятельности и поставка оперативной и справочной информации.

11. *Бухгалтерские системы.* Имеют функции текстовых, табличных редакторов и СУБД. Предназначены для автоматизации подготовки начальных бухгалтерских документов предприятия и их учета, регулярных отчетов по итогам производственной, хозяйственной и финансовой деятельности в форме, приемлемой для налоговых органов, внебюджетных фондов и органов статистического учета.

12. *Финансовые аналитические системы.* Используют в банковских и биржевых структурах. Разрешают контролировать и прогнозировать ситуацию на финансовых, торговых рынках и рынках сырья, выполнять анализ текущих событий, готовить отчеты.

13. *Экспертные системы.* Предназначены для анализа данных, содержащихся в базах знаний и выдачи результатов, при запросе пользователя. Такие системы используются, когда для принятия решения нужны широкие специальные знания. Используются в медицине, фармакологии, химии, юриспруденции. С использованием экспертных систем связана область науки, которая носит название инженерии знаний.

Инженеры знаний - это специалисты, являющиеся промежуточным звеном между разработчиками экспертных систем (программистами) и ведущими специалистами в конкретных областях науки и техники (экспертами).

14. *Геоинформационные системы (ГИС).* Предназначены для автоматизации картографических и геодезических работ на основе информации, полученной топографическим или аэрографическими методами.

15. *Системы видеомонтажа.* Предназначены для цифровой обработки видеоматериалов, монтажа, создания видеоэффектов, исправления дефектов, добавления звука, титров и субтитров. Отдельные категории представляют учебные, справочные и развлекательные системы и программы. Характерной особенностью являются повышенные требования к мультимедийной составляющей.

16. *Инструментальные языки и системы программирования.* Эти средства служат для разработки новых программ. Компьютер "понимает" и может выполнять программы в машинном коде. Каждая команда при этом имеет вид последовательности нулей и единиц. Писать программы на машинном языке крайне неудобно. Поэтому программы разрабатываются на языке, понятном человеку (инструментальный язык или алгоритмический язык программирования), после чего, специальной программой, которая называется транслятором, текст программы переводится (транслируется) на машинный код.

Инструментальные языки делятся на языки низкого уровня (близкие к машинному языку) и языки высокого уровня (близкие к человеческим языкам). К языкам низкого уровня принадлежат ассемблеры, а высокого - Pascal, Basic, C/C++, языки баз данных и т.д. В систему программирования, кроме транслятора, входит

текстовый редактор, компоновщик, библиотека стандартных программ, отладчик, визуальные средства автоматизации программирования. Примерами таких систем являются Delphi, Visual Basic, Visual C++, Visual FoxPro.

9. ОПЕРАЦИОННАЯ И ФАЙЛОВАЯ СИСТЕМЫ КОМПЬЮТЕРА

Операционная система предназначена для управления выполнением пользовательских программ, планирования и управления вычислительными ресурсами ЭВМ.

Операционные системы для персональных компьютеров делятся на:

• одно- и многозадачные (в зависимости от числа параллельно выполняемых прикладных процессов);

• одно- и многопользовательские (в зависимости от числа пользователей, одновременно работающих с операционной системой);

• непереносимые и переносимые на другие типы компьютеров;

• несетевые и сетевые, обеспечивающие работу в локальной вычислительной сети ЭВМ.

Как известно, компьютер выполняет действия в соответствии с предписаниями программы, созданной на одном из языков программирования. При работе пользователя на компьютере часто возникает необходимость выполнить операции с прикладной программой в целом, организовать работу внешних устройств, проверить работу различных блоков, скопировать информацию и т.п.

По существу, эти операции используются для работы с любой программой, воспринимаемой как единое целое. Поэтому целесообразно из всего многообразия операций, выполняемых компьютером, выделить типовые и реализовать их с помощью специализированных программ, которые следует принять в качестве стандартных средств, поставляемых вместе с аппаратной частью.

Программы, организующие работу устройств и не связанные со спецификой решаемой задачи, вошли в состав комплекса программ, названного операционной системой. Функции операционной системы многообразны, постоянно расширяются за счет введения дополнительных программ и модификации старых.

Определение. *Операционная система* — совокупность программных средств, обеспечивающая управление аппаратной частью компьютера и прикладными программами, а также их взаимодействие между собой и пользователем.

Операционная система образует автономную среду, не связанную ни с одним из языков программирования. Любая же прикладная программа связана с операционной системой и может

эксплуатироваться только на тех компьютерах, где имеется аналогичная системная среда. Прикладные программные средства, разработанные в среде одной операционной системы, не могут быть использованы для работы в среде другой операционной системы, если нет специального комплекса программ (конвертера), позволяющего это сделать. В таком случае говорят о программной несовместимости компьютеров.

Программа, созданная в среде одной операционной системы, не функционирует в среде другой операционной системы, если в ней не обеспечена возможность конвертации (преобразования) программ.

Для работы с операционной системой необходимо овладеть языком этой среды — совокупностью команд, структура которых определяется синтаксисом этого языка.

Операционная система выполняет следующие функции**:**

• управление работой каждого блока персонального компьютера и их взаимодействием;

• управление выполнением программ;

• организацию хранения информации во внешней памяти;

• взаимодействие пользователя с компьютером, т.е. поддержку интерфейса пользователя.

Обычно операционная система хранится на жестком диске, а при его отсутствии выделяется специальный гибкий диск, который называется системным диском. При включении компьютера операционная система автоматически загружается с диска в оперативную память и занимает в ней определенное место. Операционная система создается не для отдельной модели компьютера, а для серии компьютеров, в структуре которых заложена и развивается во всех последующих моделях определенная концепция.

В основе любой операционной системы лежит принцип организации работы внешнего устройства хранения информации. Несмотря на то, что внешняя память может быть технически реализована на разных материальных носителях (например, в виде гибкого магнитного диска или магнитной ленты), их объединяет принятый в операционной системе принцип организации хранения логически связанных наборов информации в виде так называемых файлов.

Определение. *Файл* — логически связанная совокупность данных или программ, для размещения которой во внешней памяти выделяется именованная область.

Файл служит учетной единицей информации в операционной системе. Любые действия с информацией в операционной

системе осуществляются над файлами: запись на диск, вывод на экран, ввод с клавиатуры, печать, считывание информации CD-ROM и пр.

В файлах могут храниться разнообразные виды и формы представления информации: тексты, рисунки, чертежи, числа, программы, таблицы и т.п. Особенности конкретных файлов определяются их форматом. Под форматом понимается элемент языка, в символическом виде описывающий представление информации в файле.

Текстовая информация хранится в файле в кодах ASCII, в так называемом текстовом формате. Содержимое текстовых файлов можно просмотреть на экране дисплея с помощью разных программных средств.

Для характеристики файла используются следующие параметры:

- полное имя файла;
- объем файла в байтах;
- дата создания файла;
- время создания файла;
- тип файла;
- специальные атрибуты файла: R (Read only) — только для чтения, H (Hidden) — скрытый файл, S (System) — системный файл, A (Archive) — архивированный файл.

С понятием файла в операционной системе тесно связано понятие логического диска - Логичес*кий* диск создается и управляется специальной программой (драйвером). Он имеет уникальное имя в виде одной латинской буквы, например C, D, E, F и т.д. Логический диск может реализовываться на жестком диске, на гибком диске, на CD-ROM, в оперативной памяти (электронный диск) и т.п. На одном физическом диске может быть создано несколько логических дисков.

К файлу можно обращаться с помощью имени, полного имени, спецификации. Для того чтобы воспользоваться одним из этих вариантов, надо знать ряд правил и соглашений, позволяющих унифицировать в операционной системе процедуру обращения к файлу. Рассмотрим эти варианты.

Имя файла всегда уникально и служит для отличия одного файла от другого. Имя файла образуется из символов, цифр, знаков подчеркивания, но в MS DOS используются только до 8 букв буквы латинского алфавита. По имени к файлу обращаются редко, обычно только в тех прикладных программах, когда это специально предусмотрено, а также при вводе имени файла, где

хранится команда операционной системы.

Обычно к файлу обращаются с помощью полного имени, за исключением оговоренных выше случаев. *Полное имя* файла более подробно характеризует файл и образуется из имени файла и типа (расширения), разделенных точкой.

Тип файла служит для характеристики хранящейся в файле информации и образуется не более чем из трех символов, причем используются, как и при образовании имени, только буквы латинского алфавита.

Понятие каталог и понятие папка используются в операционных системах в качестве объектов, предназначенных для хранения файлов и обеспечения доступа к ним. В дальнейшем будем для однозначности использовать термин каталог.

Имя логического диска, стоящее перед именем файла в спецификации, указывает логический диск, на котором следует искать файл. На этом же диске организован каталог, в котором хранятся полные имена файлов, а также их характеристики: дата и время создания; объем (в байтах); специальные атрибуты. По аналогии с библиотечной системой организации каталогов полное имя файла, зарегистрированное в каталоге, будет служить шифром, по которому операционная система находит месторасположение файла на диске.

Каталог — это справочник файлов с указанием месторасположения на диске. В операционной системе WINDOWS понятию каталог соответствует понятие папка. Различают два состояния каталога — текущее (активное) и пассивное.

Текущий (активный) каталог — каталог, в котором работа пользователя производится в текущее машинное время.

Пассивный каталог — каталог, с которым в данный момент времени не имеется связи.

В операционной системе принята *иерархическая структура организации каталогов.* На каждом диске всегда имеется единственный главный (корневой) каталог. Он находится на 0-м уровне иерархической структуры и обозначается символом "\". Корневой каталог создается при форматировании (инициализации, разметке) диска, имеет ограниченный размер. В главный каталог могут входить другие каталоги и файлы, которые создаются командами операционной системы и могут быть удалены соответствующими командами.

Определение. *Родительский каталог* — каталог, имеющий подкаталоги**.** Подкаталог — каталог, который входит в другой каталог.

Таким образом, любой каталог, содержащий каталоги нижнего уровня, может быть, с одной стороны, по отношению к ним родительским, а с другой стороны, подчиненным по отношению к каталогу верхнего уровня. Как правило, если это не вызывает путаницы, употребляют термин "каталог", подразумевая или подкаталог, или родительский каталог в зависимости от контекста.

Доступ к содержимому файла организован из главного каталога, через цепочку соподчиненных каталогов (подкаталогов) n-го уровня. В каталоге любого уровня могут храниться записи как о файлах, так и о каталогах нижнего уровня.

Описанный выше принцип организации доступа к файлу через каталог является основой файловой системы.

Файловая система — часть операционной системы, управляющая размещением и доступом к файлам и каталогам на диске

С понятием файловой системы тесно связано понятие *файловой структуры* диска, под которой понимают, как размещаются на диске: главный каталог, подкаталоги, файлы, операционная система, а также какие для них выделены объемы секторов, кластеров, дорожек.

10. ЛОГИЧЕСКИЕ ОСНОВЫ АЛГОРИТМИЗАЦИИ

Алгебра логики, основы которой заложены Джорджем Булем, используется при построении основных узлов ЭВМ, например, таких как шифратор, сумматор и так далее.

Алгебра логики оперирует с высказываниями, то есть повествовательными предложениями, о которых можно сказать истинно оно или ложно. Высказывания обозначают большими латинскими буквами и пишут A= 1(t,true, правда), B= 0 (f, false, ложь).

Над высказываниями можно производить определенные логические операции, в результате которых получаются новые высказывания. Их истинность зависит от истинности исходных выражений и вида логической операции. Наиболее часто используются логические операции, выражаемые словами «и», «или», «не».

Определение. Соединение двух (или несколько) высказываний в одно с помощью союза И (AND) называется конъюнкцией (или операцией логического умножения). Обозначаются Л, &, х. Значения логических операций определяются по правилам, задаваемым в таблице истинности. Истинность конъюнкции задается следующей таблицей:

A	B	A & B
0	0	0
0	1	0
1	0	0
1	1	1

Определение. Соединение двух (или несколько) высказываний в одно с помощью союза ИЛИ (OR) называется дизъюнкцией (или логического сложения). Обозначаются I, V, +. Таблица истинности:

A	B	A V B
0	0	0
0	1	1
1	0	1
1	1	1

Определение. Присоединение частицы НЕ (NOT) к данному высказыванию называется операцией отрицания (инверсии). $\bar{A}$, ¬ A – «не A». Таблица истинности:

A	¬ A
1	0
0	1

Операция эквивалентности обозначается A ~ B (A ≡ B, A eqv B) и задается следующей таблицей истинности:

A	B	A ~ B
0	0	1
0	1	0
1	0	0
1	1	1

Определение. Операция импликации (логического следования) объединяет высказывания словами «если…, то …».

A	B	A→ B
0	0	1
0	1	1
1	0	0
1	1	1

Пример. Пусть имеется два высказывания: «данный четырёхугольник – квадрат» (А) и «около данного четырёхугольника можно описать окружность» (В).

Рассмотрим составное высказывание A→ B. Есть три варианта, когда высказывание A→ B истинно:

1. A истинно и B истинно, то есть данный четырёхугольник квадрат, и около него можно описать окружность;

2. A ложно и B истинно, то есть данный четырёхугольник не является квадратом, но около него можно описать окружность (разумеется, это справедливо не для всякого четырёхугольника);

3. A ложно и B ложно, то есть данный четырёхугольник не является квадратом, и около него нельзя описать окружность.

Ложен только один вариант: A истинно и B ложно, то есть данный четырёхугольник является квадратом, но около него нельзя описать окружность.

Определение. Высказывания, образованные с помощью логических операций называются сложными. Истинность их устанавливают, используя таблицы истинности соответствующих операций.

Определение. Высказывания, у которых таблицы истинности совпадают, называются равносильными. Для обозначе-

ния используют знак = (A=B).

Пример. Рассмотрим сложное высказывание (A& B) V (¬A& ¬B) и составим таблицу истинности.

A	B	$\bar{A}$	B	A& B	¬A&¬B	(A&B)V(¬A&¬B)
0	0	1	1	0	1	1
0	1	1	0	0	0	0
1	0	0	1	0	0	0
1	1	0	0	1	0	1

если сравнить с таблицей истинности, для эквивалентности, то видно:

(A& B) V (¬A&¬B) = ¬A ~ ¬B

Следовательно, можно в алгебре логики проводить тождественные преобразования, заменяя высказывания равносильными, а это упрощение выражения.

Следствием самих определений логической операции является ряд свойств:

1. коммутативность (перестановочность)

A& B = B& A

A V B = B V A

2. идемпотентность

A& A= A, A V A= A

3. ассоциативность

AV (BV C)= (AV B) V C= A V B V C

A & (B& C)= (A& B) & C= A& B & C

4. дистрибутивность

A & (B V C)= (A& B) V (A& C)

A V (B& C)= (A V B) & (A V C)

5. законы де Моргана

¬ (A&B)= ¬ A V ¬ B

¬ (A V B)= ¬ A & ¬ B

6. закон универсального множества

X V 1= 1

X & 1= X

7. закон нулевого множества

X V O = X

X& O = O

Любую цифровую систему можно описать при помощи набора булевых функций. Средством обработки двоичных сигналов в ЭВМ являются логические элементы. На практике ИСТИНА

=1 - это наличие напряжения, ЛОЖЬ= 0 - отсутствие.

Определение. Логические элементы - это электронные микросхемы с одним или несколькими входами и одним выходом, через которые проходят электрические сигналы, представляющие 0,1.

Для реализации любой логической операции над двоичными сигналами достаточно элементов трех типов: И, ИЛИ, НЕ. Существуют микросхемы, реализующие более сложные логические функции: И-НЕ, называемая операцией Шеффера, и ИЛИ-НЕ, называемая Стрелка Пирса. Из логических элементов путем их комбинации стоятся основные схемы компьютера. Любую достаточно сложную логическую функцию можно реализовать, имея относительно простой набор базовых логических операций.

Первоначально были разработаны и выпускались микросхемы, соответствующие основным логическим действиям. Довольно быстро стало ясно, что это не может удовлетворить практическим потребностям. Появились более сложные типовые узлы (триггеры, регистры, сумматоры и т.п.), дающие возможность реализовывать еще более сложные логические устройства.

Определение. Триггер - электронный прибор, имеющий два устойчивых состояния, является типичным запоминающим элементом, способным хранить 1 бит информации.

Определение. Регистр - совокупность триггеров, предназначенных для хранения числа в двоичном коде.

Определение. Сумматор - устройство, обеспечивающее суммирование двоичных чисел с учетом переноса из предыдущего разряда.

11. ЭЛЕМЕНТЫ ТЕОРИИ АЛГОРИТМОВ

Выделяют следующие этапы решения задачи на ЭВМ:
1) постановка задачи;
2) математическое описание задачи;
3) разработка алгоритма решения;
4) написание программы на языке программирования;
5) подготовка исходных данных;
6) ввод программы и данных в ЭВМ;
7) отладка и тестирование программы;
8) решение задачи;
9) обработка и интерпретация результатов.

Определение. Алгоритм – это последовательность арифметических и логических действий, приводящая к получению результата и решению задачи (любая конечная система правил преобразования информации).

Свойства алгоритма:

1) детерминированность (применение алгоритма к одним и тем же данным должно приводить к одному результату);

2) массовость (алгоритм дает результат при различных исходных данных);

3) результативность (алгоритм дает результат через конечное число шагов);

4) дискретность (алгоритм должен представлять процесс решения задачи как последовательное выполнение простых шагов; каждое действие, предусмотренное алгоритмом, исполняется только после того, как закончилось исполнение предыдущего).

5) определенность (каждое правило алгоритма должно быть четким, однозначным и не оставлять места для произвола). Благодаря этому свойству выполнение алгоритма носит механический характер и не требует никаких дополнительных указаний или сведений о решаемой задаче.

Структура алгоритмов может быть:

1) *линейной:* вычислительные действия выполняются последовательно друг за другом, алгоритм не содержит условий;

2) *разветвляющейся:* в зависимости от выполнения некоторого условия вычислительный процесс осуществляется по одной или по другой ветви;

3) *циклической:* содержать многократно повторяющиеся участки вычислительного процесса.

Определение. Функциональный блок - это часть алгоритма, имеющая один вход (выполнение начинается с одного опера-

тора) и один выход (после завершения начинает выполняться один и тот же оператор).

Определение. Структурным называется алгоритм, являющийся комбинацией линейного, разветвляющегося и циклического алгоритмов.

Сложность алгоритма характеризуется числом элементарных шагов выполнения программы, что пропорционально времени выполнения алгоритма, а также требуемым объемом оперативной памяти. Алгоритмически неразрешимой называется задача, для решения которой невозможно построить алгоритм.

На практике наиболее распространены следующие формы представления

алгоритмов:

· *словесная* (запись на естественном языке);

· *графическая* (изображения из графических символов);

· *псевдокоды* (полуформализованные описания алгоритмов на условном алгоритмическом языке, включающие в себя как элементы языка программирования, так и фразы естественного языка, общепринятые математические обозначения и др.);

· *программная* (тексты на языках программирования).

Словесный способ записи алгоритмов представляет собой описание последовательных этапов обработки данных. Алгоритм задается в произвольном изложении на естественном языке. Словесный способ не имеет широкого распространения, так как такие описания:

· строго не формализуемы;

· страдают многословностью записей;

· допускают неоднозначность толкования отдельных предписаний.

Графический способ представления алгоритмов является более компактным и наглядным по сравнению со словесным. При графическом представлении алгоритм изображается в виде последовательности связанных между собой функциональных блоков, каждый из которых соответствует выполнению одного или нескольких действий. Такое графическое представление называется схемой алгоритма. В схеме алгоритма каждому типу действий (вводу исходных данных, вычислению значений выражений, проверке условий, управлению повторением действий, окончанию обработки и т.п.) соответствует геометрическая фигура, представленная в виде блочного символа. Блочные символы соединяются линиями переходов, определяющими очередность выполнения действий. В таблице приведены наиболее часто употребляе-

мые символы.

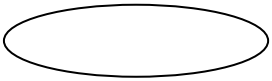
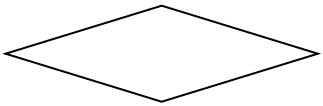
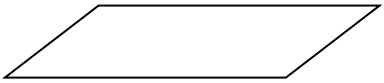
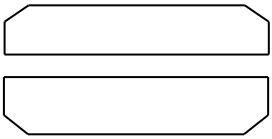
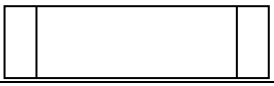
Название символа	Обозначение	Выполняемая функция
Начало/конец		Начало или конец алгоритма
Процесс вычислений		Выполняет вычисли-тельное действие или группу действий
Логический блок		Выбор направления выполнения алгоритма в зависимости от условия
Ввод /вывод		Отображение данных
Граница цикла		Отображает начало и конец цикла
Предопределенный процесс		Выполнение операций в подпрограмме
Соединитель		Указание связи между прерванными линиями в пределах одной страницы
Комментарий		Пояснительная запись

Схема алгоритма выстраивается в одном направлении: либо сверху вниз, либо слева направо. Все повороты соединительных

линий выполняются под углом 90 градусов.

Общими правилами при построении схем алгоритмов являются следующие:

- В начале алгоритма должны быть блоки ввода значений входных данных.
- После ввода значений входных данных могут следовать процесс вычислений и блоки условия.
- В конце алгоритма должны располагаться блоки вывода значений выходных данных.
- В алгоритме должен быть только один блок начала и один блок окончания.

Связи между блоками указываются направленными или ненаправленными линиями.

Псевдокод представляет собой систему обозначений и правил, предназначенную для единообразной записи алгоритмов. Псевдокод занимает промежуточное место между естественным и формальным языками. С одной стороны, он близок к обычному естественному языку, поэтому алгоритмы могут на нем записываться и читаться как обычный текст. С другой стороны, в псевдокоде используются некоторые формальные конструкции и математическая символика, что приближает запись алгоритма к общепринятой математической записи.

Алгоритмы линейной структуры

Алгоритм, в котором действия выполняются последовательно друг за другом, называется *линейным алгоритмом.*

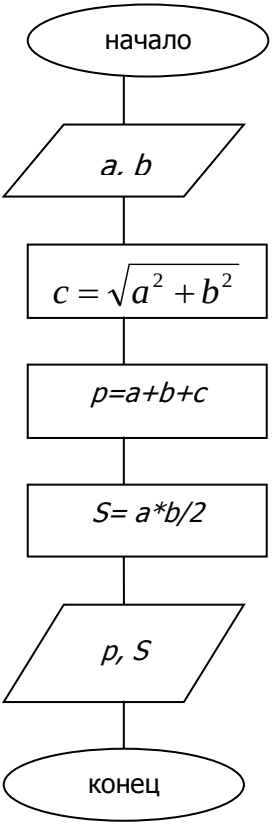
Рассмотрим пример составления схемы линейного алгоритма:

Пример

Вычислить периметр и площадь прямоугольного треугольника по длинам двух катетов.

Решение. Гипотенуза прямоугольного треугольника вычисляется по формуле Пифагора $c = \sqrt{a^2 + b^2}$, где *a, b* – катеты. Периметр и площадь прямоугольного треугольника определяются по формулам: *p=a+b+c, S= a*b/2.*

Схема алгоритма примера (рис) иллюстрирует, что сначала вводятся исходные данные *a, b* затем вычисляется гипотенуза, периметр, площадь. Все вычисления производятся последовательно и менять их местами нельзя.



Алгоритмы разветвляющейся структуры

На практике редко удается представить схему алгоритма решения задачи в виде линейной структуры. В программу может быть включено условие (например, выражение отношения или логическое отношение), в зависимости от которого, вычислительный процесс идет по той или иной ветви. Алгоритм такого вычислительного процесса называется *алгоритмом разветвляющейся структуры*. В общем случае количество ветвей в таком алгоритме

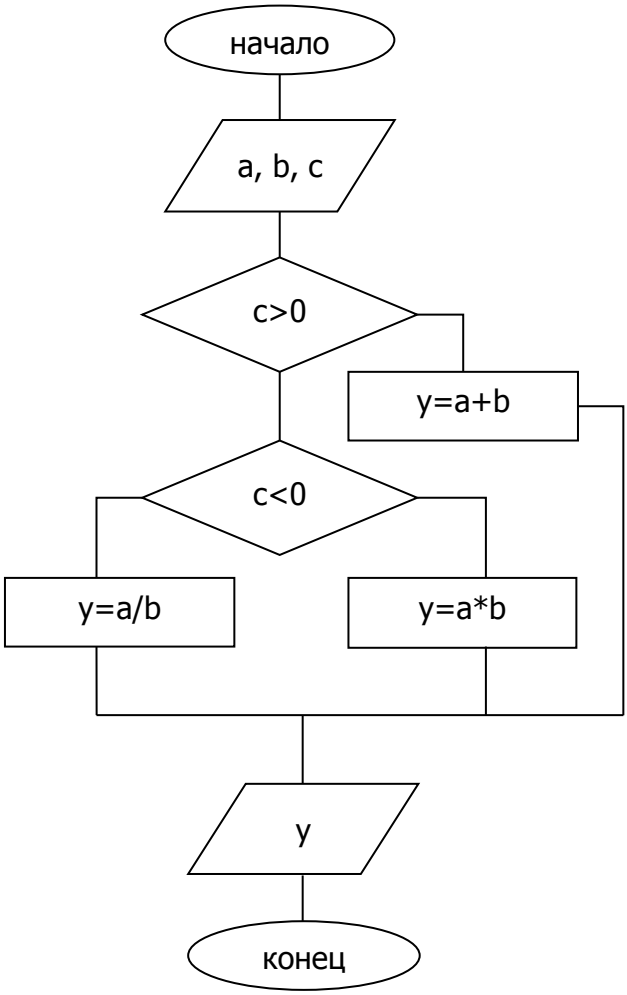
не обязательно равно двум.

Пример

Составить схему алгоритма вычисления выражения

$$y = \begin{cases} a + b, & c > 0 \\ a * b, & c < 0 \\ a / c, & c = 0 \end{cases}$$

После ввода исходных данных (переменных a, b, c) проверяется значение переменной c в логическом блоке. Если условие выполняется (истина), то после выполнения блоков первой ветви нет необходимости выполнять блоки второй ветви, осуществляется переход сразу к выводу результата и концу алгоритма. Если условие не выполняется (ложь), то переходим к проверки следующего логического условия. Решение указывается условием перехода на соответствующую ветвь.



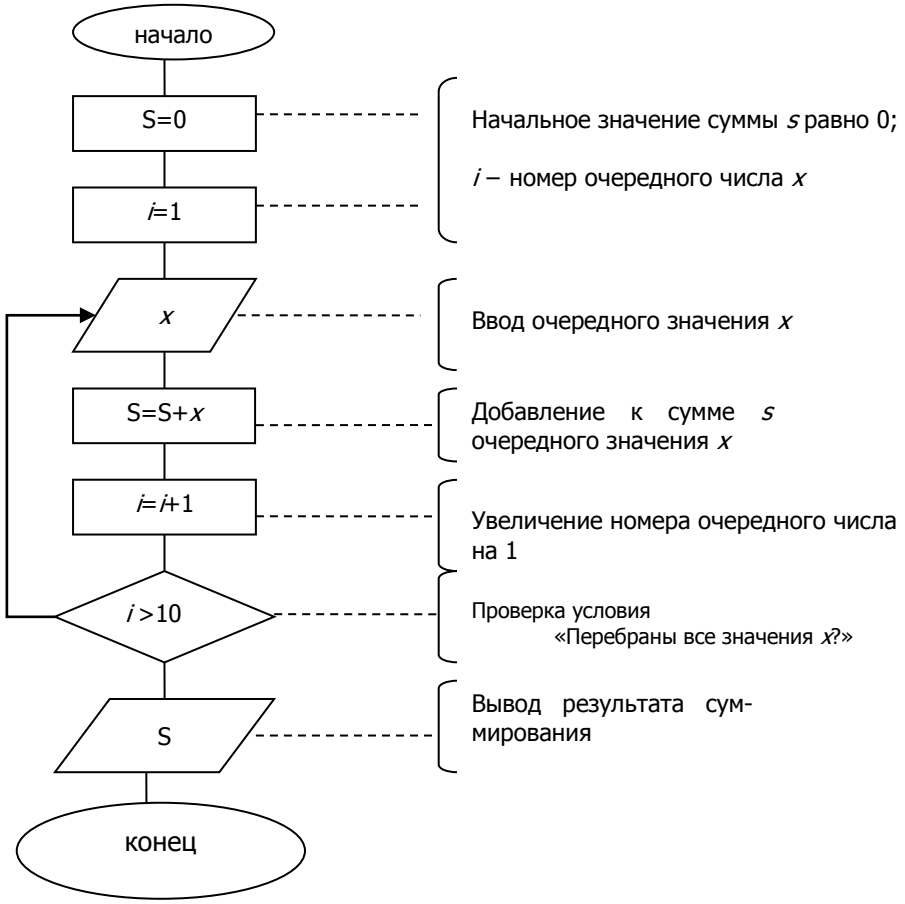
Алгоритмы циклической структуры

Алгоритм, в котором вычисления повторяются по одной и той же совокупности формул, называется *циклическим алгоритмом. Цикл* – это многократно повторяемый участок алгоритма.

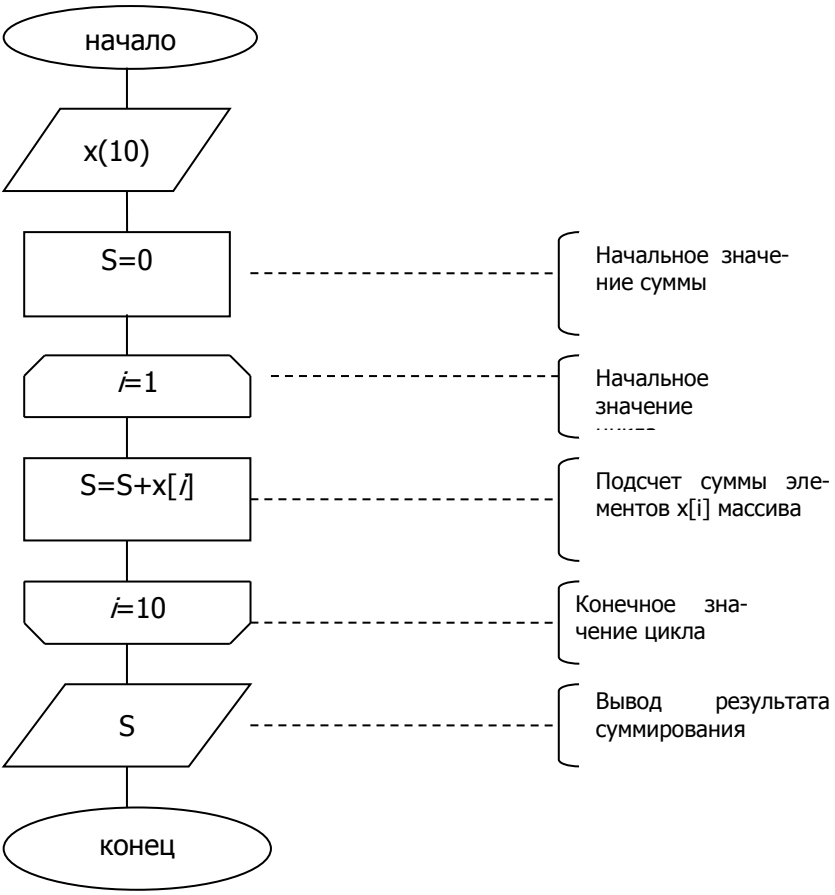
Массив это совокупность переменных, которые имеют одно и то же имя и тип. Элементы массива различаются по индексу. Имя общее, индекс оригинальный. Упорядоченность данных в массиве позволяет обращаться к любому элементу массива по его номеру (индексу), а однотипность данных позволяет использовать циклическую обработку всех элементов Различают одномерные массивы (1 индекс) – они используются для представления векторов и двумерные массивы (2 индекса) они используются для представления матриц.

Пример Составить циклический алгоритм вычисления суммы десяти чисел $S=\sum_{i=1}^{10} x_i$.

Вариант 1 построения алгоритма. Здесь в качестве переменной цикла используется переменная *i* с начальным значением, равным единице, и конечным значением, равным 10, и шагом, равным единице. В этом цикле проверка условия выхода из цикла выполняется в конце цикла. При этом тело цикла повторится десять раз.

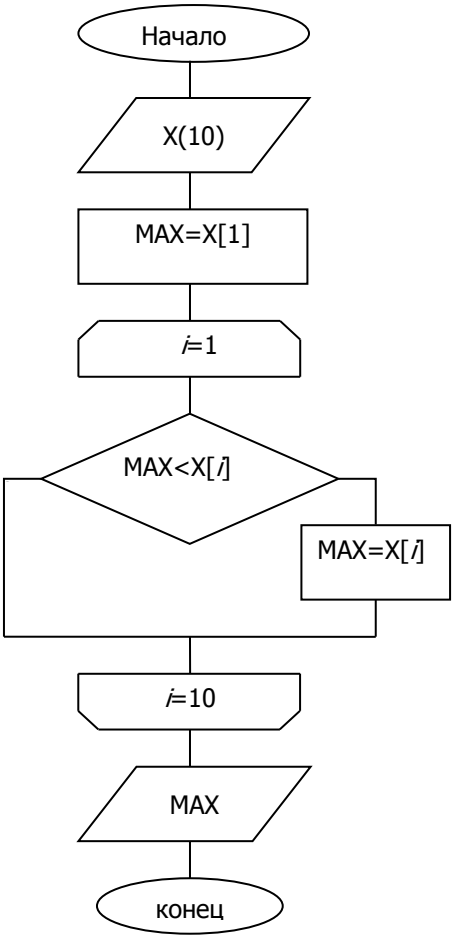


Вариант 2 построения алгоритма. Схема алгоритма получается во многих случаях более компактной и наглядной, если для ее построения использовать блоки начала и конца цикла, который выполняет все функции, необходимые для его организации. В цикле последовательно суммируются все элементы x_i массива с начальным значением S=0.

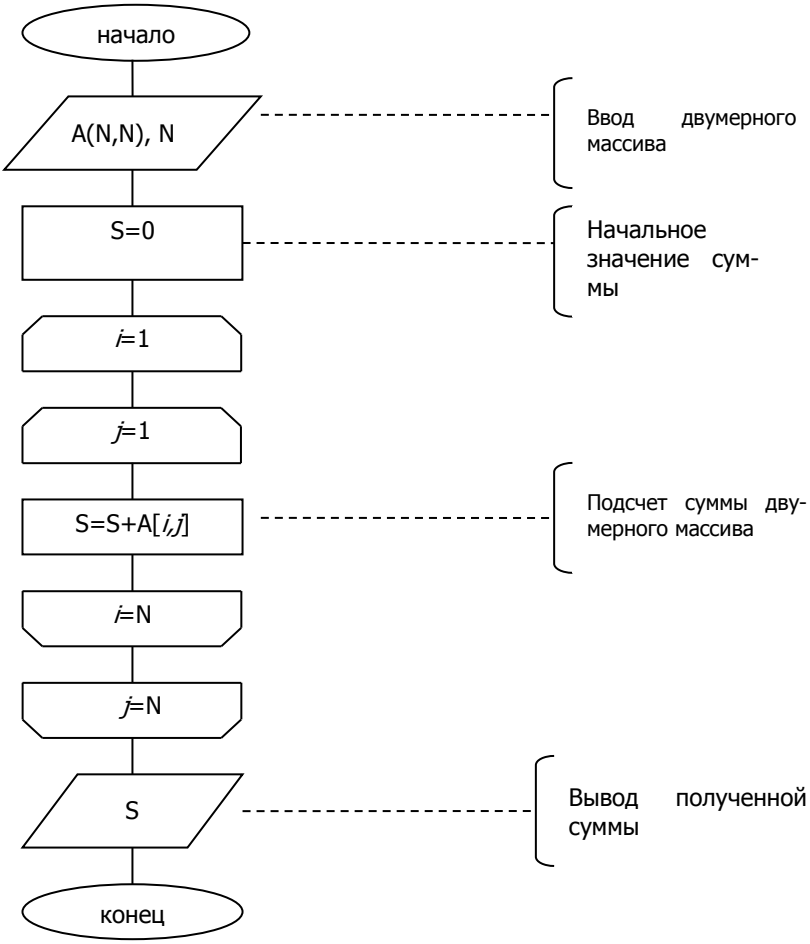


Пример Найти наибольший элемент одномерного массива, состоящего из 10 элементов.

Процесс определения наибольшего элемента заключается в цикле сравнивая текущий элемент массива с некоторым, например, с первым, условно принятым за наибольшим. Если текущее значение массива окажется больше наибольшего из предыдущих значений, то его надо считать новым наибольшим значением.



Пример Задан двумерный массив A(N,N), найти сумму элементов массивов.


начало
A(N,N), N
S=0
i=1
j=1
S=S+A[i,j]
i=N
j=N
S
конец
Ввод двумерного массива
Начальное значение суммы
Подсчет суммы двумерного массива
Вывод полученной суммы

12. ВВЕДЕНИЕ В ЯЗЫКИ ПРОГРАММИРОВАНИЯ

Языки программирования - это искусственные языки. От естественных они отличаются ограниченным числом «слов», значение которых понятно транслятору, и очень строгими правилами записи команд (операторов).

Транслятор-это специальное служебное приложение, которое либо переводит текст программы в машинный код, либо исполняет его.

С помощью языка программирования создается не готовая программа, а только ее текст, описывающий ранее разработанный алгоритм. Чтобы получить работающую программу, надо этот текст либо автоматически перевести в машинный код (для этого служат *программы - компиляторы*) и затем использовать отдельно от исходного текста, либо сразу выполнять команды языка, указанные в тексте программы (этим занимаются *программы-интерпретаторы*). Компиляторы полностью обрабатывают текст программы (исходный код). Они просматривают его в поисках синтаксических ошибок, выполняют определенный смысловой анализ и затем автоматический перевод (транслируют) на машинный язык - генерируют машинный код.

Интерпретатор моделирует некую виртуальную вычислительную машину, для которой базовыми инструкциями служат не элементарные команды процессора, а операторы языка программирования.

Уровни языков программирования

Разные типы процессов имеют разные наборы команд. Если язык программирования ориентирован на конкретный тип процессора и учитывает его особенности, то он *называется языком программирования низкого уровня*. В данном случае «низкий» не значит «плохой». Имеется ввиду, что операторы языка близки к машинному коду и ориентированы на конкретные команды процессора. Языком низкого уровня является язык *ассемблера,* который просто представляет каждую команду машинного кода, но не в виде чисел, а с помощью символьных, условных обозначений, называемых мнемониками. Однозначное преобразование одной машинной инструкции в одну команду ассемблера называют транслитерацией. Так как наборы инструкций для каждой модели процессора отличаются, конкретной компьютерной архитектуре соответствует свой язык ассемблера, написанная на нем программа может быть использована только в этой среде.

С помощью языков низкого уровня создаются очень эффектные и компактные программы, так как разработчик получает доступ ко всем возможностям процессора. С другой стороны, при этом требуется очень хорошо понимать устройства компьютера, затрудняется отладка больших приложений, а результирующая программа не может быть перенесена на компьютер с другим типом процессора. Подобные языки обычно применяют для написания небольших системных приложений, драйверов устройств, модулей стыковки с нестандартным оборудованием, когда важнейшими требованиями становятся компактность, быстродействие и возможность прямого доступа к аппаратным ресурсам.

В некоторых областях, например в машинной графике, на языке ассемблера пишутся библиотеки, эффективно реализующие требующие интенсивных вычислений алгоритмы обработки изображения.

Языки программирования высокого уровня значительно ближе и понятнее человеку, нежели компьютеру. Особенности конкретных компьютерных архитектур в них не учитываются, поэтому создаваемые программы на уровне исходных текстов легко переносимы на другие платформы. Разрабатывать программы на языках высокого уровня с помощью понятных и мощных команд значительно проще, а ошибок при создании программы допускается гораздо меньше.

Первыми языками программирования высокого уровня были COBOL, FORTRAN, затем ALGOL, BASIC, PL/1. Был накоплен определенный опыт в том, как эти языки должны быть устроены. Стало также ясно, что не может быть одного самого лучшего языка, и что при программировании различных задач удобнее использовать разные языки. В настоящее время языки программирования делятся на специализированные и универсальные. Специализированные используются для решения узкого класса задач. На универсальном языке можно запрограммировать любую задачу (вопрос об эффективности программирования и эффективности программы здесь не ставится). Универсальные условно делятся на простые и сложные. Простые имеют ограниченный набор средств и за счет этого проще в изучении и дают экономичный код (т.е. откомпилированная программа занимает меньше места в памяти и быстрее выполняется). Сложные имеют большее разнообразие синтаксических конструкций и зачастую сильно упрощают программирование, но сложны в изучении и дают менее экономичный код. Наиболее употребительными простыми языками являются PASCAL , C (более сложная версия - C++) и BASIC. В нашем

курсе мы будем изучать программирование на основе языка PASCAL. Другое деление языков - деление на императивные и декларативные. Императивные позволяют формулировать алгоритм в форме схемы отдельных операций (согласно приведенному выше определению алгоритма). Декларативные языки позволяют формулировать сразу цель программы, а алгоритм ее решения строится автоматически. Естественно, такие языки пригодны не для всех, а только для определенного класса задач, для которых формализован процесс составления алгоритма в классическом смысле. В качестве примера декларативных языков можно привести языки PROLOG и PLANNER.

Язык программирования Паскаль придуман швейцарским ученым Николасом Виртом в 1970г. Паскаль вначале предназначался для учебных целей, однако оказался настолько удачным, что широко распространился среди профессиональных программистов. Его достоинствами являются простота, естественность, хорошая усваиваемость при обучении и эффективность при реализации программ. При этом неоднократно делались попытки улучшить Паскаль за счет полезных нововведений. В результате для Паскаля, как и для других языков программирования, стала актуальной проблема приведения языка к единому стандарту, иначе терялось главное достоинство языка высокого уровня - универсальность и переносимость. Этот стандарт был создан в 1983г (стандарт ISO 7185 - 83). В этом стандарте зафиксированы те конструкции и термины Паскаля, которые должны присутствовать в каждой реализации и не могут быть изменены.

12.1 Конструкции языка Паскаль

Алфавит языка

Теперь рассмотрим алфавит языка программирования Паскаль - совокупность допустимых в языке символов (или групп символов, рассматриваемых как единое целое). В языке Pascal все компоненты формируются из множества символов стандарта ASCII. Элементы алфавита можно условно разбить на четыре группы:

- символы, используемые в идентификаторах;
- разделители;
- специальные символы;
- неиспользуемые символы.

Идентификатор - имя любого объекта программы - может включать буквы, цифры и символ подчеркивания.

Буквы - это 26 латинских букв (прописных и строчных) от

A до Z и от a до z. Помимо идентификаторов буквы могут использоваться в шестнадцатеричных числах для обозначения цифр от 10 до 15 (буквы от A до F и от a до f), строковых константах, служебных словах и комментариях. Следует иметь в виду, что прописные и строчные буквы в идентификаторах, числах и служебных словах не различаются.

Цифры – это арабские цифры от 0 до 9. В идентификаторах они могут присутствовать в любой позиции, кроме первой. Цифры используются также в изображении числовых констант. Символ подчеркивания может находиться в любой позиции.

Длина идентификатора может быть любой, но значимыми являются только первые 63 символа, и, по этим символам все идентификаторы должны быть уникальными.

Разделители используются для отделения друг от друга идентификаторов, чисел, зарезервированных слов. В качестве разделителей можно использовать:

- пробел;
- любой управляющий символ (коды от 0 до 31), включая символ возврата каретки (код 13);
- комментарий.

В любом месте программы, где можно поместить один разделитель, их можно поместить любое количество и в любом сочетании.

Комментарии заключаются либо в фигурные скобки {}, либо в скобки вида (* *) и могут занимать любое число строк. Комментарий, в котором за открывающей скобкой идет знак $, является директивой компилятора. Во время компиляции программы все комментарии, за исключением директив компилятора, игнорируются.

Специальные символы, выполняющие в языке определенные функции, можно разделить на три категории:

- разделители (знаки пунктуации);
- знаки операций;
- зарезервированные слова.

Знаки операций предназначены для обозначения тех или иных арифметических, логических или других действий. Они бывают двух типов: состоящие из небуквенных символов (например, + - * и т. д.) и буквенные операции (например, not, div, mod и т.д.), представляющие собой зарезервированные слова (таблица 1 и 2).

Зарезервированные слова включают служебные слова (например, begin, end, div и т. д.) и имена директив (например,

external, forward и т. д.). Служебные слова (таблица 3) можно использовать только по своему прямому назначению и их нельзя переопределять. Директивы также имеют свое определенное назначение, но в отличие от служебных слов их можно переопределить, однако делать это крайне нежелательно.

Таблица 1

Арифметические операции и стандартные функции

Арифметические операции		Стандартные функции	
Обозначение	Выполняемые действия	Обозначение	Вычисляемое значение
+	Сложение	ABS(X)	Абсолютное значение X
-	Вычитание	SQR(X)	x^2
*	Умножение	PRED(X)	Выдает предшествующее X целое значение
/	Деление	SUCC(X)	Выдает следующее за X целое значение
DIV	Целочисленное деление	SIN(X)	Sin(x)
MOD	Вычисление остатка от целочисленного деления	COS(X)	Cos(x)
		ARCTAN(X)	Arctan(x)
		LN(X)	Ln(x)
		EXP(X)	e^x
		SQRT(X)	$\sqrt{x}$
		TRUNC(X)	Выделение целой части X
		ROUND(X)	Целое число, ближайшее к X

Таблица2.

Логические операции Паскаля

Операции с битами информации	
NOT	унарная операция инверсии всех битов целого числа
AND	побитовая логическая операция И двух целых чисел
OR	побитовая логическая операция ИЛИ двух целых чисел
XOR	побитовая логическая операция ИСКЛЮЧАЮЩЕЕ ИЛИ двух целых чисел
Операции отношения	
: =	равное
<=	меньше или равно
<>	не равно
>=	больше или равно
<	меньше
>	больше

Зарезервированные слова включают служебные слова (например, begin, end, div и т. д.) и имена директив (например, external, forward и т. д.). Служебные слова (таблица 3) можно использовать только по своему прямому назначению и их нельзя переопределять. Директивы также имеют свое определенное назначение, но в отличие от служебных слов их можно переопределить, однако делать это крайне нежелательно.

Таблица 3

Служебные слова Паскаля

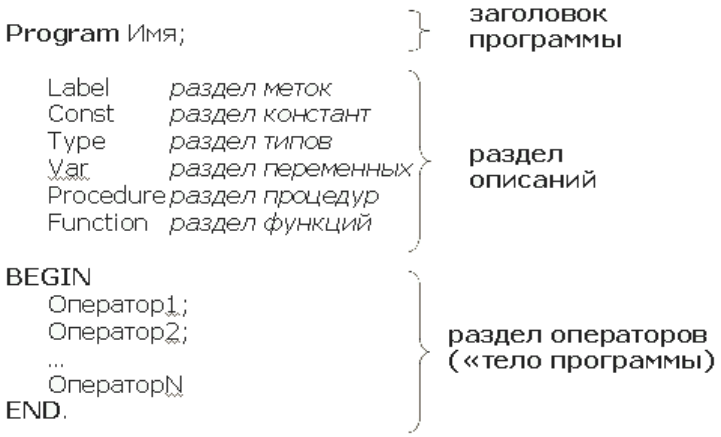
and	else	inline	procedure	unit
asm	end	interface	program	until
array	external	interrupt	record	uses
begin	file	label	repeat	var
case	for	mod	set	while
const	forward	nil	shl	with
constructor	function	not	shr	xor
destructor	goto	object	string	
div	if	of	then	

do	implementa-tion	or	to	
downto	in	packed	type	

Неиспользуемые в Pascal *символы* кодировки ASCII, такие, как, например, **%, &, "** и т. д., включая буквы русского алфавита, можно использовать в комментариях и символьных строках.

Структура программы

Программа на языке Pascal состоит из заголовка, раздела описаний и раздела операторов.



Заголовок программы

Заголовок содержит служебное слово PROGRAM, имя программы, задаваемое программистом. Заканчивается заголовок символом ";" (точка с запятой).

Раздел описаний предназначен для объявления всех встречающихся в программе данных и их характеристик (имена данных, их тип, возможные значения). Этот раздел содержит следующие подразделы: объявление меток, констант, типов, переменных, объявление процедур и функций. Порядок расположения разделов не важен и описания могут повторяться.

Объявление процедур и функций является одним разделом. Следует заметить, что не все перечисленные разделы обязательны в программе. В простых программах могут потребоваться, например, только разделы описания констант и переменных.

После каждого описания ставится символ ";".

Раздел операторов ("те- ло" программы) заключается в

операторные скобки вида: BEGIN ("начать") и END ("окончить"), при этом после служебного слова END обязательно ставится точка. В разделе операторов записывается последовательность исполняемых операторов и каждый выражает действие, которое необходимо выполнить. Исполняемые операторы отделяются друг от друга символом ";".

Хороший стиль программирования требует:

- записывать слова PROGRAM, BEGIN, END с одинаковой позиции строки;
- по отношению к ним описания и операторы принято сдвигать вправо на 3 символа. Желательно сдвиг делать на одинаковое количество позиций от края или по отношению к предыдущему сдвигу.

Раздел описания меток

Метки являются условными номерами (именами) операторов. Одной меткой можно пометить только один оператор. Метка от помеченного оператора отделяется двоеточием. Используемые в Turbo Pascal метки, могут быть двух типов:

целым числом в пределах от 0 до 9999;

обычным идентификатором.

Все используемые метки должны быть перечислены в разделе объявления меток, начинающемся зарезервированным словом **Label**, например:

Label 1, 2, Metka;

Раздел описания констант

Константами называются параметры программы, значения которых не меняются в процессе ее выполнения

В языке Паскаль возможно двоякое использование констант:

- непосредственное использование значения константы,
- использование идентификатора константы

Задание констант идентификаторами осуществляется в разделе объявления констант, начинающемся зарезервированным словом **Const**.

В Turbo Pascal имеется две разновидности констант, заданных идентификаторами:

- обычные константы, тип которых определяется их значением;
- типизированные константы, для которых в явном виде указывается их тип.

Обычные константы могут быть целого, вещественного,

символьного, логического типа и типа **string**, типизированные константы - любого типа, кроме типа файл (или содержащего компоненту типа файл). Для обычных констант задаются их имена и значения, разделенные знаком равенства.

Const
<имя константы 1> = <значение 1>,
<имя константы 2> = <значение 2>,
<имя константы N> = <значение N> ;

Значения констант могут задаваться как конкретными величинами соответствующего типа, так и константными выражениями.

Целые константы представляются знаком и цифрами (знак + можно не указывать). Предполагается десятичная система счисления, однако можно использовать и шестнадцатеричную. В этом случае перед константой следует поставить знак **$**. Диапазон целых десятичных чисел - от -2147483648 до 2147483647, а шестнадцатеричных - от $00000000 до $FFFPFFPF. В Turbo Pascal заданы два предопределенных целых числа:

Maxint = 32767
MaxLongInt = 2147483647

Вещественные константы в своем изображении содержат десятичную точку или/и показатель степени (символ E или e), например:

-0. 5
1.07
-1E-5
5E+15

Строковая константа (строка символов) - это последовательность любого, в том числе и равного нулю, количества символов из набора ASCII, расположенных на одной строке и заключенных в апострофы. Ограничением здесь может быть максимальный размер строки, воспринимаемый компилятором (не более 126 символов).

Строка, состоящая из одного символа, называется *символьной константой.*

Если между апострофами нет ни одного символа, то такая строка называется *нулевой строкой.*

При необходимости поместить в строку символ ' (апостроф), его следует записать дважды, например 'Язык ' Pascal''.

Пример. Описание строковых констант**.**

Const
Symbols ='TURBO',

Apostroph = ''';

Константные выражения - это такие выражения, которые могут быть вычислены на стадии компиляции без запуска программы. Они являются частным случаем выражений и могут состоять из констант, знаков операций, круглых скобок и некоторых стандартных функций.

Пример. Константные выражения.

'A' {выражение, состоящее из символьной константы}

Pi/4 {выражение с использованием функции и знака операции}.

В константных выражениях можно использовать следующие функции. **Abs, Chr, Hi, High, Length, Lo, Low, Odd, Ord, Pi, Pred, Ptr, Round, SizeOf, Succ, Swap, Trunc.**

Типизированные константы фактически представляют собой инициализированные переменные и могут использоваться наравне с обычными переменными, в том числе и в левой части операции присваивания.

Для каждой такой константы задается ее имя**,** тип и начальное значение Тип от имени отделяется двоеточием, начальное значение от типа - знаком равенства

const

<имя константы>:<тип константы> = <значение константы>.

Пример. Объявление типизированных констант

```
Const
 Maximum: Integer = 9999,
 Rea: Real =-05,
 Bell: Char = #7;
```

Следует иметь в виду, что типизированные константы (в том числе и константы процедур и функций) инициализируются только раз - в начале работы программы.

Раздел описания типов

Тип данных – это множество допустимых значении данных, а также совокупность операций над ними.

В Turbo Pascal можно выделить следующие группы типов:

- простые типы;
- структурированные типы;
- указатели;
- процедурные типы;
- объекты.

Простые типы, используемые в языке, разделяются на стандартные (предопределенные) и определяемые программистом.

Стандартный тип, не требующий предварительного определения, включает:

- целые типы - **Integer**
- вещественные типы - **Real**
- логические типы - **Boolean**
- символьный тип –**Char**
- тип-строка -**String**
- ASCIIZ-строка - **PChar**
- текстовый файл - **Text**
- тип-указатель – **Pointer.**

Все другие используемые типы данных должны быть определены либо в разделе объявления типов, либо в разделе объявления переменных или типизированных констант.

Раздел объявления типов начинается зарезервированным словом **type**, после которого определяются вводимые типы. Определение каждого нового типа начинается с идентификатора типа. За ним следует знак равенства, а далее - само определение, завершающееся точкой с запятой:

type
<имя типа 1> = <определение типа 1>;
<имя типа 2> = «определение типа 2>;
<имя типа N> = «определение типа N>;

В данном разделе будут описаны только простые типы данных, все остальные типы будут рассмотрены после описания переменных, констант и выражений.

Простой тип определяет упорядоченное множество значений параметра. В Turbo Pascal имеются следующие группы простых типов:

- целые типы
- логический тип
- символьный тип
- перечисляемый тип
- тип-диапазон
- вещественные типы

Все простые типы, за исключением вещественных, называются порядковыми типами. Для величин порядковых типов определены стандартные процедуры и функции: **Dec, Inc, Ord, Pred, Succ**.

Целый тип используется для описания целых чисел. В отличие от языка Паскаль, где определен единственный целый тип Integer, в Turbo Pascal имеется пять стандартных типов целых чисел, характеристики которых в приведены в таблице 4.

Таблица 4.

Целые типы данных

Название типа	Идентификатор	Диапазон представления чисел	Формат	Размер в байтах
Короткое целое со знаком	Shortint	-128..127	знаковый	1
Целое со знаком	Integer	-32768..32767	знаковый	2
Длинное целое со знаком	Longint	-2147483648.. 2147483647	знаковый	4
Короткое целое без знака	Byte	0..255	беззнаковый	1
Целое без знака	Word	0..65535	беззнаковый	2

Стандартный логический тип Boolean (размер – 1 байт) представляет собой тип данных, любой элемент которого может принимать лишь два значения: **True** и **False.** При этом справедливы следующие условия:

False < True Succ (False) = True

Ord (False) =0 Pred (True) > False

Ord(True) = 1

В Turbo Pascal 7.0 добавлено еще три *логических типа* **ByteBool** (размер - 1 байт), **WordBool** (размер - 2 байта) и **LongBool** (размер - 4 байта). Они введены для унификации с другими языками программирования и со средой Windows. Отличие их от стандартного типа **Boolean** заключается в фактической величине параметра этого типа, соответствующей значению **True**. Для вех логических типов значению **False** соответствует число 0, записанное в соответствующее количество байтов. Значению же **True** для типа **Boolean** соответствует число 1, записанное в его байт, а для других типов значению **True** соответствует любое число, отличное от нуля (хотя функция **Ord** в этом случае дает значение 1).

Стандартный символьный тип Char определяет полный набор ASCII-символов. Функция Ord от величины типа Char дает код соответствующего символа. Сравниваются величины символьного типа по своим кодам.

Перечисляемый тип не является стандартным и определя-

ется набором идентификаторов, с которыми могут совпадать значения параметра. Список идентификаторов указывается в круглых скобках, идентификаторы разделяются запятыми:

Type

<имя типа> = (идентификатор1, идентификатор2, … , идентификаторN);

Важно, в каком порядке перечислены идентификаторы при определении типа т. к. первому идентификатору присваивается порядковый номер **0**, второму **1**;

Один и тот же идентификатор можно использовать в определении только одного перечисляемого типа. Функция **Ord** от величины перечисляемого типа дает порядковый номер ее значения.

Пример. Описание перечисляемого типа.

Type

Operat = (Plus, Minus, Mult, Divide);

Логический тип является частным случаем перечисляемого, типа:

Type Boolean = (False, True);

Тип-диапазон определяет подмножество значений порядкового типа, определяемое минимальным и максимальным значением, в которое входят все значения исходного типа, находящиеся в этих границах, включая и сами границы. Тип-диапазон задается указанием минимального и максимального значений, разделенных двумя точками:

Type

<имя типа> = <минимальное значение> .. <максимальное значение>;

Минимальное значение при определении такого типа не должно быть больше максимального.

Пример. Определение типов-диапазонов.

type

Dozen = l..1999; {числа от 1 до 1999}

AddSub =Plus .. Minus; {операции слоения и вычитания}

Вещественные типы используются для описания действительных чисел. В Turbo Pascal имеется пять стандартных вещественных типов: **Rea1, Single, Double, Extended, Comp**. Характеристики этих типов представлены в таблице5.

Таблица 5.

Вещественные типы данных

Название типа	Идентификатор	Диапазон представления чисел	Значащие цифры мантиссы	Размер памяти в байтах
Вещественное одинарной точности	Singl	$1.5\cdot10^{-45}..3.4\cdot10^{38}$	7..8	4
Вещественное	Real		11..12	6
Вещественное двойной точности	Double	$5.0\cdot10^{-324}..1.7\cdot10^{308}$	15..16	8
Вещественное повышенной точности	Extended		19..20	10
Целое в формате вещественного	Comp	$2^{-63}+1..2^{63}-1$ $(-9.2\cdot10^{18}..9.2\cdot10^{18})$	19..20	10

Тип *Comp* фактически является типом целых чисел увеличенного диапазона, однако порядковым не считается.

Раздел описания переменных

Переменными называются параметры программы, значения которых могут изменяться в процессе ее выполнения.

Все используемые в программе переменные должны быть определены с указанием их типов. Переменные определяются:

- в разделе объявления переменных программы;
- в разделе объявления переменных подпрограммы;
- в разделе объявления переменных модуля;
- в заголовке подпрограммы.

Раздел объявления переменных начинается зарезервированным словом **var**, за которым следуют объявления конкретных переменных, состоящие из имени переменной (имен переменных, перечисленных через запятую, если они одного типа) , двоеточия и типа переменной (переменных). Каждое объявление завершается точкой с запятой:

var
<список переменных 1>: <тип 1>;
<список переменных 2>: <тип 2>;
<список переменных N>: <тип N>;

В разделе объявления переменных наряду с предопределенными типами можно использовать типы, объявленные перед

этим в разделе объявления типов, а также новые, вводимые только для конкретных переменных.

В заголовке подпрограммы при определении переменных можно использовать лишь стандартные или ранее определенные типы.

Пример. Объявление переменных

```
Type
Operat = (Plus, Minus, Mult, Divide);
var
X, Y, Z: Real;        {предопределенный тип}
I, J, K: Integer;     {предопределенный тип}
Operator: Operat;  {ранее определенный тип}
Digit: 0..9;          {объявление нового типа}
```

Выражение - это синтаксическая единица языка, определяющая способ вычисления некоторого значения. Выражения в языке Паскаль формируются в соответствии с рядом правил из констант, переменных, функций, знаков операций и круглых скобок.

Стандартные функции заранее разработанных подпрограмм-функций, которые можно использовать как готовые объекты в Pascal. Их количество увеличено по сравнению со стандартом языка, и все они объединены в стандартные модули. Рассмотрим наиболее часто используемые стандартные функции.

Арифметические функции можно использовать только с величинами целого и вещественного типа. Их перечень приведен в таблице 1.

Функции преобразования типа предназначены для преобразования типов величин, например символа в целое число, вещественного числа в целое и т д. К ним относятся следующие функции.

Chr(X) - преобразование ASCII-кода в символ. Аргумент функции должен быть целого типа в диапазоне (0. 255). Результатом является символ, соответствующий данному коду.

High(X) - получение максимального значения величины. Аргумент функции - параметр или идентификатор порядкового типа, типа-массива, типа-строки или открытый массив. Результат функции для величины порядкового типа - максимальное значение этой величины, типа-массива - максимальное значение индекса, типа-строки -объявленный размер строки, открытого массива - количество компонент массива минус 1 (максимальный индекс, при начале нумерации с нуля).

Low(X) - получение минимального значения величины. Ар-

гумент функции - параметр или идентификатор порядкового типа, типа-массива, типа-строки или открытый массив. Результат функции для величины порядкового типа - минимальное значение этой величины, типа-массива - минимальное значение индекса, типа-строки или открытого массива - **0**.

Ord(X) - преобразование любого порядкового типа в целый тип. Аргументом функции может быть величина любого порядкового типа (логический, символьный, перечисляемый). Результатом является величина типа Longint.

Round (X) - округление вещественного числа до ближайшего целого, і Аргумент функции - величина вещественного типа, а результат - округленная до ближайшего целого величина типа Longint. Если результат выходит за диапазон значений Longint, то при выполнении программы возникает ошибка.

Trunc(X) - получение целой части вещественного числа. Аргумент функции - величина вещественного типа, а результат - целая часть этого числа. Тип результата - Longint. Если результат выходит за диапазон значений Longint, то во время выполнения программы возникает ошибка.

Функции для величин порядкового типа позволяют выполнить ряд действий над величинами порядкового типа (найти предыдущий или последующий элемент, проверить число на нечетность). К этим функциям относятся следующие:

Odd(X) - проверка величины X на нечетность. Аргументом функции является величина типа Longint, результат равен True, если аргумент нечетный, и False - если четный.

Pred(X) - определение предыдущего значения величины X. Аргументом функции является величина любого порядкового типа, результатом - предшествующее значение того же типа (например, Pred(2) равно 1). При применении функции к первому элементу последовательности возникает ошибка.

Succ(X) - определение последующего значения величины X. Аргументом функции является величина любого порядкового типа, результатом - последующее значение того же типа (например, Succ(2) равно 3) При применении функции к последнему элементу последовательности возникает ошибка.

Знаки операций определяют действия над данными в программе. Все операции в Turbo Pascal можно разбить на следующие группы:

- арифметические операции;
- логические операции;
- операции с битами информации;

- операции со строками;
- операции отношения;
- адресная операция @.

Если в операциях используется несколько данных, то их типы должны быть либо идентичными, либо совместимыми.

Арифметические операции применимы только к величинам целых и вещественных типов. Их можно разделить на унарные и бинарные операции.

Унарный знак плюс +, поставленный перед величиной либо целого, либо вещественного типа, не оказывает никакого влияния на значение этой величины.

Унарный знак минус -, поставленный перед величиной либо целого, либо вещественного типа, приводит к изменению знака величины.

Бинарные арифметические операции и их знаки приведены в таблице 1.

Знаки операций **+, -** и ***** используются также и с другими типами операндов, но тогда они имеют иной смысл. В операциях деления делитель не должен равняться нулю. При использовании знака операции, являющегося служебным словом, он должен быть отделен от операндов хотя бы одним разделителем.

Пример. Арифметические выражения

A mod B (Если A=10 и B=3, то результат равен 1)

C div D (Если C=10 и D=3, то результат равен 3)

C+D/(F-C)/SIN(D)

Логические операции применяются к величинам логического типа (Таблица 2), результат операции - тоже логического типа. Имеется одна унарная логическая операция not (ОТРИЦАНИЕ) и три бинарные операции **and** (И), **or** (ИЛИ), **xor** (ИСКЛЮЧАЮЩЕЕ ИЛИ).

Операции с битами информации **not, and, or, xor** можно использовать для побитовых операций с целыми числами, при этом тип результата определяется наименьшим типом операндов (имеющим наименьший размер). В применении к целым числам эти операции имеют следующий смысл:

not - унарная операция инверсии всех битов целого числа;

and - побитовая логическая операция И двух целых чисел;

or - побитовая логическая операция ИЛИ двух целых чисел;

xor - побитовая логическая операция ИСКЛЮЧАЮЩЕЕ ИЛИ двух целых чисел.

К этой же группе можно отнести операции shi и shr, имеющие следующий смысл'

shi - операция I shi J сдвигает содержимое I на J битов влево Освободившиеся биты заполняются нулями.

shr - операция I shr J сдвигает содержимое I на J битов вправо. Освободившиеся биты заполняются нулями.

Операции отношения (таблица 2) предназначены для сравнения двух величин (величины должны быть сравнимых типов). Результат сравнения имеет логический тип. Операции отношения следующие:

Круглые скобки используются для заключения в них части выражения, значения которой необходимо выполнить в первую очередь. В выражении может быть любое количество круглых скобок, причем количество открывающих круглых скобок должно быть равно количеству закрывающих круглых скобок. Части выражений, заключенные в круглые скобки, должны быть либо непересекающимися, либо вложенными друг в друга.

Раздел действий (тело программы) начинается словом **begin** и предназначен для записи операторов. Операторы в разделе действий разделяются точкой с запятой. Заканчивается раздел действий словом **end**. (с точкой, которая является признаком конца программы).

12.2 Операторы языка Паскаль

Операторы языка описывают некоторые алгоритмические действия, которые необходимо выполнить для решения задачи. Тело программы можно представить как последовательность таких операторов. Идущие друг за другом операторы программы разделяются точкой с запятой.

Все операторы языка Паскаль можно разбить на две группы: простые и структурированные.

Простые операторы не содержат в себе других операторов. К ним относятся:

- оператор присваивания;
- обращение к процедуре;
- оператор безусловного перехода GOTO;
- пустой оператор.

Оператор присваивания выполняет присваивание переменной или функции значение выражения. Для этого используется знак присваивания **:=** , слева от которого записывается имя переменной или функции, которой присваивается значение, справа - выражение, значение которого вычисляется перед присваиванием:

<Переменная>:=<Выражение>

Допустимо присваивание значений переменным и функциям любого типа, за исключением типа файл. Тип выражения и тип переменной (или функции) должны быть совместимы для присваивания.

Пример. Операторы присваивания

```
X := Y;
Z :=A + B;
Res := (I>0) and (K<100) ;
I := Sqr(J) + I*K;
```

Оператор безусловного перехода GOTO позволяет изменить стандартный последовательный порядок выполнения операторов и перейти к выполнению программы, начиная с заданного оператора. Оператор, на который происходит переход, должен быть помечен меткой. Эта же метка должна быть указана и в операторе GOTO.

Использовать оператор GOTO следует крайне осторожно. Широкое его применение без особых на то оснований ухудшает понимание логики работы программы. Безусловный переход можно осуществлять далеко не из каждого места программы и не в любое место программы. Так, нельзя с помощью этого оператора перейти из основной программы в подпрограмму или выйти из подпрограммы, не рекомендуется осуществлять переход внутрь структурированного оператора, т. к. он может дать неправильный результат.

Пример. Найти частное от деления целых чисел.

```
Program Primer;
Label Out;          {описание метки}
Var X, Y, Res: Integer;     {описание переменных}
begin
Write('Введите делимое');{вывод сообщения на экран}
ReadLn(X); {чтение числа}
Write('Введите делитель');{вывод сообщения на экран}
ReadLn(Y); {чтение числа}
if Y = 0 then
begin {Составной оператор}
WriteLn('Деление на ноль!'); {выход при нулевом делителе}
goto Out;
end
        else Res:=X div Y;
WriteLn('Частное равно: ', Res);
```

Out: ; {метка "пустого" оператора}
end.

Пустой оператор не выполняет никакого действия и никак не отображается в программе (за исключением, быть может, метки или точек с запятыми, отделяющих пустой оператор от предыдущих или последующих операторов). Он может потребоваться для осуществления на него безусловного перехода.

Ввод числовых данных можно осуществлять оператором присваивания, например:

A:=5;
BB:=6.143;

Однако в этом случае программа становится не универсальной, так как выполняется только при этих значениях переменных. Для выполнения программы при различных значениях переменных предназначен оператор ввода **Read.**

Как только во время выполнения программы встречается оператор **Read**, машина останавливается и ожидает ввода числовых значений. Когда числовые значения введены, процесс выполнения программы продолжается. Оператор ввода имеет вид

Read (a1, a2,…. an),

где a1, a2, , … , an — переменные, которые последовательно получают вводимые значения.

Числовые значения указываются через пробел, признаком окончания ввода является нажатие клавиши **<Enter>.** Числовые значения вводятся после набора на экране дисплея всей программы и запуска ее на выполнение.

Пусть переменным **A, B, C** необходимо присвоить следующие значения в процессе выполнения программы:

A=0.21 B=45.34 C=6371

Оператор ввода примет вид

Read (A, B, C),

а числовые значения можно ввести следующим образом:

0.21 45.34 6371

Если вновь повторить запуск программы, то можно ввести любые другие значения.

Если переменная описана как действительная (**Real**), а ее значение является целым числом, то можно вводить число как целое и как действительное. Машина сама преобразует целое число в действительное.

Допускается использование оператора ввода без параметров **ReadLn,** осуществляющего переход на новую строку при вводе данных. Дополнительно к этому имеется оператор ввода

ReadLn (a1, a2,…. an),

который сначала вводит значения a1, a2, ..., an, а затем осуществляет переход на новую строку.

Вывод данных из памяти ЭВМ на экран дисплея осуществляет оператор вывода **Write.** Форма записи оператора

Write (a1, a2,..., an),

где ai, a2, ..., an являются выражениями (переменными, строкой символов, функциями, константами).

Например, оператор

Write ('Значение B-',B)

выводит на экран дисплея строку

Значение **B-,** а затем значение переменной B.

Для вывода целых и действительных чисел можно указывать форматы в операторе **Write**. Формат указывается через двоеточие после переменной. Для действительных чисел формат состоит из двух величин. Первая величина обозначает общее поле выводимого значения, второе — поле дробной части. Общее поле включает в себя отрицательный знак числа или пробел для положительного числа, количество цифр в целой части, точку и количество цифр в дробной части.

Так, вывод значения Y в соответствии с форматом WRITE (Y:5:2) означает, что на изображение всего значения Y отведено пять позиций, из них две — на дробную часть.

Если формат отведен больше, чем количество позиций, занимаемых числом, то перед целой частью будет отведено соответствующее количество пробелов, а после дробной части — соответствующее количество нулей.

Для вывода целых чисел формат дробной части не указывается.

В языке Паскаль допускается использование оператора вывода без параметров**WriteLn,** который осуществляет переход на новую строку экрана дисплея. Последующий оператор вывода с параметрами будет выводить данные на новую строку экрана. Оператор вывода без параметров часто используется для пропуска пустых строк.

Оператор вывода

Writeln (a1, a2,…. an),

осуществляет сначала вывод на экран дисплея значений a1, a2, ..., an, а затем — переход на новую строку. Таким образом, данный оператор эквивалентен двум оператор;

Write (ai,a2,..., an);

WriteLn;

Операторы ввода и вывода используются практически во всех программах.

Перед вводом данных рекомендуется давать поясняющий текст с помощью оператора WRITE. Этим самым устанавливается диалог пользователя и машин. Например, для ввода значений A, B, C лучше указать

Write ('Введите значения A, B, C');
Read (A, B, C);

Таким образом, перед вводом числовых значений A, B, C на экране появится сообщение

Введите значения A, B, C

после чего можно вводить значения.

Составной оператор представляет собой совокупность последовательно выполняемых операторов, заключенных в операторные скобки **begin** и **end**:

begin
<оператор 1> ;
<оператор 2>;
<оператор N>
end

Он может потребоваться в тех случаях, когда в соответствии с правилами построения конструкций языка можно использовать один оператор, а выполнить нужно несколько действий. В такой составной оператор входит ряд операторов, выполняющих требуемые действия.

В дальнейшем везде, где будет указываться, что можно использовать один оператор, им может быть и составной оператор.

Отдельные операторы внутри составного оператора отделяются друг от друга точкой с запятой. Так как завершающее составной оператор слово **end** не является отдельным предложением, то перед ним точку с запятой можно не ставить, в противном случае компилятор будет считать, что перед словом **end** стоит пустой оператор.

Можно считать, что и само тело программы, т. к. оно заключено в операторные скобки **begin** и **end**, тоже является составным оператором.

Алгоритмы ветвящейся структуры

Условный оператор IF

Условный оператор IF реализует алгоритмическую конструкцию ВЕТВЛЕНИЕ и изменяет порядок выполнения операторов в зависимости от истинности или ложности некоторого условия.

Существует два варианта оператора:

if S **then** A **else** B; {полное ветвление} и

if S **then** A; {укороченная развилка}.

В этих операторах:

S - логическое выражение;

A - оператор, который выполняется, если выражение S истинно;

B - оператор, который выполняется, если выражение S ложно.

Так как условный оператор **IF** является единым предложением, ни перед **then**, ни перед **else** точку с запятой ставить нельзя.

Примеры использования оператора:

if X < 0 **then** X := -Y;

if X < 1.5 **then** Z := X + Y **else** Z := 1.5;

Условный оператор CASE

Условный оператор CASE позволяет выбрать решение из любого количества вариантов. Структура этого оператора в Turbo Pascal:

case S of

Cl: Instruction1;

C2: Instruction2;

• • •

CN: InstructionN;

else Instruction

end

В этой структуре:

- **S** – селектор (вычисляемое значение выражения порядкового типа);
- Cl, C2, ..., CN – метки вариантов (константы, с которыми сравнивается значение выражения S);
- Instructioni, Instruction2, ..., InstructionN - операторы, из которых выполняется тот, значение которого совпадает с **S**;
- Instruction - оператор, который выполняется, если значение выражения S не совпадает ни с одним из вариантов Cl, ..., CN.

Ветвь оператора **else** является необязательной. Если она отсутствует и значение выражения S не совпадет ни с одной из перечисленных констант, весь оператор рассматривается как пустой. В отличие от оператора **IF** перед словом **else** точку с запятой можно ставить.

Если для нескольких констант нужно выполнять один и тот же оператор, их можно перечислить через запятую (или даже указать диапазон, если возможно), сопроводив их одним оператором.

Алгоритмы циклической структуры

Для составления программы циклической структуры используются операторы цикла FOR, REPEAT, WHILE.

Оператор цикла FOR используется, когда известно число повторений. При этом различают две формы записи:

1. Оператор с организацией счета при изменении параметра цикла от начального (меньшего) значения до конечного (большего) значения этого параметра цикла. Такой оператор цикла имеет вид:

FOR K: = NZ TO KZ DO S;

2. Оператор с организацией счета при изменении параметра цикла от начального (большего) значения до конечного (меньшего) значения этого параметра цикла. Такой оператор цикла имеет вид:

FOR K: = NZ DOWNTO KZ DO S;

Здесь K — параметр цикла (целочисленная переменная); NZ, KZ — выражения, задающие соответствующее начальное и конечное значения параметра цикла; S — простой или составной оператор.

Оператор цикла выполняется следующим образом. Параметру цикла K присваивается начальное значения NZ. Затем управление передается в тело цикла и выполняется оператор S, после выполнения которого параметр цикла меняет свое значение на (единицу) (шаг изменения параметра цикла). При этом шаг равен +1, если используется оператор цикла с ключевым словом TO (случай 1), а если используется оператор цикла с ключевым словом DOWNTO (случай 2), то шаг равен –1. Далее измененное значение параметра цикла сравнивается с конечным значением KZ и, если параметр цикла не превышает KZ (случай 1) или превышает KZ (случай 2), то управления передается в тело цикла и выполняется оператором S; в противном случае осуществляется выход из цикла.

Рассмотрим фрагменты записи операторов цикла:

1) Y: = 0; FOR I = 3 TO 5 DO Y = Y + 1;

2) Y: = 0; FOR I = 12 DOWNTO 6 DO Y = Y + 1;

В результате выполнения первого оператора цикла параметр цикла 1 будет изменятся от начального значения I = 3 до конечного значения I = 5 с шагом, равным +1, причем по окон-

чании цикла Y = 3. При выполнении второго оператора цикла параметр цикла I будет изменятся от начального значения I = 12 до конечного значения I = 6 с шагом, равным -1. По окончании цикла Y = 7.

Пример. Найти: $\sum_{i=1}^{n} \cos\left(i + \frac{x^2}{4}\right)$, составить программу с применением оператора цикла FOR.

```
PROGRAM   SUMFOR;
VAR   X,Y: REAL;
  I, N: INTEGER;
BEGIN
WRITELN (‘ Ввести N , X’);
READ (N, X);
Y: = 0;
FOR  I: = 1   TO   N   DO
Y: = Y  + COS  (I + SQR (X) / 4);
WRITELN (‘ N = ‘ , N, ‘   ‘ , ‘ X = ‘ , X);
WRITELN (‘ Y = ‘ , Y);
END.
```

Оператор цикла WHILE позволяет организовать цикл, количество повторений которого зависит от включенного в него условия, т.е. цикл с неизвестным числом повторений. Этот оператор имеет вид:

WHILE LV DO S;

где LV — логическое выражение;

S — простой или составной оператор.

Выполнение оператора начинается с проверки выражения LV. Если логическое выражение имеет значение TRUE, то выполняется оператор S до тех пор, пока выражение LV не примет значение FALSE. В этом случае уравнение передаётся оператору, следующему за оператором S. Если же выражение LV принимает значение FALSE при первоначальной проверке, то оператор S не выполняется ни разу. При этом чтобы выйти из цикла, внутри него нужно изменить значения переменных, входящих в логическое выражение. Иначе оператор цикла будет выполняться бесконечное число раз.

Пример. Используя условия примера 1, программу запишем в виде:

```
PROGRAM SUMWHL;
VAR   X, Y: REAL;
   I, N: INTEGER;
BEGIN
WRITELN (‘ ВВЕСТИ N, X’);
READ (N, X);
Y:= 0;
I:=1;
          WHILE I <= N DO
BEGIN {открываем операторные скобки}
Y=Y+COS (I+SQR (X)/4);
I:=I+1;
END; {закрываем операторные скобки}
WRITELN (‘N =’, N,’  ‘,’ X = ‘,X);
WRITELN (‘Y =’,Y);
END.
```

Оператор цикла REPEAT также позволяет организовать цикл с неизвестным числом повторений. Такой оператор имеет вид:

REPEAT S
UNTIL LV

Здесь S — простой или составной оператор;

LV — логическое выражение.

Выполнение оператора REPEAT начинается с вычисления оператора S и продолжается до тех пор, пока не выполняется LV, т.е. когда логическое выражение примет значение FALSE (ложь). Из этого следует, что проверка LV проводится после каждой итерации и в случае принятия LV значения TRUE (правда) осуществляется выход из цикла. При использовании составного оператора S операторные скобки (BEGIN и END) не требуются. Кроме того, оператор, стоящий перед словом UNTIL, не имеет после себя точки с запятой.

Организация массивов

К наиболее часто встречающимся типовым вычислениям относятся: вычисление суммы и произведения элементов одномерного массива, нахождение наименьшего и наибольшего элементов одномерного массива.

Массив представляет собой набор конечного числа элементов одинакового типа. Объявление массивов происходит в разделе описания переменных следующим образом:

VAR <имя массива>: ARRAY [n..m] OF <тип элементов>,

где n — начальное значение индекса массива;

m — конечное значение индекса массива.

В учебном пособии будут использоваться массивы типа INTEGER (целые) и REAL (дробные). Элементы обозначаются именем массива и следующим за ним в квадратных скобках индексом.

Для ввода (вывода) элементов массива используются операторы READ (READLN) (WRITE (WRITELN)), записанные в цикле.

Например: FOR I: = 1 TO 10 DO

READ (A[I]);

Пример. Составить программу для вычисления сумм S элементов числовой последовательности A1, A2, …, A10 по формуле

S = A1+ A2+ … +A10.

Входными данными являются значения членов последовательности и число членов последовательности; выходными сумма членов последовательности.

Для решения задачи используется циклический алгоритм. Подготовка цикла заключается в задание начального значения суммы, равного 0.

В качестве параметра цикла берём номер члена последовательности.

Начальное значение параметра цикла равно i, конечное — числу членов последовательности, шаг цикла +1.

В теле цикла выполняется суммирование. Окончание цикла будет при значении параметра цикла, превышающего количество членов последовательности.

Числовую последовательность a1, a2, …, a10 обозначим как массив действительных чисел с именем — A.

Сумму членов — через S.

Размерность массива — n, параметр цикла — i.

Программа вычисления группы членов числовой последователь-ности имеет вид:

```
PROGRAM  PR 1;
CONST  N=10;     {РАЗМЕРНОСТЬ МАССИВА A}
VAR     A: ARRAY [1..N] OF REAL;       {ОПИСАНИЕ МАССИВА A}
S :  REAL ;              {ИНДЕНТИФИКАТОР СУММЫ}
I  :  INTEGER ;          {ПАРАМЕТР ЦИКЛА}
BEGIN
{ВВОД ИСХОДНЫХ ДАННЫХ}
WRITELN  ('ВВЕСТИ  10  ЭЛЕМЕНТОВ  МАССИВА  ЧЕРЕЗ ПРОБЕЛ');
FOR    I :  = 1  TO  N  DO
```

```
READ   (A [1] );
{ВЫЧИСЛЕНИЕ СУММЫ}
S: = 0;     {ПОДГОТОВКА ЦИКЛА}
FOR    I  :  = 1  TO  N  DO
S : = S + A [ I ] ;
{ПЕЧАТЬ}
WRITELN;            {ПРОПУСК СТРОКИ}
WRITELN       (‘‘  СУММА РАВНА ‘‘    ,  S :7 :3 );
WRITELN   ( ‘‘ ВЫЧИСЛЕНИЯ ЗАКОНЧЕНЫ ’’ ) ;
END.
```

Результат работы программы выглядит следующим образом:

Введите через пробел 10 элементов массива
10,4 5,5 7,1 3,4 85,1 35,43 6,8 21,2
32,4 5,7
сумма равна 217,030

Данная программа требует следующих пояснений. В разделе определения констант задана размерность массива A, который описан в разделе описания переменных. Тип индекса - ограниченный. Тип элементов REAL. В этом же разделе описываются переменные: I — параметр цикла; S — сумма.

Для ввода элементов массива A используется цикл с оператором FOR. В качестве параметра цикла вводят номер члена последовательности. Оператор WRITELN, стоящий перед оператором FOR, введён для организации диалога между пользователем и машиной. При выполнении программы этим оператором на экран выдаётся сообщение:

«введите 10 элементов массива через пробел».

Числа вводятся с клавиатуры через пробел. После набора десятого числа нажимается клавиша <ВК> (возврат каретки).

Для вычисления суммы членов последовательности используется цикл с оператором FOR. Начальное значение суммы задаётся оператором S: = 0. Очередное значение суммы вычисляется при выполнении оператора S := S + A[I]. Выход из цикла будет при значении параметра цикла I>10.

Для вывода на экран (печать) вычисленной суммы используется оператор WRITELN, в список переменных которого включена строка символов «значение суммы равно» для комментария выводимой информации и переменная S. Значение S выводится по формату 7 : 3.

Пример. Составить программу для вычисления суммы эле-

ментов последовательности целых чисел P1, P2, … , P10, имеющих четные индексы, и произведения элементов последовательности P1, P2, … ,P10 с нечетными индексами.

Сумма S членов последовательности с четными индексами и произведение Z членов последовательности с нечетными индексами вычисляются по формулам:

S = P2+P4+P6+P8+P10;

Z = P1*P3*P5*P7*P9.

Входными данными являются значения и число членов последовательности, выходными данными — сумма членов последовательности с четными индексами и произведение членов последовательности с нечетными индексами.

Для реализации данной задачи используем циклический алгоритм, рассмотренный в примере 1, но в этом случае кроме задания начального значения суммы нужно задать начальное значение произведения, равное 1. Параметр цикла должен меняться от 1 до 5.

Числовую последовательность P1, P2, … , P10 обозначим как массив целых чисел с именем P, сумму S — переменной SUM, произведение Z — переменной P, параметр цикла — переменной I.

Программа вычисление суммы и произведения элементов массива имеет вид:

```
PROGRAM PR 2;
CONST N = 10;{размерность массива}
VAR   P: ARRAY [1..N] OF INTEGER;{описание массива целых чисел}
{тип индекса – ограниченный}
SUM: INTEGER;{сумма элементов массива}
PRO: INTEGER; {произведение элементов массива}
I: INTEGER; {параметр цикла}
BEGIN {ввод исходных данных}
WRITELN ("вводите через пробел 10 элементов массива");
FOR I:= 1 TO N DO
READ (P[I]);
{вычисление суммы элементов с четными  индексами}
{произведение элементов с нечетными индексами}
SUM:= 0; {начальное значение суммы}
PRO:= 1; {начальное значение произведения}
FOR I:= 1 TO 5 DO
BEGIN
        SUM := SUM + P [I*2];
```

```
            PRO :=  PRO * P [I*2–1];
    END;
    {печать результатов}
    WRITELN;
    WRITELN ('сумма равна',SUM:5);
    WRITELN ('произведение равно',PRO:7);
    END.
```

Результат работы программы выглядит следующим образом:

Введите через пробел 10 элементов массива
1 5 9 7 12 3 6 4 5 9
сумма равна 28
произведение равно 3240

В разделе определения констант задается размерность массива N = 10. В разделе описания переменных описывается массив целых чисел, переменные SUM и PRO, параметр цикла I.

Ввод элементов осуществляется в простом цикле с оператором WRITELN ('введите через пробел 10 элементов массива').

При подготовке цикла задается начальное значение суммы оператором SUM: = 0; начальное значение произведение — оператором PRO: = 1.

Для вычисления суммы и произведения элементов числовой последовательности организован еще один простой цикл с оператором FOR. Параметром этого цикла является переменная I, значение которой меняется от 1 до 5 с шагом 1. В теле цикла вычисляются сумма и произведение элементов, значения индексов которых соответственно

(1*2), [1*(2–1)].

Тело цикла, состоящее из двух операторов, заключено в операторные скобки BEGIN … END.

Для вывода результатов вычисления на экран дисплея используются операторы WRITELN, в списке данных которых записываются текстовые сообщения в апострофах и переменные с указанием ширины поля вывода.

Пример. Составить программу определения наибольшего элемента числовой последовательности a_1, a_2, …, a_{15}. Вывести на печать найденный максимальный элемент.

Входными данными являются значения элементов числовой последовательности и число членов последовательности, выходными данными — номер максимального элемента последователь-

ности и значение этого элемента.

Для нахождения максимального элемента воспользуемся простым перебором значений последовательности, сравнивая эти значения с возможным максимумом. Если какое-либо значение элемента окажется больше найденного максимума, то это значение берется в качестве нового значения максимума и далее продолжается перебор. При таком переборе значений используется циклический алгоритм со структурой выбора действия по условию.

В качестве параметра цикла берется номер элемента числовой последовательности. Начальное значение параметра цикла 2, конечное – равно числу элементов в массиве, шаг цикла принимается равным 1.

В качестве начального значения возможного максимума берём значение первого элемента последовательности.

В теле цикла проверяется условие: будет ли очередное значение элемента последовательности больше значения максимума. Окончание цикла имеет место после просмотра всех элементов массива.

Введём обозначение: A — числовая последовательность a1, a2, … , a15; N — размерность массива; переменная AMAX — наибольший элемент; переменная PN — порядковый номер наибольшего элемента; переменная I — параметр цикла.

Программа вычисления наибольшего элемента последовательности имеет вид:

```
PROGRAM PR3;
CONST N=15;     {размерность массива A}
VAR
A: ARRAY [1..N] OF INTEGER; {описание массива A}
AMAX: INTEGER; {обозначение максимального элемента}
PN: INTEGER; {параметр цикла}
I: INTEGER;
BEGIN
WRITELN (‘введите через пробел 15 чисел’);
FOR I := 1 TO N DO
READLN (A[I]);
AMAX := A[1]; PN := 1; {подготовка цикла}
FOR I :=2 TO N DO  {цикл}
IF A[I] > AMAX THEN
BEGIN
AMAX := A[I];
```

```
PN := I;
END;
{печать}
WRITELN; {пропуск строки}
WRITELN ('максимальный элемент последовательности A');
WRITELN ('A (' , PN:2, ') = ' , AMAX:4 );
END.
```

Результат работы программы выглядит следующим образом:

Введите через пробел 15 чисел

12 78 65 3 5 76 82 93 13 28 54 68 26 47 3

Максимальный элемент последовательности A

A (8) = 93

В разделе определения констант задан размер массива N = 15. В разделе описания переменных объявлен массив с именем A. Тип индекса — ограниченный, тип элементов — INTEGER. В этом же разделе описаны переменные: AMAX — максимальный элемент; PN — порядковый номер максимального элемента в массиве; I — параметр цикла. Эти переменные описаны как переменные типа INTEGER.

Для ввода элементов массива используется цикл с оператором FOR. Оператором WRITELN ('введите через пробел 15 чисел'); объявляется диалоговый режим ввода чисел.

Подготовка цикла заключается в задании начальных значений переменных AMAX и PN. В качестве начального значения возможного максимума принимается значение первого элемента массива A, в качестве порядкового номера максимального элемента — значение – 1.

Поиск максимума начинается со сравнения второго элемента массива с первым, принятым в качестве максимума, поэтому в качестве начального значения параметра цикла взято I = 2.

В теле цикла значение очередного элемента сравнивается с возможным максимальным с помощью оператора IF … THEN. Если значение очередного элемента больше возможного максимального, то значение этого элемента присваивается переменной AMAX, а номер элемента — переменной PN. Если условие, записанное в операторе IF, не выполняется, то пропускается группа операторов: AMAX := A[I]; PN := [I].

Выполнение цикла заканчивается после просмотра всех элементов массива.

Для вывода на экран максимального элемента последова-

тельности используются операторы WRITELN, в список переменных которых включены строки символов для комментария выводимой информации и переменные PN и AMAX.

Пример. Задан массив вещественных чисел, состоящий из десяти элементов. Требуется ввести четные элементы этого массива. Программа имеет вид:

```
PROGRAM MAS;
VAR       I : INTEGER;
A : ARRAY [1..10] OF REAL;
BEGIN
WRITELN (‘ ВВЕСТИ МАССИВ (1..10) ’);
FOR I := 1 TO 10 DO;
READ ( A [I] );
I:=0;
REPEAT
I = I + 2;
WRITE (A[I] : 5 : 2);
UNTIL I>10;
END.
```

Работа с переменными строкового типа

Переменная строкового типа (String) может рассматриваться как массив элементов символьного типа (Char). Например, если в программе определены переменные

S: string; C: char; и задано S:='Москва', то S[1]='M', S[2]='o' и т. д. и возможно присвоение, например: C:= S[1]; Таким образом строка может рассматриваться как линейный массив символов. Элементы массива, составляющие строку можно переставлять местами и получать новые слова, например:

```
for i:= 1 to N div 2 do
begin
C:= S[i];
S[i]:= S[N-i+1];
S[N-i+1]:= C
end;
Writeln(S);
```

{ исходное слово выведется справа налево: "авксоМ" }

Здесь N:= ord(S[0]); - число символов в переменной "S" хранится в переменной S[0]. Функция "ord" преобразует символьный тип в целый. N div 2 - количество перестановок для слова из "N" символов. В переменной "C" запоминается значение

i-го элемента, который меняется с элементом, симметричным относительно середины строки.

Можно производить поиск и замену заданного символа в строке, например:

```
for i:=1 to N do
if S[i]='_' then writeln('найден символ пробел');
for i:=1 to N do
if S[i]='/' then S[i]:='\';  {замена символа "/" на "\"}
```

Заменяя или переставляя символы в строке по определенной схеме (закону) можно зашифровать строку. Для дешифровки используется, как правило, схема обратной перестановки или замены символов. Например:

```
for i:=1 to N do
S[i]:= chr(ord(S[i])+2);  {преобразование исходных символов в символы с кодом большим на две единицы}
```

Напомним, что все используемые в MS-DOS символы имеют ASCII коды от 0 до 255. Здесь удобно также использовать функции Pred(C); и Succ(C).

Двумерные массивы

Массивы, рассмотренные выше, имеют элементы, упорядоченные по одному индексу, и называются одномерными массивами или векторами. Массив может быть двумерным, трехмерным и т. д. Двумерные массивы имеют элементы, упорядоченные по двум индексам, и часто называются матрицами. В ABC Pascal при описании многомерного массива диапазоны изменения индексов перечисляются через запятые, например:

```
Var A: array[1..30, 1..7] of byte.
```

Работа с большими массивами

Поскольку суммарный размер всех переменных, описанных в программе, не может превышать длины сегмента (64 К), то использование массивов больших размеров вызывает определенные трудности. Опишем известный способ "разбиения" двумерного массива с использованием переменных типа ссылка.

```
program Big_Mas;
CONST N1= 30;  N2= 50;
type M1=  array [1 . . N1 ] of REAL;  { тип M1 - массив переменных вещественного типа}
M2= array[1..N2] of  ^M1;   { тип M2 - массив ссылок на начальные адреса элементов массивов типа M1}
var a1, a2: M2;   { двумерные массивы N1xN2 переменных вещественного типа }
i, j: word;
```

```
BEGIN
for i:=1 to N2 do New(a1[i]);{ размещение массива в оперативной памяти}
for i:=1 to N2 do New(a2[i]);
for j:= 1 to N1 do
for i:= 1 to N2 do  begin
a1[i]^[j]:= j + Sin(Pi*i/N2); { пример расчета значений }
a2[i]^[j]:= j - Cos(Pi*i/N2){элементов двумерных массивов}
end;
for i:= 1 to N2 do Dispose(a1[i]); { освобождение оперативной памяти }
for i:= 1 to N2 do Dispose(a2[i]);
Readln;
END.
```

Таким образом в оперативной памяти отводится место не под двумерные массивы "a1" и "a2" размером N1xN2, а под одномерные массивы (размером N2) адресов первых элементов линейных массивов (размером N1). Операция **a1[i]^[j] (a2[i]^[j])** называется разыменование переменной (элемента массива).

Процедуры и функции

Процедуры и функции представляют собой относительно самостоятельные фрагменты программы, оформленные особым образом и снабженные именем. Упоминание этого имени в тексте программы называется вызовом процедуры (функции). Отличие функции от процедуры заключается в том, что результатом исполнения операторов, образующих тело функции, всегда является некоторое единственное значение или указатель, поэтому обращение к функции можно использовать в соответствующих выражениях наряду с переменными и константами. Условимся далее называть процедуру или функцию общим именем «подпрограмма», если только для излагаемого материала указанное отличие не имеет значения.

Подпрограммы представляют собой инструмент, с помощью которого любая программа может быть разбита на ряд в известной степени независимых друг от друга частей. Такое разбиение необходимо по двум причинам.

Во-первых, это средство экономии памяти: каждая подпрограмма существует в программе в единственном экземпляре, в то время как обращаться к ней можно многократно из разных точек программы. При вызове подпрограммы активизируется последовательность образующих ее операторов, а с помощью передавае-

мых подпрограмме параметров нужным образом модифицируется реализуемый в ней алгоритм.

Вторая причина заключается в применении методики нисходящего проектирования программ. В этом случае алгоритм представляется в виде последовательности относительно крупных подпрограмм, реализующих более или менее самостоятельные смысловые части алгоритма. Подпрограммы в свою очередь могут разбиваться на менее крупные подпрограммы нижнего уровня и т.д. Последовательное структурирование программы продолжается до тех пор, пока реализуемые подпрограммами алгоритмы не станут настолько простыми, чтобы их можно было легко запрограммировать.

В этой главе подробно рассматриваются все аспекты использования подпрограмм в Паскале.

Вызов подпрограммы осуществляется простым упоминанием имени процедуры в операторе вызова процедуры или имени функции в выражении. При использовании расширенного синтаксиса Паскаля (см. ниже) функции можно вызывать точно так же, как и процедуры. Как известно, любое имя в программе должно быть обязательно описано перед тем как оно появится среди исполняемых операторов. Не делается исключения и в отношении подпрограмм: каждую свою процедуру и функцию программисту необходимо описать в разделе описаний.

Описать подпрограмму - это значит указать ее заголовок и тело. В заголовке объявляются имя подпрограммы и формальные параметры, если они есть. Для функции, кроме того, указывается тип возвращаемого ею результата. За заголовком следует тело подпрограммы, которое, подобно программе, состоит из раздела описаний и раздела исполняемых операторов. В разделе описаний подпрограммы могут встретиться описания подпрограмм низшего уровня, в тех - описания других подпрограмм и т.д.

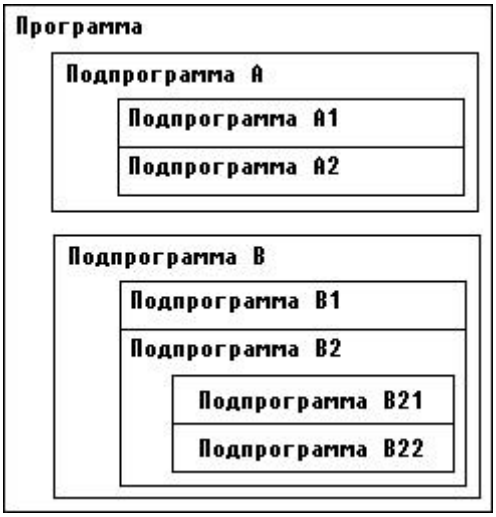


Рис.4. Пример структуры программы

Вот какую иерархию описаний получим, например, для программы, структура которой изображена на рис.4 (для простоты считается, что все подпрограммы представляют собой процедуры без параметров):

```
Program ...;
Procedure A;
Procedure A1;
.......
begin
.......
end {A1};
Procedure A2;
.......
begin
end {A2};
begin {A}
.......
end {A};
Procedure B;
Procedure B1;
.......
begin {B};
end
Procedure B2 ;
```

```
Procedure B21;
.......
```

и т.д.

Подпрограмма любого уровня имеет обычно множество имен констант, переменных, типов и вложенных в нее подпрограмм низшего уровня. Считается, что все имена, описанные внутри подпрограммы, локализуются в ней, т.е. они как бы «невидимы» снаружи подпрограммы. Таким образом, со стороны операторов, использующих обращение к подпрограмме, она трактуется как «черный ящик», в котором реализуется тот или иной алгоритм. Все детали этой реализации скрыты от глаз пользователя подпрограммы и потому недоступны ему. Например, в рассмотренном выше примере из основной программы можно обратиться к процедурам A и B, но нельзя вызвать ни одну из вложенных в них процедур A1, A2, B1 и т.д.

Сказанное относится не только к именам подпрограмм, но и вообще к любым именам, объявленным в них - типам, константам, переменным и меткам. Все имена в пределах подпрограммы, в которой они объявлены, должны быть уникальными и не могут совпадать с именем самой подпрограммы.

При входе в подпрограмму низшего уровня становятся доступными не только объявленные в ней имена, но и сохраняется доступ ко всем именам верхнего уровня. Образно говоря, любая подпрограмма как бы окружена полупрозрачными стенками: снаружи подпрограммы мы не видим ее внутренности, но, попав в подпрограмму, можем наблюдать все, что делается снаружи. Так, например, из подпрограммы B21 мы можем вызвать подпрограмму A, использовать имена, объявленные в основной программе, в подпрограммах B и B2, и даже обратиться к ним. Любая подпрограмма может, наконец, вызвать саму себя - такой способ вызова называется рекурсией.

Пусть имеем такое описание:

```
Program ..;
var V1 : ... ;
Procedure A;
var V2 :...;
.......
end {A};
Procedure B;
var V3 :...;
Procedure Bl;
var V4 :...;
```

```
Procedure B11;
var V5;
.....
```

Из процедуры B11 доступны все пять переменных V1,...,V5, из процедуры B1 доступны переменные V1,..., V4, из центральной программы - только V1.

При взаимодействии подпрограмм одного уровня иерархии вступает в силу основное правило Паскаля: любая подпрограмма перед ее использованием должна быть описана. Поэтому из подпрограммы B можно вызвать подпрограмму A, но из A вызвать B невозможно (точнее, такая возможность появляется только с использованием опережающего описания. Продолжая образное сравнение, подпрограмму южно уподобить ящику с непрозрачными стенками и дном и полупрозрачной крышей: из подпрограммы можно смотреть только «вверх» и нельзя «вниз», т.е. подпрограмме доступны только те объекты верхнего уровня, которые описаны до описания данной подпрограммы. Эти объекты называются глобальными по отношению к подпрограмме.

В отличие от стандартного Паскаля в ABC Pascal допускается произвольная последовательность описания констант, переменных, типов, меток и подпрограмм. Например, раздел VAR описания переменных может появляться в пределах раздела описаний одной и той же подпрограммы много раз и перемежаться с объявлениями других объектов и подпрограмм. Для ABC Pascal совершенно безразличен порядок следования и количество разделов VAR, CONST, TYPE, LABEL, но при определении области действия этих описаний следует помнить, что имена, описанные ниже по тексту программы, недоступны из ранее описанных подпрограмм, например:

```
var V1 : ...;
Procedure S;
var V2 : ...;
end {S};
var V3 :...;
.....
```

Из процедуры S можно обратиться к переменным V1 и V2, но нельзя использовать V3, так как описание V3 следует в программе за описанием процедуры S.

Имена, локализованные в подпрограмме, могут совпадать с ранее объявленными глобальными именами. В этом случае считается, что локальное имя «закрывает» глобальное и делает его недоступным, например:

```
var
i : Integer;
Procedure P;
var
i : Integer;
begin
writeln(i)
end {P};
begin
i := 1;
P
end.
```

Что напечатает эта программа? Все, что угодно: значение внутренней переменной I при входе в процедуру P не определено, хотя одноименная глобальная переменная имеет значение 1. Локальная переменная «закроет» глобальную и на экран будет выведено произвольное значение, содержащееся в неинициированной внутренней переменной. Если убрать описание

```
var
i : Integer;
```

из процедуры P, то на экран будет выведено значение глобальной переменной I, т.е. 1. Таким образом, одноименные глобальные и локальные переменные - это разные переменные. Любое обращение к таким переменным в теле подпрограммы трактуется как обращение к локальным переменным, т.е. глобальные переменные в этом случае попросту недоступны.

Описание подпрограммы состоит из заголовка и тела подпрограммы.

Заголовок

Заголовок процедуры имеет вид:

PROCEDURE <имя> [(<сп. ф. п . >)] ;

Заголовок функции:

FUNCTION <имя> [(<сп.ф.п.>)] : <тип>;

Здесь <имя> - имя подпрограммы (правильный идентификатор);

<сп.ф.п.> - список формальных параметров;

<тип> - тип возвращаемого функцией результата.

Сразу за заголовком подпрограммы может следовать одна из стандартных директив ASSEMBLER, EXTERNAL, FAR, FORWARD, INLINE, INTERRUPT, NEAR. Эти директивы уточняют действия компилятора и распространяются на всю подпрограмму и только на нее, т.е. если за подпрограммой следует другая подпрограмма,

стандартная директива, указанная за заголовком первой, не распространяется на вторую.

ASSEMBLER - эта директива отменяет стандартную последовательность машинных инструкций, вырабатываемых при входе в процедуру и перед выходом из нее. Тело подпрограммы в этом случае должно реализоваться с помощью команд встроенного ассемблера.

EXTERNAL - с помощью этой директивы объявляется внешняя подпрограмма.

FAR - компилятор должен создавать код подпрограммы, рассчитанный на дальнюю модель вызова. Директива NEAR заставит компилятор создать код, рассчитанный на ближнюю модель памяти. По умолчанию все подпрограммы, объявленные в интерфейсной части модулей, генерируются с расчетом на дальнюю модель вызова, а все остальные подпрограммы - на ближнюю модель.

В соответствии с архитектурой микропроцессора ПК, в программах могут использоваться две модели памяти: ближняя и дальняя. Модель памяти определяет возможность вызова процедуры из различных частей программы: если используется ближняя модель, вызов возможен только в пределах 64 Кбайт (в пределах одного сегмента кода, который выделяется основной программе и каждому используемому в ней модулю); при дальней модели вызов возможен из любого сегмента. Ближняя модель экономит один байт и несколько микросекунд на каждом вызове подпрограммы, поэтому стандартный режим компиляции предполагает эту модель памяти. Однако при передаче процедурных параметров, а также в оверлейных модулях соответствующие подпрограммы должны компилироваться с расчетом на универсальную - дальнюю - модель памяти, одинаково пригодную при любом расположении процедуры и вызывающей ее программы в памяти.

Явное объявление модели памяти стандартными директивами имеет более высокий приоритет по сравнению с опциями настройки среды ABC Pascal.

FORWARD - используется при опережающем описании для сообщения компилятору, что описание подпрограммы следует где-то дальше по тексту программы (но в пределах текущего программного модуля).

INLINE - указывает на то, что тело подпрограммы реализуется с помощью встроенных машинных инструкций.

INTERRUPT - используется при создании процедур обработки прерываний.

Список формальных параметров необязателен и может отсутствовать. Если же он есть, то в нем должны быть перечислены имена формальных параметров и их типы, например:

Procedure SB(a: Real; b: Integer; c: Char);

Как видно из примера, параметры в списке отделяются друг от друга точками с запятой. Несколько следующих подряд однотипных параметров можно объединять в подсписки, например, вместо

Function F(a: Real; b: Real): Real;

можно написать проще:

Function F(a,b: Real): Real;

Операторы тела подпрограммы рассматривают список формальных параметров как своеобразное расширение раздела описаний: все переменные из этого списка могут использоваться в любых выражениях внутри подпрограммы. Таким способом осуществляется настройка алгоритма подпрограммы на конкретную задачу.

Рассмотрим следующий пример. В языке Паскаль нет операции возведения в степень, однако с помощью встроенных функций LN(X) и EXP(X) нетрудно реализовать новую функцию с именем, например, POWER, осуществляющую возведение любого вещественного числа в любую вещественную степень. В программе (пример) вводится пара чисел X и Y и выводится на экран дисплея результат возведения X сначала в степень +Y, а затем - в степень -Y. Для выхода из программы нужно ввести Ctrl-Z и Enter.

Пример

```
var
x,y:Real;
Function Power (a, b : Real):
Real;
begin {Power}
if a > 0 then
Power := exp(b * In (a))
else if a < 0 then
Power := exp(b * ln(abs(a))
else if b = 0 then
Power := 1
else
Power := 0
end {Power} ;
{--------------------}
```

```
begin {main}
repeat
readln(x,y) ;
writeln (Power (x,y) :12:10, Power (x, -y) : 15 : 10)
until EOF
end {main} .
```

Для вызова функции POWER мы просто указали ее в качестве параметра при обращении к встроенной процедуре WRITELN. Параметры X и Y в момент обращения к функции - это фактические параметры. Они подставляются вместо формальных параметров A и B в заголовке функции и затем над ними осуществляются нужные действия. Полученный результат присваивается идентификатору функции - именно он и будет возвращен как значение функции при выходе из нее. В программе функция POWER вызывается дважды - сначала с параметрами X и Y, а затем X и -Y, поэтому будут получены два разных результата.

Механизм замены формальных параметров на фактические позволяет нужным образом настроить алгоритм, реализованный в подпрограмме. ABC Pascal следит за тем, чтобы количество и тип формальных параметров строго соответствовали количеству и типам фактических параметров в момент обращения к подпрограмме. Смысл используемых фактических параметров зависит от того, в каком порядке они перечислены при вызове подпрограммы. В примере первый по порядку фактический параметр будет возводиться в степень, задаваемую вторым параметром, а не наоборот. Пользователь должен сам следить за правильным порядком перечисления фактических параметров при обращении к подпрограмме. Любой из формальных параметров подпрограммы может быть либо параметром-значением, либо параметром-переменной, либо, наконец, параметром-константой. В предыдущем примере параметры A и B определены как параметры-значения. Если параметры определяются как параметры-переменные, перед ними необходимо ставить зарезервированное слово VAR, а если это параметры-константы,- слово CONST, например:

```
Procedure MyProcedure (var a: Real; b: Real; const c: String);
```

Здесь A - параметр-переменная, B -параметр-значение, а C - параметр-константа.

Определение формального параметра тем или иным способом существенно, в основном, только для вызывающей программы: если формальный параметр объявлен как параметр-переменная, то при вызове подпрограммы ему должен соответ-

ствовать фактический параметр в виде переменной нужного типа; если формальный параметр объявлен как параметр-значение или параметр-константа, то при вызове ему может соответствовать произвольное выражение. Контроль за неукоснительным соблюдением этого правила осуществляется компилятором Паскаля. Если бы для предыдущего примера был использован такой заголовок функции:

Function Power (var a, b : Real) : Real;

то при втором обращении к функции компилятор указал бы на несоответствие типа фактических и формальных параметров (параметр –У есть выражение, в то время как соответствующий ему формальный параметр описан как переменная).

Для того чтобы понять, в каких случаях использовать тот или иной тип параметров, рассмотрим, как осуществляется замена формальных параметров на фактические в момент обращения к подпрограмме.

Если параметр определен как значение, то перед вызовом подпрограммы это значение вычисляется, полученный результат копируется во временную память и передается подпрограмме. Важно учесть, что даже если в качестве фактического параметра указано простейшее выражение в виде переменной или константы, все равно подпрограмме будет передана лишь копия переменной (константы). Любые возможные изменения в подпрограмме значения никак не воспринимаются вызывающей программой, так как в этом случае изменяется копия фактического параметра.

Если параметр определен как переменная, то при вызове подпрограммы передается сама переменная, а не ее копия (фактически в этом случае подпрограмме передается адрес переменной). Изменение переменной приводит к изменению самого фактического параметра в вызывающей программе.

В случае вызова константы в подпрограмму также передается адрес области памяти, в которой располагается переменная или вычисленное значение. Однако компилятор блокирует любые присваивания константе нового значения в теле подпрограммы.

Представленный ниже пример 2 поясняет изложенное. В программе задаются два целых числа 5 и 7, эти числа передаются процедуре INC2, в которой они удваиваются. Один из параметров передается как переменная, другой - как значение. Значения параметров до и после вызова процедуры, а также результат их удвоения выводятся на экран.

Пример

a : Integer = 5;

```
b : Integer = 7 ;
{----------------}
Procedure Inc2 (var c: Integer; b: Integer) ;
begin {Inc2}
c := c + c;
b := b + b;
WriteLn ( 'удвоенные: ', c:5, b:5)
end {inc2};
{--------------}
begin {main}
WriteLn('исходные: ', a:5, b:5);
Inc2(a,b);
WriteLn('результат: ', a:5, b:5)
end {main}.
```

В результате прогона программы будет выведено:

```
исходные:   5  7
удвоенные: 10 14
результат: 10  7
```

Как видно из примера, удвоение второго формального параметра в процедуре INC2 не вызвало изменения фактической переменной B, так как этот параметр описан в заголовке процедуры как значение. Этот пример может служить еще и иллюстрацией механизма «накрывания» глобальной переменной одноименной локальной: хотя переменная B объявлена как глобальная (она описана в вызывающей программе перед описанием процедуры), в теле процедуры ее «закрыла» локальная переменная B, объявленная как значение.

Итак, переменные используются как средство связи алгоритма, реализованного в подпрограмме, с внешним миром: с помощью этих параметров подпрограмма может передавать результаты своей работы вызывающей программе. Разумеется, в распоряжении программиста всегда есть и другой способ передачи результатов - через глобальные переменные. Однако злоупотребление глобальными связями делает программу, как правило, запутанной, трудной в понимании и сложной в отладке. В соответствии с требованиями хорошего стиля программирования рекомендуется там, где это возможно, использовать передачу результатов через фактические переменные.

С другой стороны, описание всех формальных параметров как переменных нежелательно по двум причинам. Во-первых, это исключает возможность вызова подпрограммы с фактическими параметрами в виде выражений, что делает программу менее

компактной. Во-вторых, и главных, в подпрограмме возможно случайное использование формального параметра, например, для временного хранения промежуточного результата, т.е. всегда существует опасность непреднамеренно испортить фактическую переменную. Вот почему переменными следует объявлять только те, через которые подпрограмма в действительности передает результаты вызывающей программе. Чем меньше параметров объявлено переменными и чем меньше в подпрограмме используется глобальных переменных, тем меньше опасность получения непредусмотренных программистом побочных эффектов, связанных с вызовом подпрограммы, тем проще программа в понимании и отладке. По той же причине не рекомендуется использовать переменные в заголовке функции: если результатом работы функции не может быть единственное значение, то логичнее использовать процедуру или нужным образом декомпозировать алгоритм на несколько подпрограмм.

Существует еще одно обстоятельство, которое следует учитывать при выборе вида формальных параметров. Как уже говорилось, при объявлении параметра-значения осуществляется копирование фактического параметра во временную память. Если этим параметром будет массив большой размерности, то существенные затраты времени и памяти на копирование при многократных обращениях к подпрограмме можно минимизировать, объявив этот параметр константой. Параметр-константа не копируется во временную область памяти, что сокращает затраты времени на вызов подпрограммы, однако любые его изменения в теле подпрограммы невозможны - за этим строго следит компилятор.

Может сложиться впечатление, что объявление переменных в списке формальных параметров подпрограммы ничем не отличается от объявления их в разделе описания переменных. Действительно, в обоих случаях много общего, но есть одно существенное различие: типом любого параметра в списке формальных параметров может быть только стандартный или ранее объявленный тип. Поэтому нельзя, например, объявить следующую процедуру:

Procedure S (a: array [1..10] of Real);

так как в списке формальных параметров фактически объявляется тип-диапазон, указывающий границы индексов массива.

Если мы хотим передать какой-то элемент массива, то проблем, как правило, не возникает, но если в подпрограмму передается весь массив, то следует первоначально описать его тип.

Например:

```
type
atype = array [1..10]of Real;
Procedure S (a: atype);
.......
```

Поскольку строка является фактически своеобразным массивом, ее передача в подпрограмму осуществляется аналогичным образом:

```
type
intype = String [15] ;
outype = String [30] ;
Function St (s : intype): outype;
.......
```

Требование описать любой тип-массив или тип-строку перед объявлением подпрограммы на первый взгляд кажется несущественным. Действительно, в рамках простейших вычислительных задач обычно заранее известна структура всех используемых в программе данных, поэтому статическое описание массивов не вызывает проблем. Однако разработка программных средств универсального назначения связана со значительными трудностями. По существу, речь идет о том, что в Паскале невозможно использовать в подпрограммах массивы с «плавающими» границами изменения индексов. Например, если разработана программа, обрабатывающая матрицу 10x10 элементов, то для обработки матрицы 9x11 элементов необходимо переопределить тип, т.е. перекомпилировать всю программу (речь идет не о динамическом размещении массивов в куче, а о статическом описании массивов и передаче их как параметров в подпрограммы). Этот недостаток, как и отсутствие в языке средств обработки исключительных ситуаций (прерываний), унаследован из стандартного Паскаля и представляет собой объект постоянной и вполне заслуженной его критики. Разработчики ABC Pascal не рискнули кардинально изменить свойства базового языка, но, тем не менее, включили в него некоторые средства, позволяющие в известной степени смягчить отмеченные недостатки. Эти недостатки практически полностью устранены в языке Object Pascal, используемом в визуальной среде программирования Delphi.

Прежде всего, в среде ABC Pascal можно установить режим компиляции, при котором отключается контроль за совпадением фактической и формальной длины строки. Это позволяет легко решить вопрос о передаче подпрограмме строки произвольной длины. При передаче строки меньшего размера формальный па-

раметр будет иметь ту же длину, что и параметр обращения; передача строки большего размера приведет к ее усечению до максимального размера формального параметра. Следует сказать, что контроль включается только при передаче строки, объявленной как формальный параметр-переменная. Если, соответствующий параметр объявлен некоторым значением, эта опция игнорируется и длина не контролируется.

Значительно сложнее обстоит дело с передачей массивов произвольной длины. Наиболее универсальным приемом в этом случае будет, судя по всему, работа с указателями и использование индексной арифметики. Несколько проще можно решить эту проблему при помощи нетипизированных параметров (см. ниже). В версии ABC Pascal язык поддерживает так называемые открытые массивы, легко решающие проблему передачи подпрограмме одномерных массивов переменной длины.

Открытый массив представляет собой формальный параметр подпрограммы, описывающий базовый тип элементов массива, но не определяющий его размерности и границы:

```
Procedure MyProc(OpenArray: array of Integer);
```

Внутри подпрограммы такой параметр трактуется как одномерный массив с нулевой нижней границей. Верхняя граница открытого массива возвращается функцией HIGH. Используя 0 как минимальный индекс и значение, возвращаемое функцией HIGH, как максимальный индекс, подпрограмма может обрабатывать одномерные массивы произвольной длины:

```
{Иллюстрация использования открытых массивов: программа выводит на экран содержимое двух одномерных массивов разной длины с помощью одной процедуры ArrayPrint}
Procedure  ArrayPrint(aArray: array of Integer);
var
k: Integer;
begin
for k := 0 to High(aArray) do
Write(aArray[k]:8);
WriteLn
end;
const
A:array [-1..2] of Integer = (0,1,2,3);
B: array [5..7] of Integer = (4,5,6);
begin
ArrayPrint(A);
ArrayPrint(B)
```

```
end.
```

Как видно из этого примера, фактические границы массивов A и B, передаваемых в качестве параметров вызова процедуре ArrayPrint, не имеют значения. Однако размерность открытых массивов (количество индексов) всегда равна 1 - за этим следит компилятор. Если бы, например, мы добавили в программу двумерный массив C:

```
var
C: array [1..3,1..5] of Integer;
```

то обращение

```
ArrayPrint(C)
```

вызывало бы сообщение об ошибке

```
Error26: Type mismatch.
```

(Ошибка 26: Несоответствие типов.)

Процедурные типы.

Процедуры и функции.

Процедурные типы - это нововведение фирмы Borland (в стандартном Паскале таких типов нет). Основное назначение этих типов - дать программисту гибкие средства передачи функций и процедур в качестве фактических параметров обращения к другим процедурам и функциям.

Для объявления процедурного типа используется заголовок процедуры (функции), в котором опускается ее имя, например:

```
type
Prod = Procedure (a, b, c: Real; var d: Real);
Proc2 = Procedure (var a, b) ;
Proc3 = Procedure;
Func1 = Function: String;
Func2 = Function (var s: String): Real;
```

Как видно из приведенных примеров, существует два процедурных типа: тип-процедура и тип-функция.

Пример 3 иллюстрирует механизм передачи процедур в качестве фактических параметров вызова. Программа выводит на экран таблицу двух функций:

```
sin1(x) = (sin(x) + 1) * exp(-x)
cos1(x) = (cos(x) + 1) * exp(-x).
```

Вычисление и печать значений этих функций реализуются в процедуре PRINTFUNC, которой в качестве параметров передаются номер позиции N на экране, куда будет выводиться очередной результат (с помощью этого параметра реализуется вывод в две колонки), и имя нужной функции.

Пример.

```
Uses CRT;
type
Func = Function (x: Real) : Real;
{----------------}
Procedure PrintFunc (XPos: Byte; F:Func) ;
{Осуществляет печать функции F . (XPos - горизонтальная позиция начала вывода) }
const
np = 20; {Количество вычислений функций}
var
x : Real; i : Integer;
begin {PrintFunc}
for i := 1 to np do
begin
x := i * (2 * pi / np) ;
GotoXY (XPos, WhereY) ;
WriteLn (x:5:3, F(x):18:5)
end
end; {PrintFunc}
{-----------------}
Function Sin1fx: Real): Real; far;
begin
sinl := (sin(x) + 1) * exp(-x)
end;
Function Cos1(x: Real): Real; far;
begin
cosl := (cos(x) + 1) * exp(-x)
end;
{---------------}
begin {основная программа}
ClrScr; {Очищаем экран}
PrintFunc (1, sin1); GotoXY (1,1); {Переводим курсор в левый верхний угол}
PrintFunc (40, cos1)
end.
```

Обратите внимание: для установления правильных связей функций SIN1 и COS1 с процедурой PRINTFUNC они должны компилироваться с расчетом на модуль памяти. Вот почему в программу вставлены стандартные директивы FAR сразу за заголовками функций. В таком режиме должны компилироваться любые процедуры (функции), которые будут передаваться в качестве фактических параметров вызова.

Стандартные процедуры (функции) ABC Pascal не могут передаваться рассмотренным способом.

В программе могут быть объявлены переменные процедурных типов, например, так:

```
var
p1 : Proc1;
f1, f2 : Func2;
p : array [1..N] of Proc1;
```

Переменным процедурных типов допускается присваивать в качестве значений имена соответствующих подпрограмм. После присваивания имя переменной становится синонимом имени подпрограммы, например:

```
type
Proc = Procedure (n: word; var a: Byte);
var
ProcVar: Proc; x, y : Byte;
Procedure Procl(x: word; var y: Byte); far;
begin
if x > 255 then
y := x mod 255
else
y := Byte(x)
end;
begin {Главная программа}
ProcVar := Proc1;
for x := 150 to' 180 do
begin
ProcVar (x + 100, y);
Write (y:8)
end
end.
```

Разумеется, такого рода присваивания допустимы и для функций, например:

```
type
FuncType = Function (i : Integer) : Integer;
var
VarFunc : FuncType;
i : Integer;
Function MyFunc (count : Integer) : Integer; far;
begin
.....
end; {MyFunc}
```

```
begin {Основная программа}
.......
i := MyFunc(1); {Обычное использование результата функции}
.......
VarFunc := MyFunc;
{Присваивание переменной процедурного типа имени функции MyFunc}
.......
end.
```

Отметим, что присваивание

```
VarFunc := MyFunc(1);
```

будет недопустимым, так как слева и справа от знака присваивания используются несовместимые типы: слева - процедурный тип, а справа - INTEGER; имя функции со списком фактических параметров MyFunc(1) трактуется ABC Pascal как обращение к значению функции, в то время как имя функции без списка параметров рассматривается как имя функции.

В отличие от стандартного Паскаля, в ABC Pascal разрешается использовать в передаваемой процедуре (функции) любые типы параметров: значения, переменные, константы (в стандартном Паскале только значения).

Нетипизированные переменные

Еще одно очень полезное нововведение фирмы Borland - возможность использования нетипизированных параметров. Параметр считается нетипизированным, если тип формального параметра-переменной в заголовке подпрограммы не указан, при этом соответствующий ему фактический параметр может быть переменной любого типа. Заметим, что нетипизированными могут быть только переменные.

Нетипизированные параметры обычно используются в случае, когда тип данных несущественен. Такие ситуации чаще всего возникают при разного рода копированиях одной области памяти в другую, например, с помощью процедур BLOCKREAD, BLOCKWRITE, MOVE и т.п.

Нетипизированные параметры в сочетании с механизмом совмещения данных в памяти можно использовать для передачи подпрограмме одномерных массивов переменной длины.

В примере функция NORMA вычисляет норму вектора, длина которого меняется случайным образом. Стандартная константа MAXINT содержит максимальное значение целого типа INTEGER и равна 32767.

Следует учесть, что при обращении к функции NORMA массив X помещается в стек и передается по ссылке, поэтому описание локальной переменной A в виде одномерного массива максимально возможной длины в 65532 байта (встроенная константа MAXINT определяет максимально возможное значение типа INTEGER и равна 32767), совпадающего с X, на самом деле не приведет к выделению дополнительного объема памяти под размещение этой переменной. Иными словами, переменная A - фиктивная переменная, размер которой никак не влияет на объем используемой памяти. С таким же успехом можно было бы объявить ее в виде массива из одного элемента, правда, в этом случае необходимо позаботиться об отключении контроля выхода индекса за границы диапазона.

Пример

```
const
NN = 100; {Максимальная длина вектора}
var
a : array [1..NN] of Real;
i, j, N : Integer;
{---------------}
Function Norma (var x; N: Integer) : Real;
var
a : array [1..2*MaxInt div SizeOf (Real) ] of Real absolute x;
i : Integer;
s : Real;
begin {Norma}
s := 0;
for i := 1 to N do
s := s + sqr (a [i] ) ;
Norma := sqrt(s)
end {Norma} ;
{------------------}
begin {main}
for i := 1 to 10 do
begin
N := Random (NN) + 1; {Текущая длина вектора}
for j := 1 to N do
a [ j ] : = Random ;
WriteLn ('N = ', N:2,норма=',Norma(a, N):10:7)
end
end {main} .
```

Как видно из рассмотренного примера, передача одномерных массивов переменной длины не вызывает никаких трудностей. Сложнее обстоит дело с многомерными массивами, однако и в этом случае использование описанного приема (нетипизированный параметр и совмещение его в памяти с фиктивной переменной).

Рекурсия и опережающее описание

Рекурсия - это такой способ организации вычислительного процесса, при котором подпрограмма в ходе выполнения составляющих ее операторов обращается сама к себе.

Рассмотрим классический пример - вычисление факториала. Программа вводит с клавиатуры целое число N и выводит на экран значение N!, которое вычисляется с помощью рекурсивной функции PAC. Для выхода из программы необходимо либо ввести достаточно большое целое число, чтобы вызвать переполнение при умножении чисел с плавающей запятой, либо нажать Ctrl-Z и Enter.

При выполнении правильно организованной рекурсивной подпрограммы осуществляется многократный переход от некоторого текущего уровня организации алгоритма к нижнему уровню последовательно до тех пор, пока, наконец, не будет получено тривиальное решение поставленной задачи. В примере решение при N = 0 тривиально и используется для остановки рекурсии.

Пример

```
Program Factorial;
{$S+} {Включаем контроль переполнения стека}
var
n: Integer;
Function Facfn: Integer): Real;
{Рекурсивная функция, вычисляющая n ! }
begin {Fac}
 if n < 0 then
WriteLn ('Ошибка в задании N')
else
if n = 0 then
Fac := 1
else Fac := n * Fac(n-l)
end {Fac} ;
{---------------}
begin {main} repeat
  ReadLn (n) ;
WriteLn ('n!= ',Fac(n))
```

```
until EOF
end {main} .
```

Рекурсивная форма организации алгоритма обычно выглядит изящнее итерационной и дает более компактный текст программы, но при выполнении, как правило, медленнее и может вызвать переполнение стека (при каждом входе в подпрограмму ее локальные переменные размещаются в особым образом организованной области памяти, называемой программным стеком). Переполнение стека особенно ощутимо сказывается при работе с сопроцессором: если программа использует арифметический сопроцессор, результат любой вещественной функции возвращается через аппаратный стек сопроцессора, рассчитанный всего на 8 уровней. Если, например, попытаться заменить тип REAL функции FAC (см. пример) на EXTENDED, программа перестанет работать уже при N = 8. Чтобы избежать переполнения стека сопроцессора, следует размещать промежуточные результаты во вспомогательной переменной. Вот правильный вариант примера для работы с типом EXTENDED:

```
Program Factorial;
{$S+,N+,E+} {Включаем контроль Стека и работу сопроцессора}
var
n: Integer;
Function Fac(n: Integer): extended;
var
F: extended; {Буферная переменная для разгрузки стека сопроцессора}
{Рекурсивная функция, вычисляющая n! }
begin {Pac}
if n < 0 then
WriteLn ('Ошибка в задании N') else
if n = 0 then
Fac := 1 else begin
F := Fac(n-l) ; Fac := F * n end end {Fac} ;
{--------------}
begin {main}
repeat
ReadLn (n) ;
WriteLn ('n! = ',Fac(n))
until EOF
end {main} .
```

Рекурсивный вызов может быть косвенным. В этом случае

подпрограмма обращается к себе опосредованно, путем вызова другой подпрограммы, в которой содержится обращение к первой, например:

```
Procedure A (i : Byte) ;
begin
.......
B (i);
.......
end ;
Procedure B (j : Byte) ;
.......
begin
.......
A(j);
.......
end;
```

Если строго следовать правилу, согласно которому каждый идентификатор перед употреблением должен быть описан, то такую программную конструкцию использовать нельзя. Для того, чтобы такого рода вызовы стали возможны, вводится опережающее описание:

```
Procedure B(j : Byte); forward;
Procedure A(i : Byte);
begin
.......
B (i) ;
.......
end ;
Procedure B;
begin
.......
A(j);
.......
end;
```

Как видим, опережающее описание заключается в том, что объявляется лишь заголовок процедуры B, а ее тело заменяется стандартной директивой FORWARD. Теперь в процедуре A можно использовать обращение к процедуре B - ведь она уже описана, точнее, известны ее формальные параметры, и компилятор может правильным образом организовать ее вызов. Обратите внимание: тело процедуры B начинается заголовком, в котором уже не указываются описанные ранее формальные параметры.

13. МОДУЛИ

Модуль имеет следующую структуру:

UNIT <имя>;
INTERFACE
<интерфейсная часть>
IMPLEMENTATION
<исполняемая часть>
BEGIN
<инициирующая часть>
END.

Здесь UNIT - зарезервированное слово (единица); начинает заголовок модуля; <имя> - имя модуля (правильный идентификатор); INTERFACE - зарезервированное слово (интерфейс); начинает интерфейсную часть модуля;

IMPLEMENTATION - зарезервированное слово (выполнение); начинает исполняемую часть;

BEGIN - зарезервированное слово; начинает инициирующую часть модуля;

конструкция BEGIN Инициирующая частъ> необязательна;

END - зарезервированное слово - признак конца модуля.

Таким образом, модуль состоит из заголовка и трех составных частей, любая из которых может быть пустой.

Заголовок модуля состоит из зарезервированного слова UNIT и следующего за ним имени модуля. Для правильной работы среды ABC Pascal и возможности подключения средств, облегчающих разработку крупных программ, это имя должно совпадать с именем дискового файла, в который помещается исходный текст модуля. Если, например, имеем заголовок

Unit Global;

то исходный текст соответствующего модуля должен размещаться в дисковом файле GLOBAL.PAS. Имя модуля служит для его связи с другими модулями и основной программой. Эта связь устанавливается специальным предложением

USES <сп.модулей>

Здесь USES - зарезервированное слово {использует);

<сп.модулей> - список модулей, с которыми устанавливается связь; элементами списка являются имена модулей, отделяемые друг от друга запятыми, например:

Uses CRT, Graph, Global;

Если объявление USES... используется, оно должно открывать раздел описаний основной программы. Модули могут исполь-

зовать другие модули. Предложение USES в модулях может следовать либо сразу за зарезервированным словом INTERFACE, либо сразу за словом IMPLEMENTATION, либо, наконец, и там, и там (т.е. допускаются два предложения USES).

Интерфейсная часть открывается зарезервированным словом INTERFACE. В этой части содержатся объявления всех глобальных объектов модуля (типов, констант, переменных и подпрограмм), которые должны стать доступными основной программе и/или другим модулям. При объявлении глобальных подпрограмм в интерфейсной части указывается только их заголовок, например:

```
Unit Cmplx;
Interface
type
complex = record
re, im : real
end;
Procedure AddC (x, y : complex; var z : complex);
Procedure MulC (x, y : complex; var z : complex);
```

Если теперь в основной программе написать предложение

```
Uses Cmplx;
```

то в программе станут доступными тип COMPLEX и две процедуры - ADDC и MULC из мдуля CMPLX

Отметим, что объявление подпрограмм в интерфейсной части автоматически сопровождается их компиляцией с использованием дальней модели памяти. Таким образом обеспечивается доступ к подпрограммам из основной программы и других модулей. Следует учесть, что все константы и переменные, объявленные в интерфейсной части модуля, равно как и глобальные константы и переменные основной программы, помещаются компилятором ABC Pascal в общий сегмент данных (максимальная длина сегмента 65536 байт). Порядок появления различных разделов объявлений и их количество может быть произвольным. Если в интерфейсной части объявляются внешние подпрограммы или подпрограммы в машинных кодах, их тела (т.е. зарезервированное слово EXTERNAL, в первом случае, и машинные коды вместе со словом INLINE - во втором) должны следовать сразу за их заголовками в исполняемой части модуля (не в интерфейсной!). В интерфейсной части модулей нельзя использовать опережающее описание.

Исполняемая часть начинается зарезервированным словом IMPLEMENTATION и содержит описания подпрограмм, объявленных в интерфейсной части. В ней могут объявляться локальные

для модуля объекты - вспомогательные типы, константы, переменные и блоки, а также метки, если они используются в инициирующей части.

Описанию подпрограммы, объявленной в интерфейсной части модуля, в исполняемой части должен предшествовать заголовок, в котором можно опускать список формальных переменных (и тип результата для функции), так как они уже описаны в интерфейсной части. Но если заголовок подпрограммы приводится в полном виде, т.е. со списком формальных параметров и объявлением результата, он должен совпадать с заголовком, объявленным в интерфейсной части, например:

```
Unit Cmplx;
Interface
type
complex = record
re, im : real
end;
Procedure AddC (x, y : complex; var z : complex);
Implementation
Procedure AddC;
begin
z.re := x.re +Y.re;
z.im := x.im +y.im
end;
end.
```

Локальные переменные и константы, а также все программные коды, порожденные при компиляции модуля, помещаются в общий сегмент памяти.

Инициирующая часть завершает модуль. Она может отсутствовать вместе с начинающим ее словом BEGIN или быть пустой - тогда за BEGIN сразу следует признак конца модуля (слово END и следующая за ним точка).

В инициирующей части размещаются исполняемые операторы, содержащие некоторый фрагмент программы. Эти операторы исполняются до передачи управления основной программе и обычно используются для подготовки ее работы. Например, в них могут инициироваться переменные, открываться нужные файлы, устанавливаться связи с другими ПК по коммуникационным каналам и т.п.:

```
Unit FileText;
Interface
Procedure Print(s : string);
```

```
Implementation
var
f: text; const
name = 'output.txt'; Procedure Print;
begin
WriteLn(f, s)
end;
{ Начало инициирующей части: }
begin
assign(f, name);
rewrite(f);
{ Конец инициирующей части }
end.
```

Не рекомендуется делать инициирующую часть пустой, лучше ее опустить: пустая часть содержит пустой оператор, которому будет передано управление при запуске программы. Это часто вызывает проблемы при разработке оверлейных программ.

В среде ABC Pascal имеются средства, управляющие способом компиляции модулей и облегчающие разработку крупных программных проектов. В частности, определены три режима компиляции: COMPILE, MAKE и BUILD. Режимы отличаются только способом связи компилируемого модуля или основной программы с другими модулями, объявленными в предложении USES,

При компиляции модуля или основной программы в режиме COMPILE все упоминающиеся в предложении USES модули должны быть предварительно откомпилированы и результаты компиляции помещены в одноименные файлы с расширением TPU. Например, если в программе (модуле) имеется предложение

Uses Global;

то на диске в каталоге, объявленном опцией UNIT DIRECTORIES, уже должен находиться файл GLOBAL.TPU. Файл с расширением TPU (от англ. Turbo Pascal Unit) создается автоматически в результате компиляции модуля (если основная программа может компилироваться без создания исполняемого EXE-файла, то компиляция модуля всегда приводит к созданию TPU-файла).

В режиме MAKE компилятор проверяет наличие TPU-файлов для каждого объявленного модуля. Если какой-либо из файлов не обнаружен, система пытается отыскать одноименный файл с расширением PAS, т.е. файл с исходным текстом модуля, и, если искомый файл найден, приступает к его компиляции. Кроме того, в этом режиме система следит за возможными изменениями исходного текста любого используемого модуля. Если в PAS-файл (ис-

ходный текст модуля) внесены какие-либо изменения, то независимо от того, есть ли уже в каталоге соответствующий TPU-файл или нет, система осуществляет его компиляцию перед компиляцией основной программы. Более того, если изменения внесены в интерфейсную часть модуля, то будут перекомпилированы также и все другие модули, обращающиеся к нему. Режим MAKE, таким образом, существенно облегчает процесс разработки крупных программ с множеством модулей: программист избавляется от необходимости следить за соответствием существующих TPU-файлов их исходному тексту, так как система делает это автоматически.

В режиме BUILD существующие TPU-файлы игнорируются, и система пытается отыскать (и компилировать) соответствующий PAS-файл для каждого объявленного в предложении USES модуля. После компиляции в режиме BUILD программист может быть уверен в том, что учтены все сделанные им изменения в любом из модулей.

Подключение модулей к основной программе и их возможная компиляция осуществляются в порядке их объявления в предложении USES. При переходе к очередному модулю система предварительно отыскивает все модули, на которые он ссылается. Ссылки модулей друг на друга могут образовывать древовидную структуру любой сложности, однако запрещается явное или косвенное обращение модуля к самому себе. Например, недопустимы следующие объявления:

```
Unit A;              Unit B;
Interface            Interface
Uses B;              Uses A;
.......              .......
Implementation       Implementation
.......              .......
end.                 end.
```

Это ограничение можно обойти, если «спрятать» предложение USES в исполняемые части зависимых модулей:

```
Unit A;              Unit B;
Interface            Interface
.......              .......
Implementation       Implementation
Uses B;              Uses A;
......               .......
end.                 end.
```

Например, в следующей программе (пример) осуществля-

ются четыре арифметические операции над парой комплексных чисел.

Пример

```
Uses Cmplx;
var
a, b, c : complex;
begin
a.re := 1; a.im := 1;
b.re := 1; b.im := 2;
AddC(a, b, c);
WriteLn('Сложение: 'c.re:5:1, c.im:5:1,'i') ;
SubC(a, b, c) ;
WriteLn('Вычитание: 'c.re:5:1, c.im:5:1,'i');
MulC(a, b, c);
WriteLn('Умножение: 'c.re:5:1, c.im:5:l,'i') ;
DivC(a, b, c);
WriteLn('Деление: 'c.re:5:l, c.im:5:1,'i');
end.
```

После объявления Uses Cmplx программе стали доступны все объекты, объявленные в интерфейсной части модуля CMPLX. При необходимости можно переопределить любой их этих объектов, как это произошло, например, с объявленной в модуле типизированной константой С. Переопределение объекта означает, что вновь объявленный объект «закрывает» ранее определенный в модуле одноименный объект. Чтобы получить доступ к «закрытому» объекту, нужно воспользоваться составным именем: перед именем объекта поставить имя модуля и точку. Например, оператор

```
WriteLn(cmplx.c.re:5:l, cmplx.c.im:5:1,'i');
```

выведет на экран содержимое «закрытой» типизированной константы из предыдущего примера.

Стандартные модули

Использование процедуры CRT

Программирование клавиатуры

Дополнительные возможности управления клавиатурой реализуются двумя функциями: KeyPressed и ReadKey.

Функция KeyPressed.

Возвращает значение типа Boolean, указывающее состояние буфера клавиатуры: False означает, что буфер пуст, а True - что в буфере есть хотя бы один символ, еще не прочитанный программой.

В MS-DOS реализуется так называемый асинхронный буферизованный ввод с клавиатуры. По мере нажатия на клавиши соответствующие коды помещаются в особый буфер, откуда они могут быть затем прочитаны программой. Стандартная длина буфера рассчитана на хранение до 16 кодов символов. Если программа достаточно долго не обращается к клавиатуре, а пользователь нажимает клавиши, буфер может оказаться переполненным. В этот момент раздается звуковой сигнал и «лишние» коды теряются. Чтение из буфера обеспечивается процедурами Read/ReadLn и функцией ReadKey. Замечу, что обращение к функции KeyPressed не задерживает исполнения программы: функция немедленно анализирует буфер и возвращает то или иное значение, не дожидаясь нажатия клавиши.

Функция ReadKey.

Возвращает значение типа Char. При обращении к этой функции анализируется буфер клавиатуры: если в нем есть хотя бы один не прочитанный символ, код этого символа берется из буфера и возвращается в качестве значения функции, в противном случае функция будет ожидать нажатия на любую клавишу. Ввод символа с помощью этой функции не сопровождается эхоповтором и содержимое экрана не меняется.

Пусть, например, в какой-то точке программы необходимо игнорировать все ранее нажатые клавиши, коды которых еще не прочитаны из буфера, т.е. необходимо очистить буфер. Этого можно достичь следующим способом:

```
Uses CRT;
var
C: Char;
begin
while KeyPressed do
C := ReadKey;
.....
end.
```

При использовании процедуры ReadKey необходимо учесть, что в клавиатурный буфер помещаются так называемые расширенные коды нажатых клавиш. Если нажимается любая алфавитно-цифровая клавиша, расширенный код совпадает с ASCII-кодом соответствующего символа. Например, если нажимается клавиша с латинской буквой «a» (в нижнем регистре), функция ReadKey возвращает значение chr (97), а если «A» (в верхнем регистре) - значение chr (65). При нажатии функциональных клавиш F1...F10, клавиш управления курсором, клавиш Ins, Home, Del, End, PgUp,

PgDn в буфер помещается двухбайтная последовательность: сначала символ #0, а затем расширенный код клавиши. Таким образом, значение #0, возвращаемое функцией ReadKey, используется исключительно для того, чтобы указать программе на генерацию расширенного кода. Получив это значение, программа должна еще раз обратиться к функции, чтобы прочитать расширенный код клавиши.

Т.е. код сканирования клавиши. Этот код определяется порядком, в соответствии с которым микропроцессор клавиатуры Intel 8042 периодически опрашивает (сканирует) состояние клавиш.

Следующая простая программа позволит Вам определить расширенный код любой клавиши. Для завершения работы программы нажмите клавишу Esc.

```
Uses CRT;
var
C: Char;
begin
repeat
C := ReadKey;
if C<>#0 then
WriteLn(ord(C))
else
WriteLnCO1 ,ord(ReadKey) :8)
until C=#27 {27 - расширенный код клавиши Esc}
end.
```

Если Вы воспользуетесь этой программой, то обнаружите, что нажатие на некоторые клавиши игнорируется функцией ReadKey. Это прежде всего так называемые сдвиговые клавиши - Shift, Ctrl, Alt. Сдвиговые клавиши в MS-DOS обычно используются для переключения регистров клавиатуры и нажимаются в сочетании с другими клавишами. Именно таким способом, например, различается ввод прописных и строчных букв. Кроме того, функция игнорирует переключающие клавиши Caps Lock, Num. Lock, Scroll Lock, а также «лишние» функциональные клавиши F11 и F12 клавиатуры IBM AT, не имеющие аналога на клавиатуре ранних моделей IBMPC/XT (в этих машинах использовалась 84-клавишная клавиатура, в то время как на IBM AT - 101-клавишная).

В табл. 1 приводятся расширенные коды клавиш, возвращаемые функцией ord(ReadKey). Для режима ввода кириллицы приводятся коды, соответствующие альтернативному варианту кодировки.

Таблица 9

Расширенные коды клавиш

Код		**Клавиша или комбинация клавиш**	**Код**		**Клавиша или комбинация клавиш**
Первый байт	**Второй байт**		**Первый байт**	**Второй байт**	
Алфавитно-цифровые клавиши					
8	-	Backspace (Забой)	9	-	Tab (Табуляция)
13	-	Enter	32	-	Пробел
33	-	!	34	-	"
35	-	#	36	-	$
37	-	%	38	-	&
39	-	'	40	-	(
41	-	)	42	-	*
43	-	+	44	-	,
45	-	-	46	-	.
47	-	/	46...57	-	0...9
58	-		59	-	;
60	-	<	61	-	=
62	-	>	63	-	?
64	-	@	65...90	-	A...Z
91	-	[	92	-	\
93	-	]	94	-	^
95	-		96	-	'
97...122	-	a...z	123	-	{
124	-	I	125	-	}
126	-	~	128...159	-	А...Я
160...175	-	а...п	224...239	-	р...я

Управляющие клавиши и их сочетания со сдвиговыми					
0	3	Ctrl-2	0	15	Shift-Tab
0	16...25	Alt-Q...Alt-P (верхний ряд букв)	0	30...38	Alt-A...Alt-L (средний ряд букв)
0	44...50	Alt-Z...Alt-M (нижний ряд букв)	0	59...68	F1...F10
0	- 71	Ноте	0	72	Курсор вверх
0	73	PgUp	0	75	Курсор влево
0	77	Курсор вправо	0	79	End
0	80	Курсор вниз	0	81	PgDn
0	82	Ins	0	83	Del
0	84...93	Shift-Fl...Shift-F10	0	94...103	Ctrl-F1... Ctrl-F10
0	104...113	Alt-Fl...Alt-F10	0	114	Ctrl-PrtScr
0	115	Ctrl-курсор влево	0	116	Ctrl-Курсор вправо
0	117	Ctrl-End	0	118	Ctrl-PgDn
0	119	Ctrl-Home	0	120...131	Alt-1. ..Alt-= (верхний ряд кла-виш)
0	132	Ctrl-PgUp			

Текстовый вывод на экран

Текстовые возможности CGA стали стандартом де-факто и поддерживаются во всех последующих разработках IBM - адаптерах EGA, MCGA, VGA и SVGA. Возможности модуля CRT рассматриваются применительно к адаптерам этого типа.

Процедура TextMode.

Используется для задания одного из возможных текстовых режимов работы адаптера. Заголовок процедуры:

Procedure TextMode(Mode: Word);

Здесь Mode - код текстового режима. В качестве значения этого выражения могут использоваться следующие константы, определенные в модуле CRT:

```
const
BW40=0{Черно-белый режим 40x25}
Co40=1{Цветной режим 40x25}
BW80=2{Черно-белый режим 80x25}
Co80=3{Цветной режим 80x25}
Mono=7{Используется с MDA}
Font8x8=256{Используется для загружаемого шрифта в режиме 80x43
или 80x50 с адаптерами EGA илиVGA}
```

Код режима, установленного с помощью вызова процедуры TextMode, запоминается в глобальной переменной LastMode модуля CRT и может использоваться для восстановления начального состояния экрана.

Следующая программа иллюстрирует использование этой процедуры в различных режимах. Отметим, что при вызове TextMode сбрасываются все ранее сделанные установки цвета и окон, экран очищается и курсор переводится в его левый верхний угол.

```
Uses CRT;
Procedure Print(S: String);
(Выводит сообщение S и ждет инициативы пользователя}
begin
WriteLn(S); {Выводим сообщение}
WriteLn('Нажмите клавишу Enter...');
ReadLn {Ждем нажатия клавиши Enter}
end; {Print}
var
LM: Word;{Начальный режим экрана}
begin
LM := LastMode; {Запо-  минаем начальный режим ра-
```

боты дисплея}

```
TextMode(Co40);
Print('Режим 40x25");
TextMode(CoSO) ;
Print('Режим 80x25');
TextMode(Co40+Font8x8);
Print('Режим Co40+Font8x8') ;
TextMode(Co80+Font8x8);
Print('Режим Co80+Font8x8');
{Восстанавливаем исходный режим работы:}
TextMode(LM)
end.
```

Процедура TextColpr.

Определяет цвет выводимых символов. Заголовок процедуры:

```
Procedure TextColor(Color: Byte);
```

Процедура TextBackground.

Определяет цвет фона. Заголовок:

```
Procedure TextBackground(Color: Byte);
```

Единственным параметром обращения к этим процедурам должно быть выражение типа Byte, задающее код нужного цвета. Этот код удобно определять с помощью следующих мнемонических констант, объявленных в модуле CRT:

```
const
Black = 0;{Черный}
Blue = 1;{Темно-синий}
Green = 2 ;{Темно-зеленый}
Cyan = 3;{Бирюзовый}
Red = 4 ;{Красный}
Magenta = 5;{Фиолетовый}
Brown = 6 ;{Коричневый}
LightGray = 7;{Светло-серый}
DarkGray = 8;{Темно-серый}
LightBlue = 9;{Синий}
LightGreen = 10;{Светло-зеленый}
LightCyan = 11;{Светло-бирюзовый}
LightRed = 12;{Розовый}
LightMagenta = 13;{Малиновый}
Yellow = 14;{Желтый}
White ' =15;{Белый}
Blink =128;{Мерцание символа}
```

Следующая программа иллюстрирует цветовые возможно-

сти Турбо Паскаля.

```
Uses CRT;
const
Col: array [1..15] of String [16] =
('темно-синий','темно-зеленый','бирюзовый','красный',
'фиолетовый','коричневый','светло-серый','темно-серый',
'синий','зеленый','светло-бирюзовый','розовый',
'малиновый','желтый','белый');
var
k: Byte;
begin
for k := 1 to 15 do
begin {Выводим 15 сообщений различными цветами}
TextColor(k);
WriteLn('Цвет ', k, ' - ',Col[k])
end;
TextColor(White+Blink); {Белые мигающие символы}
WriteLn('Мерцание символов');
{Восстанавливаем стандартный цвет}
TextColor(LightGray);
WriteLn
end.
```

Обратим внимание на последний оператор WriteLn: если его убрать, режим мерцания символов сохранится после завершения программы, несмотря на то, что перед ним стоит оператор

```
TextColor(LightGray)
```

Дело в том, что все цветовые определения предварительно заносятся в специальную переменную TextAttr модуля CRT и используются для настройки адаптера только при обращении к процедурам Write/WriteLn.

Процедура ClrScr.

Очищает экран или окно (см. ниже процедуру Window). После обращения к ней экран (окно) заполняется цветом фона и курсор устанавливается в его левый верхний угол. Например:

```
Uses CRT;
var
C: Char
begin
TextBackground(red) ;
ClrScr;{Заполняем экран красным цветом}
WriteLn('Нажмите любую клавишу...');
C := ReadKey; {Ждем нажатия любой клавиши}
```

```
TextBackground(Black) ;
ClrScr {Восстанавливаем черный фон экрана}
end.
```

Процедура Window.

Определяет текстовое окно - область экрана, которая в дальнейшем будет рассматриваться процедурами вывода как весь экран. Сразу после вызова процедуры курсор помещается в левый верхний угол окна, а само окно очищается (заполняется цветом фона). По мере вывода курсор, как обычно, смещается вправо и при достижении правой границы окна переходит на новую строку, а если он к этому моменту находился на последней строке, содержимое окна сдвигается вверх на одну строку, т.е. осуществляется «прокрутка» окна. Заголовок процедуры:

```
Procedure Window(XI,Y1,X2,Y2: Byte);
```

ЗдесьX1...Y2 - координаты левого верхнего (XI,Y1) и правого нижнего (X2,Y2) углов окна. Они задаются в координатах экрана, причем левый верхний угол экрана имеет координаты (1,1), горизонтальная координата увеличивается слева направо, а вертикальная - сверху вниз.

В следующем примере иллюстрируется вывод достаточно длинного сообщения в двух разных окнах.

```
Uses CRT;
var
k: integer;
begin
{Создаем левое окно -желтые символы на синем фоне:}
TextBackground(Blue);
Window(5,2,35,17);
TextColor(Yellow);
for k := 1 to 100 do
Write(' Нажмите клавишу Enter...');
ReadLn; {Ждем нажатия Enter}
ClrScr; {Очищаем окно}
{Создаем правое окно - белые символы на красном фоне:}
TextBackground(Red);
TextColor(White);
Window(40,2,70,17);
for k := 1 to 100 do
Write(' Нажмите клавишу Enter...');
ReadLn;
TextMode(C080) {Сбрасываем все установки}
end.
```

Обращение к процедуре Window игнорируется, если какая-либо из координат выходит за границы экрана или если нарушается одно из условий: X2>X1 и Y2>Y1. Каждое новое обращение к Window отменяет предыдущее определение окна. Границы текущего окна запоминаются в двух глобальных переменных модуля CRT: переменная WindMin типа Word хранит X1 и Y1 (XI - в младшем байте), а переменная того же типа WindMax - X2 и Y2(X2 - в младшем байте). При желании Вы можете изменять их нужным образом без обращения к Window. Например, вместо оператора

Window(40,2,70,17);

можно было бы использовать два оператора

WindMin := 39+(1 shl 8);
WindMax := 69+(16 shl 8);

(в отличие от обращения к Window координаты, хранящиеся в переменных WindMin и WindMax, соответствуют началу отсчета 0,0).

Процедура GotoXY.

Переводит курсор в нужное место экрана или текущего окна. Заголовок процедуры:

Procedure GotoXY(X,Y: Byte);

Здесь X, Y - новые координаты курсора. Координаты задаются относительно границ экрана (окна), т.е оператор

GotoXY(1,1);

означает указание перевести курсор в левый верхний угол экрана (или окна, если к этому моменту на экране определено окно). Обращение к процедуре игнорируется, если новые координаты выходят за границы экрана (окна).

Функции whereX и WhereY.

С помощью этих функций типа Byte можно определить текущие координаты курсора: WhereX возвращает его горизонтальную, а WhereY - вертикальную координаты.

В следующей программе сначала в центре экрана создается окно, которое обводится рамкой, затем в окне выводится таблица из двух колонок.

```
Uses CRT;
const
LU =#218;{Левый верхний угол рамки}
RU =#191;{Правый верхний угол)}
LD =#192;{Левый нижний}
RD =#217;{Правый нижний}
H =#196;{Горизонтальная черта}
V =#179;{Вертикальная черта}
```

```
X1 =14;{Координаты окна}
Y1 =5;
X2 =66;
Y2 =20;
Txt = 'Нажмите клавишу Enter...';
var
k: integer;
begin
ClrScr; {Очищаем экран}
{Создаем окно в центре экрана - желтые символы на синем фоне:}
TextBackground(Blue);
TextColor(Yellow);
Window(X1,Y1,X2,Y2);
ClrScr;
{Обводим окно рамкой}
Write(LU); {Левый верхний угол}
{Горизонтальная линия}
for k: = X1+1 to X2-1 do Write(H);
Write(RU);{Верхний правый угол}
for k := Y1+1 to Y2-1 do{Вертикальные линии}
begin
GotoXY(1,k-Y1+1);{Переходим к левой границе}
Write(V);{Левая черта}
GotoXY(X2-X1+1,WhereY){Правая граница}
Write(V){Правая черта}
end;
Write(LD);
{Левый нижний угол}
Window(X1,Y1,X2,Y2+1);{Расширяем вниз на одну строку координаты окна, иначе вывод в правый нижний угол вызовет прокрутку окна вверх}
GotoXY(2,Y2-Y1+1); {Возвращаем курсор из левого верхнего угла окна на нужное место}
{Горизонтальная рамка}
for k:= X1+1 to X2-1 do Write(H);
Write(RD); {Правый нижний угол}
{Определяем внутреннюю часть окна}
Window(X1+1,Y1+1,X2-1,Y2-1);
{Выводим левый столбец}
for k := Y1+1 to Y2-2 do
WriteLn('Левый столбец, строка ',k-Y1);;
```

```
{Ждем нажатия любой клавиши}
Write('Нажмите любую клавишу...');
k := ord(ReadKey); if k=0 then
k := ord(ReadKey);
DelLine; {Стираем приглашение}
{Выводим правый столбец}
for k := Y1+1 to Y2-2 do
begin
GotoXY((X2-X1) div 2,k-Y1);
Write('Правый столбец, строка ',k-Y1)
end ;
{Выводим сообщение и ждем нажатия клавиши Enter}
GotoXY((X2-X1-Length(Txt)) div 2,Y2-Y1-1);
TextColor(White);
Write(Txt);
ReadLn;
{Восстанавливаем стандартный режим}
TextMode(CO80)
end.
```

Три следующие процедуры без параметров могут оказаться полезными при разработке текстовых редакторов.

Процедура ClrEOL.

Стирает часть строки от текущего положения курсора до правой границы окна (экрана). Положение курсора не меняется.

Процедура DelLine.

Уничтожает всю строку с курсором в текущем окне (или на экране, если окно не создано). При этом все строки ниже удаляемой (если они есть) сдвигаются вверх на одну строку.

Процедура InsLine.

Вставляет строку: строка с курсором и все строки ниже ее сдвигаются вниз на одну строку; строка, вышедшая за нижнюю границу окна (экрана), безвозвратно теряется; текущее положение курсора не меняется.

Процедуры LowVideo, NormVideo и HighVideo.

С помощью этих процедур без параметров можно устанавливать соответственно пониженную, нормальную и повышенную яркость символов. Например:

```
Uses CRT;
begin
LowVideo;
WriteLn('Пониженная яркость');
NormVideo;
```

```
WriteLn('Нормальная яркость');
HighVideo;
WriteLn('Повышенная яркость')
end.
```

Заметим, что на практике нет разницы между пониженной и нормальной яркостью изображения.

Процедура AssignCRT.

Связывает текстовую файловую переменную F с экраном с помощью непосредственного обращения к видеопамяти (т.е. к памяти, используемой адаптером для создания изображения на экране). В результате вывод в такой текстовый файл осуществляется значительно (в 3...5 раз) быстрее, чем если бы этот файл был связан с экраном стандартной процедурой Assign. Заголовок процедуры:

```
Procedure AssignCRT(F: Text);
```

В следующей программе измеряется скорость вывода на экран с помощью стандартной файловой процедуры и с помощью непосредственного обращения к видеопамяти. Вначале файловая переменная F связывается «медленной» процедурой Assign со стандартным устройством CON (т.е. с экраном) и подсчитывается количество N1 циклов вывода некоторого текста за 5*55 = 275 миллисекунд системных часов. Затем файловая переменная связывается с экраном с помощью процедуры быстрого доступа AssignCRT и точно так же подсчитывается количество N2 циклов вывода. В конце программы счетчики N1 и N2 выводятся на экран.

```
Uses CRT;
var
F: Text;
t: LongInt;{Начало отсчета времени}
N1,N2: Word;{Счетчики вывода}
const
txt = ' Text';
begin
{----- Стандартный вывод в файл -----}
Assign(F,'CON');
Rewrite(F);
N1 := 0;{Готовим счетчик вывода}
ClrScr;{Очищаем экран}
{Запоминаем начальный момент:}
t := MemL[$0040:$006C];
{Ждем начала нового 55-мс интервала, чтобы исключить
```

```
погрешность в определении времени:}
        while MemL[$0040:$006C]=t do;
        {Цикл вывода за 5 интервалов}
        while MemL[$0040:$006C]<t+6 do
        begin
        inc(N1) ;
        Write(F,txt)
        end;
        Close(F);
        {----- Вывод с помощью быстрой процедуры прямого доступа к экрану - ----}
        AssignCRT(F);
        Rewrite(F);
        N2 := 0;
        ClrScr;
        t := MemL[$0040:$006C];
        while MemL[$0040:$006C]=t do;
        while MemL[$0040:$006C]<t+6 do
        begin
        inc(N2);
        Write(F,txt)
        end ;
        Close(F);
        {Печатаем результат}
        ClrScr;
        WriteLn(Nl,N2:10)
        end.
```

Следует учесть, что вывод на экран обычным образом - без использования файловой переменной (например, оператором Write (txt)) также осуществляется с помощью непосредственного доступа к видеопамяти, поэтому ценность процедуры AssignCRT весьма сомнительна. Прямой доступ к видеопамяти регулируется глобальной логической переменной DirectVideo модуля CRT: если эта переменная имеет значение True, доступ разрешен, если False - доступ к экрану осуществляется с помощью относительно медленных средств операционной системы MS-DOS. По умолчанию переменная DirectVideo имеет значение True.

14. ОСНОВЫ И МЕТОДЫ ЗАЩИТЫ ИНФОРМАЦИИ

Общие понятия информационной безопасности

Персональные компьютеры, системы управления и сети на их основе, быстро входят во все области человеческой деятельности. Среди них можно выделить такие сферы применения как военная, коммерческая, банковская, посредническая, научные исследования по высоким технологиям и другие. Очевидно, широко используя компьютеры и сети для обработки и передачи информации, эти отрасли должны быть надежно защищены от возможности доступа к ней посторонних лиц, ее утраты или искажения. Согласно статистическим данным, более 80% компаний несут финансовые убытки из-за нарушения целостности и конфиденциальности используемых данных.

Кроме информации, составляющей государственную или коммерческую тайну, существует информация, представляющая собой интеллектуальную собственность. К ней можно отнести результаты научных исследований, программы, обеспечивающие функционирование компьютера, игровые программы, оригинальные аудио и видео клипы, которые находятся под защитой законов, принятых в большинстве стран мирового сообщества. Стоимость такой информации в мире составляет несколько триллионов долларов в год. Её несанкционированное копирование снижает доходы компаний и авторов, занятых её разработкой.

Усложнение методов и средств организации машинной обработки, повсеместное использование глобальной сети Internet приводит к тому, что информация становится всё более уязвимой. Этому способствуют такие факторы, как постоянно возрастающие объёмы обрабатываемых данных, накопление и хранение данных в ограниченных местах, постоянное расширение круга пользователей, имеющих доступ к ресурсам, программам и данным, недостаточный уровень защиты аппаратных и программных средств компьютеров и коммуникационных систем и т.п.

Учитывая эти факты, защита информации в процессе её сбора, хранения, обработки и передачи приобретает исключительно важное значение.

Основные понятия информационной безопасности

Введем ряд определений, используемых при описании средств и методов защиты информации в системах автоматизированной обработки, построенных на основе средств вычислительной техники.

Компьютерная система (КС) — организационно-техническая

система, представляющую совокупность следующих взаимосвязанных компонентов:

- технические средства обработки и передачи данных;
- *методы и алгоритмы обработки* в виде соответствующего программного обеспечения;
- *данные* — информация на различных носителях и находящаяся в процессе обработки;
- *конечные пользователи* — персонал и пользователи, использующие КС с целью удовлетворения информационных потребностей;
- объект доступа, или объект, — любой элемент КС, доступ к которому может быть произвольно ограничен (файлы, устройства, каналы);
- *субъект доступа, или субъект*, — любая сущность, способная инициировать выполнение операций над объектом (пользователи, процессы).

Информационная безопасность — состояние КС, при котором она способна противостоять дестабилизирующему воздействию внешних и внутренних информационных угроз и при этом не создавать таких угроз для элементов самой КС и внешней среды.

Конфиденциальность информации — свойство информации быть доступной только ограниченному кругу конечных пользователей и иных субъектов доступа, прошедших соответствующую проверку и допущенных к ее использованию.

Целостность информации — свойство сохранять свою структуру

и содержание в процессе хранения, использования и передачи.

Достоверность информации — свойство, выражаемое в строгой

принадлежности информации субъекту, который является ее источником.

Доступ к информации — возможность субъекта осуществлять определенные действия с информацией.

Санкционированный доступ к информации — доступ с выполнением правил разграничения доступа к информации.

Несанкционированный доступ (НСД) — доступ с нарушениемправил разграничения доступа субъекта к информации, с использованием штатных средств (программного или аппаратного обеспечения), предоставляемых КС.

Правила разграничения доступа — регламентация прав до-

ступа субъекта к определенному компоненту системы.

Идентификация — получение от субъекта доступа к сведениям (имя, учетный номер и т.д.), позволяющим выделить его из множества субъектов.

Аутентификация — получение от субъекта сведений (пароль, биометрические параметры и т.д.), подтверждающих, что идентифицируемый субъект является тем, за кого себя выдает.

Угроза информационной безопасности КС — возможность воздействия на информацию, обрабатываемую КС, с целью ее искажения, уничтожения, копирования или блокирования, а также возможность воздействия на компоненты КС, приводящие к сбою их функционирования.

Уязвимость КС — любая характеристика, которая может привести к реализации угрозы.

Атака КС — действия злоумышленника, предпринимаемые с целью обнаружения уязвимости КС и получения несанкционированного доступа к информации.

Безопасная, или защищенная, КС — КС, снабженная средствами защиты для противодействия угрозам безопасности.

Комплекс средств защиты — совокупность аппаратных и программных средств, обеспечивающих информационную безопасность.

Политика безопасности — совокупность норм и правил, регламентирующих работу средств защиты от заданного множества угроз.

Дискреционная модель разграничения доступа — способ разграничения доступа субъектов к объектам, при котором права доступа задаются некоторым перечнем прав доступа субъекта к объекту. При реализации представляет собой матрицу, строками которой являются субъекты, а столбцами — объекты; элементы матрицы характеризуют набор прав доступа.

Полномочная (мандатная) модель разграничения доступа — способ разграничения доступа субъектов к объектам, при котором каждому объекту ставится в соответствие уровень секретности, а каждому субъекту уровень доверия к нему. Субъект может получить доступ к объекту, если его уровень доверия не меньше уровня секретности объекта.

Анализ угроз информационной безопасности

Для успешного противодействия угрозам и атакам КС, а также выбора способов и средств защиты, политики безопасности и анализа рисков от возможного НСД, необходимо классифициро-

вать существующие угрозы информационной безопасности. Каждый признак классификации должен отражать одно из обобщённых требований к системе защиты, а сами угрозы позволяют детализировать эти требования. Современные КС и сети являются сложными системами, подверженными, кроме того, влиянию чрезвычайно большого числа факторов и поэтому формализовать задачу описания полного множества угроз не представляется возможным. Как следствие, для защищённой КС определяется не полный перечень угроз, а перечень классов угроз, которым должен противодействовать комплекс средств защиты.

Классификация угроз может быть проведена по ряду базовых признаков:

1. По природе возникновения: объективные природные явления, не зависящих от человека; субъективные действия, вызванные деятельностью человека.

2. По степени преднамеренности: ошибки конечного пользователя или персонала; преднамеренного действия, для получения НСД к информации.

3. По степени зависимости от активности КС: проявляющиеся независимо от активности КС (вскрытие шифров, хищение носителей информации); проявляющиеся в процессе обработки данных (внедрение вирусов, сбор "мусора" в памяти, сохранение и анализ работы клавиатуры и устройств отображения).

4. По степени воздействия на КС: пассивные угрозы (сбор данных путём выведывания или подсматривания за работой пользователей); активные угрозы (внедрение программных или аппаратных закладок и вирусов для модификации информации или дезорганизации работы КС).

5. По способу доступа к ресурсам КС: получение паролей и прав доступа, используя халатность владельцев и персонала, несанкционированное использование терминалов пользователей, физического сетевого адреса, аппаратного блока кодирования и др.; обход средств защиты, путём загрузки посторонней операционной защиты со сменного носителя; использование недокументированных возможностей операционной системы.

6. По текущему месту расположения информации в КС: внешние запоминающие устройства; оперативная память; сети связи; монитор или иное отображающее устройство (возможность скрытой съёмки работы принтеров, графопостроителей, световых панелей и т.д.).

Необходимо отметить, что абсолютно надёжных систем защиты не существует. Кроме того, любая система защиты увеличи-

вает время доступа к информации, поэтому построение защищённых КС не ставит целью надёжно защититься от всех классов угроз. Уровень системы защиты – это компромисс между понесёнными убытками от потери конфиденциальности информации, с одной стороны, и убытками от усложнения, удорожания КС и увеличения времени доступа к ресурсам от введения систем защиты, с другой стороны.

Критерии защищенности средств компьютерных систем

Министерством обороны США в 1983 году были разработаны определения требований к аппаратному, программному и специальному программному обеспечению под названием "Критерии оценки безопасности компьютерных систем", получившие неофициальное, но прочно утвердившееся название "Оранжевая книга".

В "Оранжевой книге" предложены три категории требований безопасности: политика безопасности, аудит (мониторинг производимых действий), корректность, в рамках которых сформулированы шесть базовых критериев безопасности.

Критерий 1. Политика безопасности. КС должна поддерживать точно определённую политику безопасности. Возможность доступа субъектов к объектам должна определяться на основании их идентификации и набора правил управления доступом. Там, где это возможно должна использоваться мандатное управление доступом, позволяющее эффективно разграничивать доступ к информации разной степени конфиденциальности.

Критерий 2. Метки. Каждый объект доступа в КС должен иметь метку безопасности, используемую в качестве исходной информации для исполнения процедур контроля доступа.

Критерий 3. Идентификация и аутентификация. Все субъекты должны иметь уникальные идентификаторы. Доступ субъекта к ресурсам КС должен осуществляться на основании результатов идентификации и подтверждения подлинности их идентификаторов (аутентификация). Идентификаторы и аутентификационные данные должны быть защищены от НСД, модификации и уничтожения.

Критерий 4. Регистрация и учёт. Для определения степени ответственности пользователей за действия в системе, все происходящие в ней события, имеющие значение для поддержания конфиденциальности и целостности информации должны отслеживаться и регистрироваться в защищённом объекте (файле-журнале). Система регистрации должна осуществлять анализ об-

щего потока событий и выделять из него только те события, которые оказывают влияние на безопасность КС. Доступ к объекту аудита для просмотра должен быть разрешён только специальной группе пользователей – аудиторов. Запись должен быть разрешён только субъекту, олицетворяющему систему.

Критерий 5. Контроль корректности функционирования средств защиты. Все средства защиты, обеспечивающие политику безопасности, должны находиться под контролем средств, проверяющих корректность их функционирования и быть независимыми от них.

Критерий 6. Непрерывность защиты. Все средства защиты должны быть защищены от несанкционированного воздействия или отключения. Защита должна быть постоянной и непрерывной в любом режиме функционирования системы, защиты и КС. Это требование должно распространяться на весь жизненный цикл КС.

Гостехкомиссией при Президенте Российской Федерации были приняты руководящие документы, посвящённые вопросам защиты информации в автоматизированных системах. Основой этих документов является концепция защиты средств вычислительной техники и автоматизированных систем от несанкционированного доступа к информации и основные принципы защиты КС.

Для определения принципов защиты информации вводится понятие несанкционированного доступа к информации. Это понятие является чрезвычайно важным, так как определяет, от чего сертифицированные по руководящим документам средства вычислительной техники и КС должны защищать информацию. В соответствие с принятой в руководящих документах классификацией, основными способами НСД являются:

- непосредственное обращение к объектам доступа (получение процессом, управляемым пользователем доступа к файлу);
- создание программных и технических средств, выполняющих обращение к объектам доступа в обход средств защиты;
- модификация средств защиты, позволяющая осуществить НСД (программные и аппаратные закладки);
- внедрение в технические средства аппаратных или программных механизмов, нарушающих структуру и функции КС и позволяющие осуществить НСД (загрузка нестандартной операционной системы без функций защиты).

Руководящие материалы представляют семь критериев защиты КС.

1. Защита КС основывается на положениях существующих законов, стандартов и нормативно-методических документов по защите информации.

2. Защита средств вычислительной техники обеспечивается комплексом программно-технических средств.

3. Защита КС обеспечивается комплексом программно-технических средств и поддерживающих их организационных мер.

4. Защита КС должна обеспечиваться на всех технологических этапах обработки информации и во всех режимах функционирования, в том числе при проведении ремонтных и регламентных работ.

5. Программно-технические средства не должны существенно ухудшать основные функциональные характеристики КС (надёжность, производительность, возможность изменения конфигурации).

6. Оценка эффективности средств защиты, учитывающей всю совокупность технических характеристик, включая технические решения и практическую реализацию средств защиты.

7. Защита КС должна предусматривать контроль эффективности средств защиты от НСД, который может быть периодическим или включаться по мере необходимости пользователем или контролирующими органами.

ЛИТЕРАТУРА

1. Беляев М.А., Лысенко В.В. Основы информатики Изв-во: Феникс, 2010
2. Меженный О.А. Самоучитель Turbo Pascal Изд-во: Диалектика, 2008. –336с.
3. Федоренко Ю. Алгоритмы и программы на Turbo Pascal.Изд-во:Питер,2001. - 240 с.
4. Информатика: Учебник. /Под ред. Н.В. Макаровой. – М: Финансы и статистика, 2001. – 768 с.
5. Информатика. Базовый курс/ Под ред. С.В. Симонович – СПб: Издательство «Питер», 2000. - 640 с.
6. Кетков Ю.Л. и др. Персональный компьютер: Школьная энциклопедия. М.: Большая Российская энциклопедия, 1998.
7. Культин Н.Б.. Turbo Pascal в задачах и примерах. СПб.: БХВ – Санкт-Петербург, 2006- 256с..
8. Вирт Н. Алгоритмы и структуры данных: пер. с англ. - М.: Мир, 1985.
9. Немнюгин С.А. Turbo Pascal. СПб: Питер, 2000.
10. Соболь Б.В. [и др.]-Информатика : учебник/ Изд. 3-е, дополн. и перераб. — Ростов н/Д: Феникс, 2007. — 446с.-(Высшее образование).
11. Анин Б. А. Защита компьютерной информации. — СПб.: БХВ-Петербург, 2000. - 384 с.