Федеральное агентство по образованию
АМУРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ГОУВПО «АмГУ»
Факультет математики и информатики

УТВЕРЖДАЮ
Зав. кафедрой МАиМ
Т.В. Труфанова
«___»_____ 2007г.

ИНФОРМАТИКА

Учебно – методический

комплекс дисциплины

Составитель: Д.Л. Харичева

Благовещенск
2007

ББК
Х

Печатается по решению редакционно-издательского совета факультета математики и информатики Амурского государственного университета

Харичева Д.Л.

Информатика. Учебно – методический комплекс дисциплины для студентов очной формы обучения специальности 010101 «Математика». – Благовещенск: Амурский гос. ун–т, 2007. − 163 с.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Основной целью дисциплины является обучение студентов основам работы на ЭВМ, техническим и программным средствам реализации информационных процессов, программированию, теоретическим основам и практическим навыкам проектирования и реализации программ на современных ЭВМ, а также знакомство и получение навыков работы с современными информационными технологиями и сетями.

В результате изучения дисциплины студенты должны знать что такое информация, общие характеристики процессов сбора, передачи, обработки и накопления информации и уметь использовать технические и программные средства реализации информационных процессов, системные программные средства, операционные системы, оболочки, сервисные программы, возможности современных информационных технологий, базовые конструкции алгоритмов и их реализацию на ЭВМ, современные методы и средства программирования, а также современные языки программирования.

Информатика изучается студентами – математиками в 1 семестре первого курса. По данной дисциплине предусмотрено 36 часов лекций, 36 часов практических занятий, 36 часов лабораторных занятий и 45 часов самостоятельной работы – всего 153 часа. По итогам изучения предусмотрен экзамен.

ГОСУДАРСТВЕННЫЙ ОБРАЗОВАТЕЛЬНЫЙ СТАНДАРТ ПО ИНФОРМАТИКЕ

Предмет, цели и задачи курса. Вычислительная техника и научно-технический прогресс. Представление об информационном обществе. Информационные ресурсы, продукты и услуги. ЭВМ, их назначение и использование в различных областях человеческой деятельности. Понятие информации и кодирования. Виды и свойства информации. Этапы обработки информации в ЭВМ. Общая характеристика сбора, хранения, обработки, накопления и передачи информации. Способы представления и преобразования информации. ЭВМ как универсальное устройство обработки информации, классификации ЭВМ. Структура и функции аппаратной части ПК. Тенденции развития ЭВМ. Программное обеспечение ЭВМ, его состав и назначение: Системное программное обеспечение. Понятие операционной системы, назначение. Характеристика и основы работы основных операционных систем - MS-DOS, WINDOWS, UNIX, OS/2 и др. Программы-драйверы, программы - оболочки (NC, WC), утилиты. Прикладное программное обеспечение. Классификация прикладных программ. Текстовые, табличные, графические редакторы, базы данных, научные редакторы и системы, интегрированные и издательские системы - краткая характеристика, назначение и классификация. Языки и системы программирования. Понятие компьютерных сетей, общие принципы организации и функционирования. Архитектура сетей. Локальные и глобальные компьютерные сети. Интернет. Понятие информационных технологий и систем, классификации, назначение, области применения. ГИС-технологии, назначение и основные возможности. Экспертные системы. Алгоритм, способы задания и описания алгоритмов. Основные конструкции алгоритмов. Типы алгоритмов. Организация алгоритмов линейной, разветвляющейся, циклической и сложной комбинированной структуры. Этапы внутреннего и внешнего проектирования. Отладка и тестирование программ. Модели надежности. Основы программирования на языке

Паскаль. Общая характеристика Языка. Основные конструкции Языка. Понятие блочной структуры. Синтаксические диаграммы. Структура программы. Константы и переменные. Типы данных. Ввод-вывод данных. Операторы. Процедуры и функции. Рекурсия. Структурированные типы данных: строки, массивы, множества, записи. Файлы в Паскале. Особенности различных реализаций Паскаля. Работа с графикой в среде Турбо Паскаль. Аппаратная и программная поддержка графики, инициализация графики. Базовые процедуры и функции: построения графических образов, работы с цветовой палитрой, работы с текстом.

ТЕМАТИЧЕСКИЙ ПЛАН ЛЕКЦИОННЫХ ЗАНЯТИЙ

Наименование темы		Количество часов
1	Вычислительная техника в работе	2
2	Технические средства ЭВМ	2
3	Системное и прикладное программное обеспечение. Языки программирования	2
4	Современные информационные технологии и сети	2
5	Алгоритмы и алгоритмизация	2
6	Этапы и методы разработки программ	2
7	Основные элементы языка Паскаль	2
8	Типы данных в языке Паскаль. Выражения, операции, операнды	2
9	Ввод-вывод данных. Простые и составные операторы	2
10	Структурные операторы: if, case, while, repeat, for. Вложенные операторы цикла	2
11	Процедуры и функции на языке Паскаль. Рекурсия	2
12	Массивы, действия над ними. Сортировка массивов	2
13	Строки, записи, множества в Паскале	2
14	Файлы в Паскале	2
15	Множества. Динамические структуры данных	2
16	Особенности различных реализаций Паскаля	2
17	Графические возможности Паскаля. Инициализация графики.	2
18	Базовые процедуры и функции построения графических образов, работы с цветовой палитрой и текстом	2
	ИТОГО	36

ТЕМАТИЧЕСКИЙ ПЛАН ПРАКТИЧЕСКИХ ЗАНЯТИЙ

Наименование темы		Количество часов
1	Информация, ее виды и свойства. Способы представления и преобразования информации. Системы счисления	2
2	Современные информационные сети, Интернет. Услуги, предоставляемые сетью. Современные информационные технологии и системы	2
3	Алгоритмы линейной, разветвленной и циклической структуры	2
4	Итерационные алгоритмы	2
5	Обработка массивов данных	2
6	Функции и процедуры	2
7	Обработка символьных данных	2
8	Работа с записями	2
9	Работа с внешними файлами	2
	ИТОГО	18

В ходе выполнения практических работ по данному курсу студенты приобретают знания о способах представления и преобразования информации, ее кодирования, технических и программных средствах реализации информационных процессов, алгоритмизации, основных этапах решения задач на ЭВМ, современных языках программирования.

ТЕМАТИЧЕСКИЙ ПЛАН ЛАБОРАТОРНЫХ ЗАНЯТИЙ

Наименование темы		Количество часов
1		2
1	Знакомство с ПК. Изучение операционной системы MS-DOS. Работа с файлами и каталогами	2
2	Программная оболочка NC, меню, функциональные клавиши. Работа с файлами и катало-	2
3	Знакомство с операционной системой WINDOWS - базовые возможности. Прикладное	4
4	Изучение среды программирования Турбо Паскаль. Программирование формул.	1

1		2
5	Программы разветвляющей структуры. Табулирование функций	1
6	Программирование алгоритмов циклической структуры. Нахождение корней уравнений	2
7	Нахождение суммы ряда с точностью	2
8	Вычисление интеграла	2
9	Вложенные оператора цикла. Нахождение экстремума функции	2
10	Работа с одномерными массивами	2
11	Обработка матриц	2
12	Подпрограммы в языке Паскаль. Процедуры	2
13	Подпрограммы в языке Паскаль. Функции	2
14	Обработка символьных данных. Множества	2
15	Записи	2
16	Файлы	2
17	Инициализация графики. Построение графических примитивов	2
18	Работа с текстом. Построение графиков функций	2
ИТОГО		36

При выполнении лабораторных работ по данному курсу студенты должны ознакомиться с современными методами и средствами программирования, языком Турбо Паскаль, его синтаксисом и семантикой, типах данных, способах и механизмах управления данными, методах и основных этапах трансляции; продемонстрировать умение программирования различных функциональных и вычислительных задач. Лабораторная работа выполнятся строго в соответствии с выданным преподавателем заданием и вариантом. По окончанию занятия студент обязан сдать разработанную программу и объяснить алгоритм решения поставленной задачи.

САМОСТОЯТЕЛЬНАЯ РАБОТА СТУДЕНТОВ

В качестве самостоятельной работы по дисциплине «Информатика»

студентам предлагается рассмотреть и изучить следующие вопросы:

1. Операционная система MS DOS, оболочка NC, операционная система WINDOWS.
2. Работа с текстовыми, табличными редакторами, базами данных - основные возможности.
3. Глобальные и локальные сети. Работа в Интернет, основные услуги, предоставляемые сетями. Посещение Интернет-центра.
4. Среда программирования Турбо Паскаль. Функциональные возможности меню.
5. Методы сортировки массивов.
6. Численные методы решения уравнений (половинного деления, касательных, Ньютона, простой итерации).
7. Приближенное интегрирование функции (метод прямоугольников, метод трапеций).
8. Типы данных (стек, очередь).

МАТЕРИАЛЫ ДЛЯ ЧТЕНИЯ ЛЕКЦИЙ И ПРОВЕДЕНИЯ ПРАКТИЧЕСКИХ ЗАНЯТИЙ

ИНФОРМАЦИЯ И ФОРМЫ ЕЕ ПРЕДСТАВЛЕНИЯ

Понятие ***информации*** является основополагающим понятием информатики. Любая деятельность человека представляет собой процесс сбора и переработки информации, принятия на ее основе решении и их выполнения. С появлением современных средств вычислительной техники информация стала выступать в качестве одного из важнейших ресурсов научно-технического прогресса.

Информация содержится в человеческой речи, текстах книг, журналов и газет, сообщениях радио и телевидения, показаниях приборов и т. д. Человек воспринимает информацию с помощью органов чувств, хранит и перерабатывает ее с помощью мозга и центральной нервной системы. Передаваемая информация обычно касается каких-то предметов или нас самих и связана с событиями, происходящими в окружающем нас мире.

В рамках науки информация является первичным и неопределяемым понятием. Оно предполагает наличие материального носителя информации, источника информации, передатчика информации, приемника и канала связи между источником и приемником. Понятие информации используется во всех сферах: науке, технике, культуре, социологии и повседневной жизни. Конкретное толкование элементов, связанных с понятием информации, зависит от метода конкретной науки, цели исследования или просто от наших представлений.

Термин ***«информация»*** происходит от латинского *informatio* — разъяснение, изложение, осведомленность. Энциклопедический словарь (М.: Сов. энциклопедия, 1990) определяет информацию в исторической эволюции: первоначально — сведения, передаваемые людьми устным, письменным или другим способом (с помощью условных сигналов, технических средств и т. д.); с середины XX века — общенаучное понятие, включающее обмен сведениями между людьми, человеком и автоматом, обмен сигналами в животном и растительном мире (передача признаков от клетки к клетке, от организма к организму).

Более узкое определение дается в технике, где это понятие включает в себя все сведения, являющиеся объектом хранения, передачи и преобразования.

Наиболее общее определение имеет место в философии, где под информацией понимается отражение реального мира. Информацию как

философскую категорию рассматривают как один из атрибутов материи, отражающий ее структуру.

В эволюционном ряду *вещество -> энергия -> информация* каждое последующее проявление материи отличается от предыдущего тем, что людям было труднее его распознать, выделить и использовать в чистом виде. Именно сложность выделения различных проявлений материи обусловила, наверное, указанную последовательность познания природы человечеством.

С понятием информации связаны такие понятия, как сигнал, сообщение и данные.

Сигнал (от латинского *signum* — знак) представляет собой любой процесс, несущий информацию.

Сообщение — это информация, представленная в определенной форме и предназначенная для передачи.

Данные — это информация, представленная в формализованном виде и предназначенная для обработки ее техническими средствами, например, ЭВМ.

Различают две формы представления информации — непрерывную и дискретную. Поскольку носителями информации являются сигналы, то в качестве последних могут использоваться физические процессы различной природы. Например, процесс протекания электрического тока в цепи, процесс механического перемещения тела, процесс распространения света и т. д. Информация представляется (отражается) значением одного или нескольких параметров физического процесса (сигнала), либо комбинацией нескольких параметров.

Сигнал называется непрерывным, если его параметр в заданных пределах может принимать любые промежуточные значения. Сигнал называется дискретным, если его параметр в заданных пределах может принимать отдельные фиксированные значения.

Следует различать непрерывность или дискретность сигнала по уровню и во времени.

На рис. 1 в виде графиков изображены: а) непрерывный по уровню и во времени сигнал *Хнн;* б) дискретный по уровню и непрерывный во времени сигнал *Хдн*; в) непрерывный по уровню и дискретный во времени сигнал *Хнд*; г) дискретный по уровню и во времени сигнал *Хдд*.

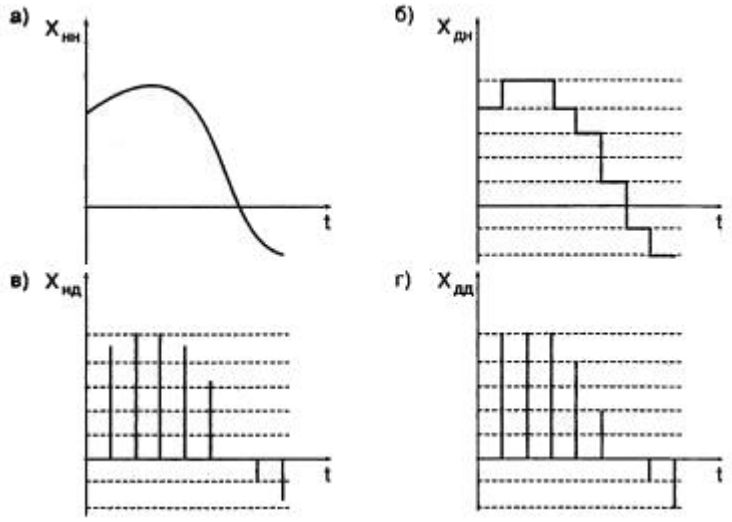


Рис. 1. Виды информационных процессов

Наконец, все многообразие окружающей нас информации можно сгруппировать по различным признакам, т. е. классифицировать по видам. Например, в зависимости от области возникновения информацию, отражающую процессы и явления неодушевленной природы, называют элементарной, процессы животного и растительного мира — биологической, человеческого общества — социальной.

По способу передачи и восприятия различают следующие виды информации: визуальную — передаваемую видимыми образами и символами, аудиальную — звуками, тактильную — ощущениями, органолептическую — запахами и вкусом, машинную — выдаваемую и воспринимаемую средствами вычислительной техники, и т. д.

ПОНЯТИЕ КОЛИЧЕСТВА ИНФОРМАЦИИ

Количеством информации называют числовую характеристику сигнала, отражающую ту *степень неопределенности* (неполноту знаний), которая исчезает после получения сообщения в виде данного сигнала. Эту меру неопределенности в теории информации называют ***энтропией.*** Если в результате получения сообщения достигается полная ясность в каком-то вопросе, говорят, что была получена полная или исчерпывающая информация и необходимости в получении дополнительной информации нет. И, наоборот, если после получения сообщения неопределенность осталась прежней, значит, информации получено не было (нулевая информация).

Приведенные рассуждения показывают, что между понятиями информация, неопределенность и возможность выбора существует тесная связь. Так, любая неопределенность предполагает возможность выбора, а любая информация, уменьшая неопределенность, уменьшает и возможность выбора. При полной информации выбора нет. Частичная информация

уменьшает число вариантов выбора, сокращая тем самым неопределенность.

Пример. Человек бросает монету и наблюдает, какой стороной она упадет. Обе стороны монеты равноправны, поэтому одинаково вероятно, что выпадет одна или другая сторона. Такой ситуации приписывается начальная неопределенность, характеризуемая двумя возможностями. После того, как монета упадет, достигается полная ясность и неопределенность исчезает (становится равной нулю).

Приведенный пример относится к группе событий, применительно к которым может быть поставлен вопрос типа «да-нет». Количество информации, которое можно получить при ответе на вопрос типа «да-нет», называется ***битом*** (англ. *bit* — сокращенное от *binary digit* — двоичная единица). Бит — минимальная единица количества информации, ибо получить информацию меньшую, чем 1 бит, невозможно. При получении информации в 1 бит неопределенность уменьшается в 2 раза. Таким образом, каждое бросание монеты дает нам информацию в 1 бит.

В качестве других моделей получения такого же количества информации могут выступать электрическая лампочка, двухпозиционный выключатель, магнитный сердечник, диод и т. п. Включенное состояние этих объектов обычно обозначают цифрой 1, а выключенное — цифрой 0.

Рассмотрим систему из двух электрических лампочек, которые независимо друг от друга могут быть включены или выключены. Для такой системы возможны следующие состояния:

Лампа A 0 0 1 1

Лампа B 0 1 0 1

Чтобы получить полную информацию о состоянии системы, необходимо задать два вопроса типа «да-нет» — по лампочке A и лампочке B соответственно. В этом случае количество информации, содержащейся в данной системе, определяется уже в 2 бита, а число возможных состояний системы — 4. Если взять три лампочки, то необходимо задать уже три вопроса и получить 3 бита информации. Количество состояний такой системы равно 8 и т. д.

Связь между количеством информации и числом состояний системы устанавливается формулой Хартли:

$i=\log_2 N$,

где i — количество информации в битах; *N* —. число возможных состояний. Ту же формулу можно представить иначе:

$N = 2^i$.

Группа из 8 битов информации называется ***байтом.*** Если бит — минимальная единица информации, то байт ее основная единица. Существуют производные единицы информации: килобайт (кбайт, кб), мегабайт (Мбайт, Мб) и гигабайт (Гбайт, Гб).

1 кб =1024 байта - 2^{10} (1024) байтов.

1 Мб = 1024 кбайта = 2^{20}(1024 х 1024) байтов.

1 Гб = 1024 Мбайта - 2^{30} (1024х1024 х 1024)байтов.

Эти единицы чаще всего используют для указания объема памяти ЭВМ.

ИНФОРМАЦИОННЫЕ ПРОЦЕССЫ И ТЕХНОЛОГИИ

Информационные процессы (сбор, обработка и передача информации) всегда играли важную роль в науке, технике и жизни общества. В ходе эволюции человечества просматривается устойчивая тенденция к автоматизации этих процессов, хотя их внутреннее содержание по существу осталось неизменным.

Сбор информации — это деятельность субъекта, в ходе которой он получает сведения об интересующем его объекте. Сбор информации может производиться или человеком, или с помощью технических средств и систем — аппаратно. Например, пользователь может получить информацию о движении поездов или самолетов сам, изучив расписание, или же от другого человека непосредственно, либо через какие-то документы, составленные этим человеком, или с помощью технических средств (автоматической справки, телефона и т. д.). Задача сбора информации не может быть решена в отрыве от других задач, — в частности, задачи обмена информацией (передачи).

Обмен информацией — это процесс, в ходе которого источник информации ее передает, а получатель — принимает. Если в передаваемых сообщениях обнаружены ошибки, то организуется повторная передача этой информации. В результате обмена информацией между источником и получателем устанавливается своеобразный «информационный баланс», при котором в идеальном случае получатель будет располагать той же информацией, что и источник.

Обмен информации производится с помощью сигналов, являющихся ее материальным носителем. Источниками информации могут быть любые объекты реального мира, обладающие определенными свойствами и способностями. Если объект относится к неживой природе, то он

вырабатывает сигналы, непосредственно отражающие его свойства. Если объектом-источником является человек, то вырабатываемые им сигналы могут не только непосредственно отражать его свойства, но и соответствовать тем знакам, которые человек вырабатывает с целью обмена информацией.

Принятую информацию получатель может использовать неоднократно. С этой целью он должен зафиксировать ее на материальном носителе (магнитном, фото, кино и др.). Процесс формирования исходного, несистематизированного массива информации называется ***накоплением*** информации. Среди записанных сигналов могут быть такие, которые отражают ценную или часто используемую информацию. Часть информации в данный момент времени особой ценности может не представлять, хотя, возможно, потребуется в дальнейшем.

Хранение информации — это процесс поддержания исходной информации в виде, обеспечивающем выдачу данных по запросам конечных пользователей в установленные сроки.

Обработка информации — это упорядоченный процесс ее преобразования в соответствии с алгоритмом решения задачи.

После решения задачи обработки информации результат должен быть выдан конечным пользователям в требуемом виде. Эта операция реализуется в ходе решения задачи ***выдачи*** информации. Выдача информации, как правило, производится с помощью внешних устройств ЭВМ в виде текстов, таблиц, графиков и пр.

Информационная техника представляет собой материальную основу информационной технологии, с помощью которой осуществляется сбор, хранение, передача и обработка информации. До середины XIX века, когда доминирующими были процессы сбора и накопления информации, основу информационной техники составляли перо, чернильница и бумага. Коммуникация (связь) осуществлялась путем направления пакетов (депеш). На смену «ручной» информационной технике в конце XIX века пришла «механическая» (пишущая машинка, телефон, телеграф и др.), что послужило базой для принципиальных изменений в технологии обработки информации. Понадобилось еще много лет, чтобы перейти от запоминания и передачи информации к ее переработке. Это стало возможно с появлением во второй половине нашего столетия такой информационной техники, как электронные вычислительные машины, положившие начало «компьютерной технологии».

Древние греки считали, что технология *(techne* — мастерство + *togos* — учение) — это мастерство (искусство) делать вещи. Более емкое определение это понятие приобрело в процессе индустриализации общества.

Технология — это совокупность знаний о способах и средствах проведения производственных процессов, при которых происходит качественное изменение обрабатываемых объектов.

Технологиям управляемых процессов свойственны упорядоченность и организованность, которые противопоставляются стихийным процессам. Исторически термин «технология» возник в сфере материального производства. Информационную технологию в данном контексте можно считать технологией использования программно-аппаратных средств вычислительной техники в данной предметной области.

Информационная технология — это совокупность методов, производственных процессов и программно-технических средств, объединенных в технологическую цепочку, обеспечивающую сбор, обработку, хранение, распространение и отображение информации с целью снижения трудоемкости процессов использования информационного ресурса, а также повышения их надежности и оперативности.

Информационные технологии характеризуются следующими основными свойствами:

- предметом (объектом) обработки (процесса) являются ***данные;***
- целью процесса является получение ***информации;***
- средствами осуществления процесса являются программные, аппаратные и программно-аппаратные ***вычислительные комплексы;***
- процессы обработки данных разделяются на ***операции*** в соответствии с данной предметной областью;
- выбор управляющих воздействий на процессы должен осуществляться ***лицами, принимающими решение;***
- критериями оптимизации процесса являются ***своевременность доставки*** информации пользователю, ее ***надежность, достоверность, полнота.***

Из всех видов технологий информационная технология сферы управления предъявляет самые высокие требования к «человеческому фактору», оказывая принципиальное влияние на квалификацию работника, содержание его труда, физическую и умственную нагрузку, профессиональные перспективы и уровень социальных отношений.

ЭВМ КАК СРЕДСТВО ОБРАБОТКИ ИНФОРМАЦИИ

При рассмотрении ЭВМ как средства обработки информации важную роль играют понятие архитектуры ЭВМ, классификация ЭВМ, структура и принципы функционирования ЭВМ, а также основные характеристики вычислительной техник.

Понятие архитектуры ЭВМ

С середины 60-х годов существенно изменился подход к созданию вычислительных машин. Вместо независимой разработки аппаратуры и некоторых средств математического обеспечения стала проектироваться система, состоящая из совокупности *аппаратных (hardware)* и *программных (software)* средств. При этом на первый план выдвинулась концепция их взаимодействия. Так возникло принципиально новое понятие — архитектура ЭВМ.

Под ***архитектурой ЭВМ*** понимается совокупность общих принципов организации аппаратно-программных средств и их характеристик, определяющая функциональные возможности ЭВМ при решении соответствующих классов задач.

Архитектура ЭВМ охватывает широкий круг проблем, связанных с построением комплекса аппаратных и программных средств и учитывающих множество факторов. Среди этих факторов важнейшими являются: стоимость, сфера применения, функциональные возможности, удобство эксплуатации, а одним из главных компонентов архитектуры являются аппаратные средства. Основные компоненты архитектуры ЭВМ можно представить в виде схемы, показанной на рис. 2.

Рис. 2. Основные компоненты архитектуры ЭВМ

Архитектуру вычислительного средства следует отличать от его структуры. Структура вычислительного средства определяет его конкретный состав на некотором уровне детализации (устройства, блоки узлы и т. д.) и описывает связи внутри средства во всей их полноте. Архитектура же определяет правила взаимодействия составных частей вычислительного средства, описание которых выполняется в той мере, в какой это необходимо для формирования правил их взаимодействия. Она регламентирует не все связи, а наиболее важные, которые должны быть известны для более

грамотного использования данного средства.

Так, пользователю ЭВМ безразлично, на каких элементах выполнены электронные схемы, схемно или программно реализуются команды и т. д. Важно другое: как те или иные структурные особенности ЭВМ связаны с возможностями, предоставляемыми пользователю, какие альтернативы реализованы при создании машины и по каким критериям принимались решения, как связаны между собой характеристики отдельных устройств, входящих в состав ЭВМ, и какое влияние они оказывают на общие характеристики машины. Иными словами, архитектура ЭВМ действительно отражает круг проблем, относящихся к общему проектированию и построению вычислительных машин и их программного обеспечения.

Классификация ЭВМ

Чтобы судить о возможностях ЭВМ, их принято разделять на группы по определенным признакам, т. е. классифицировать. Сравнительно недавно классифицировать ЭВМ по различным признакам не составляло большого труда. Важно было только определить признак классификации, например: по назначению, по габаритам, по производительности, по стоимости, по элементной базе и т. д.

С развитием технологии производства ЭВМ классифицировать их стало все более затруднительно, ибо стирались грани между такими важными характеристиками, как производительность, емкость внутренней и внешней памяти, габариты, вес, энергопотребление и др. Например, персональный компьютер, для размещения которого достаточно стола, имеет практически такие же возможности и технические характеристики, что и достаточно совершенная в недавнем прошлом ЭВМ Единой системы (ЕС), занимающая машинный зал в сотни квадратных метров. Поэтому разделение ЭВМ по названным признакам нельзя воспринимать как классификацию по техническим параметрам. Это, скорее, эвристический подход, где большой вес имеет предполагаемая сфера применения компьютеров.

С этой точки зрения ***классификацию*** вычислительных машин по таким показателям, как ***габариты и производительность,*** можно представить следующим образом:

- сверхпроизводительные ЭВМ и системы (супер-ЭВМ);
- большие ЭВМ (универсальные ЭВМ общего назначения);
- средние ЭВМ;
- малые или мини-ЭВМ;
- микро-ЭВМ;
- персональные компьютеры;
- микропроцессоры.

Отметим, что понятия «большие», «средние» и «малые» для отечественных ЭВМ весьма условны и не соответствуют подобным категориям зарубежных ЭВМ.

Исторически первыми появились **большие ЭВМ (универсальные ЭВМ общего назначения)**, элементная база которых прошла путь от электронных ламп до схем со сверхвысокой степенью интеграции. В процессе эволюционного развития больших ЭВМ можно выделить отдельные периоды, связываемые с пятью поколениями ЭВМ.

Поколение ЭВМ определяется элементной базой (лампы, полупроводники, микросхемы различной степени интеграции), архитектурой и вычислительными возможностями.

Основное назначение больших ЭВМ — выполнение работ, связанных с обработкой и хранением больших объемов информации, проведением сложных расчетов и исследований в ходе решения вычислительных и информационно-логических задач. Такими машинами, как правило, оснащаются вычислительные центры, используемые совместно несколькими организациями. Большие машины составляли основу парка вычислительной техники до середины 70-х годов и успешно эксплуатируются поныне. К ним относятся большинство моделей фирмы IBM (семейства 360,370,390) и их отечественные аналоги ЕС ЭВМ.

В настоящее время высказываются полярные мнения о перспективах развития больших машин. Согласно одному из них, возможности больших машин полностью перекрываются, с одной стороны, супер-ЭВМ, а с другой — мини-ЭВМ и, выработав свой ресурс, этот класс прекратит свое существование. Другая сторона убеждает в необходимости развития универсальных больших и супер-ЭВМ, которые обладают способностью работать одновременно с большим количеством пользователей, создавать гигантские базы данных и обеспечивать эффективную вычислительную работу. К этому следует добавить, что большие ЭВМ обеспечивают устойчивость вычислительного процесса, безопасность информации и низкую стоимость ее обработки.

Производительность больших ЭВМ порой оказывается недостаточной для ряда приложений, например, таких как прогнозирование метеообстановки, ядерная энергетика, оборона и т. д. Эти обстоятельства стимулировали создание сверхбольших или суперЭВМ. Такие машины обладают колоссальным быстродействием в миллиарды операций в секунду, основанном на выполнении параллельных вычислений и использовании многоуровневой иерархической структуры ЗУ(запоминающих устройств), требуют для своего размещения специальных помещений и крайне сложны в эксплуатации. Стоимость отдельной ЭВМ такого класса достигает десятков

миллионов долларов. Представители этого класса ЭВМ — компьютеры фирм Cray Research, Control Data Corporation (CDC) и отечественные супер-ЭВМ семейства Эльбрус.

Средние ЭВМ представляют некоторый интерес в историческом плане. На определенном этапе развития ЭВМ, когда их номенклатура и, соответственно, возможности были ограниченными, появление средних машин было закономерным. Вычислительные машины этого класса обладают несколько меньшими возможностями, чем большие ЭВМ, но зато им присуща и более низкая стоимость. Они предназначены для использования всюду, где приходится постоянно обрабатывать достаточно большие объемы информации с приемлемыми временными затратами. В настоящее время трудно определить четкую грань между средними ЭВМ и большими с одной стороны и малыми — с другой. К средним могут быть отнесены некоторые модели ЕС ЭВМ, например: ЕС-1036, ЕС-1130, ЕС-1120. За рубежом средние ЭВМ выпускают фирмы IBM (International Business Machinary), DEC (Digital Equipment Corporation), Hewlett Packard, COMPAREX и др.

Малые ЭВМ составляют самый многочисленный и быстроразвивающийся класс ЭВМ. Их популярность объясняется малыми размерами, низкой стоимостью (по сравнению с большими и средними ЭВМ) и универсальными возможностями.

Класс **мини-ЭВМ** появился в 60-е годы (12-разрядная ЭВМ PD5-5 фирмы DEC). Их появление было обусловлено развитием элементной базы и избыточностью ресурсов больших и средних ЭВМ для ряда приложений. Для мини-ЭВМ характерно представление данных с узким диапазоном значений (машинное слово — 2 байта), использование принципа магистральности в архитектуре и более простое взаимодействие человека и ЭВМ. Такие машины широко применяются для управления сложными видами оборудования, создания систем автоматизированного проектирования и гибких производственных систем. К мини-ЭВМ относятся машины серии PDP (затем VAX) фирмы DEC и их отечественные аналоги — модели семейства малых ЭВМ (СМ ЭВМ).

При переходе от схем с малой и средней степенями интеграции к интегральным микросхемам с большой и сверхбольшой степенями интеграции оказалось возможным создание на одной БИС или СБИС функционально законченного устройства обработки информации, выполняющего функции процессора. Такое устройство принято называть ***микропроцессором.*** Изобретение микропроцессора привело к появлению еще одного класса ЭВМ — **микро-ЭВМ.** Определяющим признаком микро-ЭВМ является наличие одного или нескольких микропроцессоров. Создание микропроцессора не только изменило центральную часть ЭВМ, но и привело

к необходимости разработки малогабаритных устройств ее периферийной части. Микро-ЭВМ, благодаря малым размерам, высокой производительности, повышенной надежности и небольшой стоимости нашли широкое распространение во всех сферах народного хозяйства и оборонного комплекса. С появлением микропроцессоров и микро-ЭВМ становится возможным создание так называемых ***интеллектуальных терминалов,*** выполняющих сложные процедуры предварительной обработки информации.

Успехи в развитии микропроцессоров и микро-ЭВМ привели к появлению **персональных ЭВМ** (ПЭВМ), предназначенных для индивидуального обслуживания пользователя и ориентированных на решение различных задач неспециалистами в области вычислительной техники. Все оборудование персональной ЭВМ размещается в пределах стола.

ПЭВМ, выпускаемые в сотнях тысяч и миллионах экземпляров, вносят коренные изменения в формы использования вычислительных средств, в значительной степени расширяют масштабы их применения. Они широко используются как для поддержки различных видов профессиональной деятельности (инженерной, административной, производственной, литературной, финансовой и др.), так и в быту, например для обучения и досуга.

Персональный компьютер позволяет эффективно выполнять научно-технические и финансово-экономические расчеты, организовывать базы данных, подготавливать и редактировать документы и любые другие тексты, вести делопроизводство, обрабатывать графическую информацию и т. д. Выполнение многих из указанных функций поддерживается многочисленными эффективными универсальными функциональными пакетами программ.

На основе ПЭВМ создаются ***автоматизированные рабочие места (АРМ)*** для представителей разных профессий (конструкторов, технологов, административного аппарата и др.).

Рынок персональных и микро-ЭВМ непрерывно расширяется за счет поставок ведущих мировых фирм: IBM, Compaq, Hewlett Packard, Apple (США), Siemens (Германия), ICL (Англия) и др.

БАЗОВАЯ АППАРАТНАЯ КОНФИГУРАЦИЯ

Персональный компьютер – универсальная техническая система. Его *конфигурацию* (состав оборудования) можно гибко изменять по мере необходимости. Тем не менее, существует понятие ***базовой конфигурации,*** которую считают типовой. В таком комплекте компьютер обычно

поставляется. Понятие базовой конфигурации может меняться. В настоящее время в базовой конфигурации рассматривают четыре устройства (рис. 3):

- системный блок;
- монитор;
- клавиатуру;
- мышь.

Помимо компьютеров с базовой конфигурации всё большее распространение получают мультимедийные компьютеры, оснащенные устройством чтения компакт-дисков, колонками и микрофоном.

Рис. 3. Конфигурация мультимедийного компьютера

Системный блок

Системный блок представляет собой основной узел, внутри которого установлены наиболее важные компоненты. Устройства, находящиеся внутри системного блока, называют ***внутренними,*** а устройства, подключаемые к нему снаружи, называют ***внешними***. Внешние дополнительные устройства, предназначенные для ввода, вывода и длительного хранения данных, также называют ***периферийными***.

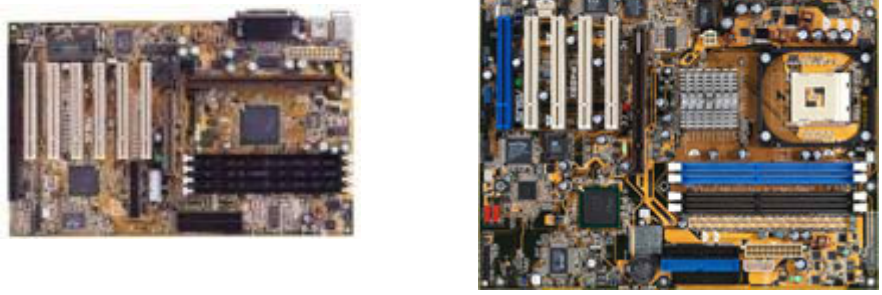
По внешнему виду системные блоки различаются формой корпуса. Выбор того или иного типа корпуса определяется вкусом и потребностями модернизации компьютера. Наиболее оптимальным типом корпуса для большинства пользователей является корпус типа *mini tower*.

Кроме формы, для корпуса важен параметр, называемый *форм-фактором.* От него зависят требования к размещаемым устройствам. В настоящее время в основном используются корпуса двух форм-факторов: *AT и ATX* . Форм-фактор корпуса должен быть обязательно согласован с форм-фактором главной (системной) платы компьютера, так называемой *материнской платы*.

Материнская плата

Материнская плата (mother board) – основная плата персонального компьютера, представляющая из себя лист стеклотекстолита, покрытый медной фольгой. Путем травления фольги получают тонкие медные проводники соединяющие электронные компоненты. На материнской плате размещаются:

- ***процессор*** – основная микросхема, выполняющая большинство математических и логических операций;
- ***шины*** – наборы проводников, по которым происходит обмен сигналами между внутренними устройствами компьютера;
- ***оперативная память (оперативное запоминающее устройство, ОЗУ)*** – набор микросхем, предназначенных для временного хранения данных, когда компьютер включен;
- ***ПЗУ (постоянное запоминающее устройство)*** – микросхема, предназначенная для длительного хранения данных, в том числе и когда компьютер выключен;
- ***микропроцессорный комплект (чипсет)*** – набор микросхем, управляющих работой внутренних устройств компьютера и определяющих основные функциональные возможности материнской платы;
- разъемы для подключения дополнительных устройств (слоты).



а б

Рис. 4. Материнские платы: а) Pentium III P3B-F фирмы ASUSTeK; б) ASUSTeK P4G8X-Deluxe Socket478 фирмы ASUSTeK

Менялись микропроцессоры, рождались и умирали системные и локальные шины, а вид и размеры материнской платы практически не менялись с 1984 г. Например, размер оригинальной материнской платы IBM PC/AT под названием Baby-AT был равен 217 на 331 мм, а размеры современной материнской платы P3B-F равны 192 мм на 304 мм. (рис. 4).

Некоторые из них имеют отличительный признак – наличие гнезда для подключения микропроцессора с нулевым усилием сопряжения – ZIF (Zero Insertion Force) типа Socket 7. В данное гнездо можно ставить как процессор Intel Pentium, так и процессоры Cyrix 6x86, AMD K5 и K6. Более

современные платы имеют разъем Slot 1 или Slot 2 для подключения процессоров Pentium II и Pentium III.

Процессор

Процессор (микропроцессор, центральный процессор, CPU) – основная микросхема компьютера, в которой и производятся все вычисления. Он представляет из себя большую микросхему (например, размеры микропроцессора Pentium примерно 5*5*0,5 см), которую можно легко найти на материнской плате. На процессоре установлен большой медный ребристый радиатор, охлаждаемый вентилятором. Конструктивно процессор состоит из ячеек, в которых данные могут не только храниться, но и изменяться. Внутренние ячейки процессора называют ***регистрами***. Важно также отметить, что данные, попавшие в некоторые регистры, рассматриваются не как данные, а как команды, управляющие обработкой данных в других регистрах. Среди регистров процессора есть и такие, которые в зависимости от своего содержания способны модифицировать исполнение команд. Таким образом, управляя засылкой данных в разные регистры процессора, можно управлять обработкой данных. На этом и основано исполнение программ.

С остальными устройствами компьютера, и в первую очередь с оперативной памятью, процессор связан несколькими группами проводников, называемых *шинами.* Основных шин три: *шина данных, адресная тина* и *командная шина.*

Адресная шина. У процессоров Intel Pentium (а именно они наиболее распространены в персональных компьютерах) адресная шина 32-разрядная, то есть состоит из 32 параллельных линий. В зависимости от того, есть напряжение на какой-то из линий или нет, говорят, что на этой линии выставлена единица или ноль. Комбинация из 32 нулей и единиц образует 32-разрядный адрес, указывающий на одну из ячеек оперативной памяти. К ней и подключается процессор для копирования данных из ячейки в один из своих регистров.

Шина данных. По этой шине происходит копирование данных из оперативной памяти в регистры процессора и обратно. В компьютерах, собранных на базе процессоров Intel Pentium, шина данных 64-разрядная, то есть состоит из 64 линий, по которым за один раз на обработку поступают сразу 8 байтов.

Шина команд. Для того чтобы процессор мог обрабатывать данные, ему нужны команды. Он должен знать, что следует сделать с теми байтами, которые хранятся в его регистрах. Эти команды поступают в процессор тоже из оперативной памяти, но не из тех областей, где хранятся массивы данных, а оттуда, где хранятся программы. Команды тоже представлены в виде

байтов. Самые простые команды укладываются в один байт, однако есть и такие, для которых нужно два, три и более байтов. В большинстве современных процессоров шина команд 32-разрядная (например, в процессоре Intel Pentium), хотя существуют 64-разрядные процессоры и даже 128-разрядные.

Система команд процессора. В процессе работы процессор обслуживает данные, находящиеся в его регистрах, в поле оперативной памяти, а также данные, находящиеся во внешних портах процессора. Часть данных он интерпретирует непосредственно как данные, часть данных – как адресные данные, а часть – как команды. Совокупность всех возможных команд, которые может выполнить процессор над данными, образует так называемую *систему команд процессора*. Процессоры, относящиеся к одному семейству, имеют одинаковые или близкие системы команд. Процессоры, относящиеся к разным семействам, различаются по системе команд и невзаимозаменяемыми.

Совместимость процессоров. Если два процессора имеют одинаковую систему команд, то они полностью совместимы на программном уровне. Это означает, что программа, написанная для одного процессора, может исполняться и другим процессором. Процессоры, имеющие разные системы команд, как правило, несовместимы или ограниченно совместимы на программном уровне.

а) б) в)

Рис. 5. Микропроцессоры, разработанные фирмами Intel и AMD: а) CPU Intel Pentium 4 2.8 ГГц; б) CPU Intel Pentium III 550 МГц ; в) CPU AMD K6-3-450 МГц

Группы процессоров, имеющих ограниченную совместимость, рассматривают как ***семейства процессоров***. Так, например, все процессоры Intel Pentium относятся к так называемому семейству x86. Родоначальником этого семейства был 16-разрядный процессор Intel 8086, на базе которого собиралась первая модель компьютера *IBM PC*. Впоследствии выпускались процессоры Intel 80286, Intel 80386, Intel 80486, Intel Pentium 60,66,75,90,100,133; несколько моделей процессоров Intel Pentium MMX, модели Intel Pentium Pro, Intel Pentium II, Intel Celeron, Intel Xeon, Intel

Pentium III (см. рис. 5,а), Intel Pentium IV и другие. Все эти модели, и не только они, а также многие модели процессоров компаний AMD (см. рис. 5,б) и Cyrix относятся к семейству x86 и обладают совместимостью по принципу «сверху вниз».

Основные параметры процессоров. Основными параметрами процессоров являются: *рабочее напряжение, разрядность, рабочая тактовая частота, коэффициент внутреннего умножения тактовой частоты* и размер *кэш-памяти.*

Рабочее напряжение процессора обеспечивает материнская плата, поэтому разным маркам процессоров соответствуют разные материнские платы (их надо выбирать совместно). По мере развития процессорной техники происходит постепенное понижение рабочего напряжения. Ранние модели процессоров x86 имели рабочее напряжение 5 В. С переходом к процессорам Intel Pentium оно было понижено до 3,3 В, а в настоящее время оно составляет менее 3 В. Причем ядро процессора питается пониженным напряжением 2,2 В. Понижение рабочего напряжения позволяет уменьшить расстояния между структурными элементами в кристалле процессора до десятитысячных долей миллиметра, не опасаясь электрического пробоя. Пропорционально квадрату напряжения уменьшается и тепловыделение в процессоре, а это позволяет увеличивать его производительность без угрозы перегрева.

Разрядность процессора показывает, сколько бит данных он может принять и обработать в своих регистрах за один раз *(за один такт)*. Первые процессоры x86 были 16-разрядными. Начиная с процессора 80386 они имеют 32-разрядную архитектуру. Современные процессоры семейства Intel Pentium остаются 32-разрядными, хотя и работают с 64-разрядной шиной данных (разрядность процессора определяется не разрядностью шины данных, а разрядностью командной шины).

Тактовые сигналы процессор получает от материнской платы, которая, в отличие от процессора, представляет собой не кристалл кремния, а большой набор проводников и микросхем. По чисто физическим причинам материнская плата не может работать со столь высокими частотами, как процессор. Сегодня ее предел составляет 100-133 МГц. Для получения более высоких частот в процессоре происходит ***внутреннее умножение частоты*** на коэффициент 3; 3,5; 4; 4,5; 5 и более.

Обмен данными внутри процессора происходит в несколько раз быстрее, чем обмен с другими устройствами, например с оперативной памятью. Для того чтобы уменьшить количество обращений к оперативной памяти, внутри процессора создают буферную область – так называемую ***кэш-память***. Это как бы «сверхоперативная память». Когда процессору нужны данные, он

сначала обращается в кэш-память, и только если там нужных данных нет, происходит его обращение в оперативную память. Принимая блок данных из оперативной памяти, процессор заносит его одновременно и в кэш-память. «Удачные» обращения в кэш-память называют *попаданиями в кэш.* Процент попаданий тем выше, чем больше размер кэш-памяти, поэтому высокопроизводительные процессоры комплектуют повышенным объемом кэш-памяти.

Кэш-память третьего уровня выполняют на быстродействующих микросхемах типа *SRAM* и размещают на материнской плате вблизи процессора. Ее объемы могут достигать нескольких Мбайт, но работает она на частоте материнской платы.

Шинные интерфейсы материнской платы

Связь между всеми собственными и подключаемыми устройствами материнской платы выполняют ее шины и логические устройства, размещенные в микросхемах микропроцессорного комплекта (чипсета). От архитектуры этих элементов во многом зависит производительность компьютера.

ISA. Историческим достижением компьютеров платформы *IBM PC* стало внедрение почти двадцать лет назад архитектуры, получившей статус *промышленного стандарта ISA (Industry Standard Architecture).* Она не только позволила связать все устройства системного блока между собой, но и обеспечила простое подключение новых устройств через стандартные разъемы (слоты). Пропускная способность шины, выполненной по такой архитектуре, составляет до 5,5 Мбайт/с, но, несмотря на низкую пропускную способность, эта шина продолжает использоваться в компьютерах для подключения сравнительно «медленных» внешних устройств, например звуковых карт и модемов.

EISA. Расширением стандарта *ISA* стал стандарт *EISA (Extended ISA),* отличающийся увеличенным разъемом и увеличенной производительностью (до 32 Мбайт/с). Как и *ISA, в* настоящее время данный стандарт считается устаревшим. После 2000 года выпуск материнских плат с разъемами *ISA/EISA* и устройств, подключаемых к ним, прекращается.

VLB. Название интерфейса переводится как *локальная шина стандарта VESA (VESA Local Bus).* Понятие «локальной шины» впервые появилось в конце 80-х годов. Оно связано тем, что при внедрении процессоров третьего и четвертого поколений (Intel 80386 и Intel 80486) частоты основной шины (в качестве основной использовалась шина *ISA/EISA)* стало недостаточно для обмена между процессором и . оперативной памятью. Локальная шина, имеющая повышенную частоту, связала между собой процессор и память в обход основной шины. Впоследствии в эту шину «врезали» интерфейс для

подключения видеоадаптера, который тоже требует повышенной пропускной способности, – так появился стандарт *VLB,* который позволил поднять тактовую частоту локальной шины до 50 МГц и обеспечил пиковую пропускную способность до 130 Мбайт/с.

Основным недостатком интерфейса *VLB* стало то, что предельная частота локальной шины и, соответственно, ее пропускная способность зависят от числа устройств, подключенных к шине. Так, например, при частоте 50 Мгц к шине может быть подключено только одно устройство (видеокарта). Для сравнения скажем, что при частоте 40 Мгц возможно подключение двух, а при частоте 33 МГц – трех устройств.

PCI. Интерфейс *PCI (Peripheral Component Interconnect – стандарт подключения, внешних компонентов)* был введен в персональных компьютерах, выполненных на базе процессоров Intel Pentium. По своей сути это тоже интерфейс локальной шины, связывающей процессор с оперативной памятью, в которую врезаны разъемы для р. подключения внешних устройств. Для связи с основной шиной компьютера *(ISA/EISA)* используются специальные интерфейсные преобразователи – *мосты PCI (PCI Bridge)*. В современных компьютерах функции моста *PCI* выполняют микросхемы микропроцессорного комплекта (чипсета).

FSB. Шина *PCI,* появившаяся в компьютерах на базе процессоров Intel Pentium как локальная шина, предназначенная для связи процессора с оперативной памятью, недолго оставалась в этом качестве. Сегодня она используется только как шина для подключения внешних устройств, а для связи процессора и памяти, начиная с процессора Intel Pentium Pro используется специальная шина, получившая название *Front Side Bus (FSB)*. Эта шина работает на очень высокой частоте 100-125 МГц. В настоящее время внедряются материнские платы с частотой шины *FSB* 133 МГц и ведутся разработки плат с частотой до 200 МГц. Частота шины *FSB* является одним из основных потребительских параметров – именно он и указывается в спецификации материнской платы. Пропускная способность шины *FSB* при частоте 100 МГц составляет порядка 800 Мбайт/с.

AGP. Видеоадаптер – устройство, требующее особенно высокой скорости передачи данных. Как при внедрении локальной шины *VLB,* так и при внедрении локальной шины *PCI* видеоадаптер всегда был первым устройством, «врезаемым» в новую шину. Сегодня параметры шины *PCI* уже не соответствуют требованиям видеоадаптеров, поэтому для них разработана отдельная шина, получившая название *AGP (Advanced Graphic Port – усовершенствованный графический порт)*. Частота этой шины соответствует частоте шины *PCI* (33 МГц или 66 МГц), но она имеет много более высокую пропускную способность – до 1066 Мбайт/с (в режиме четырехкратного умножения).

PCMCIA *(Personal Computer Memory Card International Association – стандарт международной ассоциации производителей плат памяти для персональных компьютеров).* Этот стандарт определяет интерфейс подключения плоских карт памяти небольших размеров и используется в портативных персональных компьютерах.

USB *(Universal Serial Bus – универсальная последовательная магистраль).* Это одно из последних нововведений в архитектурах материнских плат. Этот стандарт определяет способ взаимодействия компьютера с периферийным оборудованием. Он позволяет подключать до 256 различных устройств, имеющих последовательный интерфейс. Устройства могут включаться цепочками (каждое следующее устройство подключается к предыдущему). Производительность шины *USB* относительно невелика и составляет до 1,5 Мбит/с, но для таких устройств, как клавиатура, мышь, модем, джойстик и т. п., этого достаточно. Удобство шины состоит в том, что она практически исключает конфликты между различным оборудованием, позволяет подключать и отключать устройства в «горячем режиме» (не выключая компьютер) и позволяет объединять несколько компьютеров в простейшую локальную сеть без применения специального оборудования и программного обеспечения.

Параметры микропроцессорного комплекта (чипсета) в наибольшей степени определяют свойства и функции материнской платы. В настоящее время большинство чипсетов материнских плат выпускаются на базе двух микросхем, получивших название «северный мост» и «южный мост».

«Северный мост» управляет взаимосвязью четырех устройств: процессора, оперативной памяти, порта *AGP* и шины *PCI.* Поэтому его также называют ***четырехпортовым контроллером***.

«Южный мост» называют также ***функциональным контроллером***. Он выполняет функции контроллера жестких и гибких дисков, функции моста *ISA – PCI,* контроллера клавиатуры, мыши, шины *USB* и т. п.

Оперативная память

Оперативная память (RAM – Random Access Memory) – это массив кристаллических ячеек, способных хранить данные. Существует много различных типов оперативной памяти, но с точки зрения физического принципа действия различают *динамическую память (DRAM)* и *статическую память (SRAM).*

Ячейки динамической памяти (DRAM) можно представить в виде микроконденсаторов, способных накапливать заряд на своих обкладках. Это наиболее распространенный и экономически доступный тип памяти. Недостатки этого типа связаны, во-первых, с тем, что как при заряде, так и

при разряде конденсаторов неизбежны переходные процессы, то есть запись данных происходит сравнительно медленно. Второй важный недостаток связан с тем, что заряды ячеек имеют свойство рассеиваться в пространстве, причем весьма быстро. Если оперативную память постоянно не «подзаряжать», утрата данных происходит через несколько сотых долей секунды. Для борьбы с этим явлением в компьютере происходит постоянная ***регенерация (освежение, подзарядка)*** ячеек оперативной памяти. Регенерация осуществляется несколько десятков раз в секунду и вызывает непроизводительный расход ресурсов вычислительной системы.

Одна адресуемая ячейка содержит восемь двоичных ячеек, в которых можно сохранить 8 бит, то есть один байт данных. Таким образом, адрес любой ячейки памяти можно выразить четырьмя байтами.

Оперативная память в компьютере размещается на стандартных панельках, называемых ***модулями***. Модули оперативной памяти вставляют в соответствующие разъемы на материнской плате. Если к разъемам есть удобный доступ, то операцию можно выполнять своими руками. Если удобного доступа нет, может потребоваться неполная разборка узлов системного блока, и в таких случаях операцию поручают специалистам.

Конструктивно модули памяти имеют два исполнения – ***однорядные*** *(SIMM-модули)* и ***двухрядные*** *(DIMM-модули)* (см. рис. 6). На компьютерах с процессорами Pentium однорядные модули можно применять только парами (количество разъемов для их установки на материнской плате всегда четное), а *DIMM-модули* можно устанавливать по одному. Многие модели материнских плат имеют разъемы как того, так и другого типа, но комбинировать на одной плате модули разных типов нельзя.

а  б 

Рис. 6. Модули памяти: а – SIMM; б – DIMM

Основными характеристиками модулей оперативной памяти являются объем памяти и время доступа. *SIMM*-модули поставляются объемами 4,8,16,32 Мбайт, а *DIMM*-модули – 16, 32, 64, 128 Мбайт и более. Время доступа показывает, сколько времени необходимо для обращения к ячейкам памяти – чем оно меньше, тем лучше. Время доступа измеряется в миллиардных долях секунды *(наносекундах,* нс). Типичное время доступа к оперативной памяти для *SIMM*-модулей – 50-70 нс. Для современных *DIMM*-модулей оно составляет 7-10 нс.

Жесткий диск

Жесткий диск – основное устройство для долговременного хранения больших объемов данных и программ (см. рис. 7). На самом деле это не один диск, а группа соосных дисков, имеющих магнитное покрытие и вращающихся с высокой скоростью. Таким образом, этот «диск» имеет не две поверхности, как должно быть у обычного плоского диска, а 2n поверхностей, где *n* – число отдельных дисков в группе.

Рис. 7. Жесткий диск MPD3043AT U_DMA фирмы FUJITSU емкостью 4.3 Gb (IDE)

Над каждой поверхностью располагается головка, предназначенная для чтения-записи данных. При высоких скоростях вращения дисков (90 об/с) в зазоре между головкой и поверхностью образуется аэродинамическая подушка, и головка парит над магнитной поверхностью на высоте, составляющей несколько тысячных долей миллиметра. При изменении силы тока, протекающего через головку, происходит изменение напряженности динамического магнитного поля в зазоре, что вызывает изменения в стационарном магнитном поле ферромагнитных частиц, образующих покрытие диска. Так осуществляется запись данных на магнитный диск.

Дисковод гибких дисков

Информация на жестком диске может храниться годами, однако иногда требуется ее перенос с одного компьютера на другой. Несмотря на свое название, жесткий диск является весьма хрупким прибором, чувствительным к перегрузкам, ударам и толчкам. Теоретически, переносить информацию с одного рабочего места на другое путем переноса жесткого диска возможно, и в некоторых случаях так и поступают, но все-таки этот прием считается нетехнологичным, поскольку требует особой аккуратности и определенной квалификации.

Для оперативного переноса небольших объемов информации используют так называемые ***гибкие магнитные диски (дискеты)*** (рис. 8), которые вставляют в специальный накопитель – ***дисковод*** (рис. 9). Приемное отверстие накопителя находится на лицевой панели системного блока. Правильное направление подачи гибкого диска отмечено стрелкой на его пластиковом кожухе.

Рис. 8. Дискета размером 3,5 дюйма емкостью 1,4 Мбайт

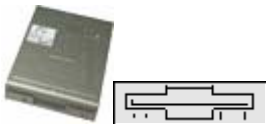


Рис. 9. Дисковод гибких дисков на 3,5 дюйма

Основными параметрами гибких дисков являются: технологический размер (измеряется в дюймах), плотность записи (измеряется в кратных единицах) и полная емкость.

Гибкие диски размером 3,5 дюйма выпускают с 1980 года. Односторонний диск ***обычной плотности*** имел емкость 180 Кбайт, двусторонний – 360 Кбайт, а ***двусторонний двойной плотности*** – 720 Кбайт. Ныне стандартными считают диски размером 3,5 дюйма ***высокой плотности***. Они имеют емкость **1440 Кбайт (1,4 Мбайт)** и маркируются буквами ***HD (high density – высокая плотность).***

Дисковод компакт-дисков CD-ROM

В период 1994-1995 годах в базовую конфигурацию персональных компьютеров перестали включать дисководы гибких дисков диаметром 5,25 дюйма, но вместо них стандартной стала считаться установка дисковода *CD-ROM,* имеющего такие же внешние размеры (рис. 10).

Рис. 10. Дисковод CD-ROM с производительностью 32x фирмы SAMSUNG

Аббревиатура ***CD-ROM (Compact Disc Read-Only Memory)*** переводится на русский язык как ***постоянное запоминающее устройство на основе компакт-диска***. Принцип действия этого устройства состоит в считывании числовых данных с помощью лазерного луча, отражающегося от поверхности диска. Цифровая запись на компакт-диске отличается от записи на магнитных дисках очень высокой плотностью, и стандартный компакт-диск может хранить примерно 650 Мбайт данных.

Большие объемы данных характерны для ***мультимедийной информации*** (графика, музыка, видео), поэтому дисководы *CD-ROM* относят к аппаратным средствам мультимедиа. Программные продукты, распространяемые на лазерных дисках, называют *мультимедийными изданиями.* Сегодня мультимедийные издания завоевывают все более

прочное место среди других традиционных видов изданий. Так, например, существуют книги, альбомы, энциклопедии и даже периодические издания (электронные журналы), выпускаемые на *CD-ROM*.

Монитор

Монитор – устройство визуального представления данных (рис. 11, 12). Это не единственно возможное, но главное устройство вывода. Его основными потребительскими параметрами являются: размер и шаг маски экрана, максимальная частота регенерации изображения, класс защиты.

По способу формирования изображения мониторы делятся на ***жидкокристаллические (LCD)*** и построенные на основе ***электронно-лучевой трубки (CRT)***.

Рис. 11. CRT-монитор Samtron 17" фирмы Samsung

Рис. 12. Монитор LCD 15" Sony N50

Клавиатура

Клавиатура – клавишное устройство управления персональным компьютером (рис. 13). Служит для ввода *алфавитно-цифровых (знаковых)* данных, а также команд управления. Комбинация монитора и клавиатуры обеспечивает простейший *интерфейс пользователя. С* помощью клавиатуры управляют компьютерной системой, а с помощью монитора получают от нее отклик.

Известно два типа клавиатур – *клавиатура с **механическими** и **мембранными переключателями***. Переключатель первого типа это обычный механический датчик, традиционное устройство известное уже несколько десятилетий. Второй тип датчика устроен несколько сложнее. Переключатель данного типа представляет из себя набор мембран, при нажатии на клавишу верхняя мембрана прогибается и через специальное отверстие в изолирующей мембране замыкается на нижнюю мембрану. Как правило, предпочтение отдается клавиатуре с механическими датчиками.

Они выдерживают многолетнюю эксплуатацию, надежны и поддаются ремонту в случае необходимости.

Из данного объяснения ясно, что каждой клавише присвоен уникальный цифровой код и существуют специальные таблицы кодировки клавиатуры. Например, кодовая таблица США имеет номер 437 (как правило, она записана в специальную микросхему – знакогенератор процессора), а кодовая страница России имеет номер 866. Для смены кодировки клавиатуры применяются специальные программы – клавиатурные драйверы. Современные клавиатуры способны не только передавать данные в процессор, но и воспринимать команды от него.

Функциональные клавиши ***Дополнительная панель***

Алфавитно-цифровые клавиши ***Клавиши управления курсором***

Рис. 13. Группы клавиш стандартной клавиатуры

Для разных языков существуют различные схемы закрепления символов национальных алфавитов за конкретными алфавитно-цифровыми клавишами. Такие схемы называются ***раскладками клавиатуры***. Переключения между различными раскладками выполняются программным образом – это одна из функций операционной системы. Соответственно, способ переключения зависит от того, в какой операционной системе работает компьютер. Например, в системе Windows 98 для этой цели могут использоваться следующие комбинации: левая клавиша ALT+SHIFT или CTRL+SHIFT. При работе с другой операционной системой способ переключения можно установить по справочной системе той программы, которая выполняет переключение.

Мышь

Мышь – устройство управления манипуляторного типа. Представляет собой плоскую коробочку с двумя-тремя кнопками (рис. 14). Перемещение мыши по плоской поверхности синхронизировано с перемещением графического объекта *(указателя мыши)* на экране монитора.

Рис. 14. Устройство управления манипуляторного типа Logitech Pilot Wheel Mouse

Принцип действия. В отличие от рассмотренной ранее клавиатуры, мышь не является стандартным органом управления, и персональный компьютер не имеет для нее выделенного порта. Для мыши нет и постоянного выделенного прерывания, а базовые средства ввода и вывода *(BIOS)* компьютера, размещенные в постоянном запоминающем устройстве (ПЗУ), не содержат программных средств для обработки прерываний мыши.

ВИДЫ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ЭВМ

Назначением ЭВМ является выполнение программ. Программа содержит команды, определяющие порядок действий компьютера. Совокупность программ для компьютера образует ***программное обеспечение (ПО).*** По функциональному признаку различают следующие виды ПО:

- системное;
- прикладное.

Под ***системным (базовым)*** понимается программное обеспечение, включающее в себя операционные системы, сетевое ПО, сервисные программы, а также средства разработки программ (трансляторы, редакторы связей, отладчики и пр.).

Основные функции ***операционных систем*** (ОС) заключаются в управлении ресурсами (физическими и логическими) и процессами вычислительных систем. Физическими ресурсами являются: оперативная память, процессор, монитор, печатающее устройство, магнитные и оптические диски. К логическим ресурсам можно отнести программы, файлы, события и т. д. Под процессом понимается некоторая последовательность действий, предписанная соответствующей программой и используемыми ею данными.

В настоящее время существует большое количество ОС, разработанных для ЭВМ различных типов. На ЭВМ Единой Системы (ЕС ЭВМ), например, использовались такие операционные системы, как СВМ и ОС ЕС, на малых ЭВМ (СМ-4, СМ-1420 и др.) - ОС РВ и RSX-11. На персональных ЭВМ долгое время эксплуатировалась ОС-MS-DOS. В настоящее время получили распространение системы Windows 98/Me, Windows 2000, Linix.

Сетевое ПО предназначено для управления общими ресурсами в распределенных вычислительных системах: сетевыми накопителями на магнитных дисках, принтерами, сканерами, передаваемыми сообщениями и т. д. К сетевому ПО относят ОС, поддерживающие работу ЭВМ в сетевых конфигурациях (так называемые сетевые ОС), а также отдельные сетевые программы (пакеты), используемые совместно с обычными, не сетевыми ОС.

Например, большое распространение получили следующие сетевые ОС: NetWare 4.1 (фирма Novell), Windows NT Server 3.5 (фирма Microsoft) и LAN Server 4.0 Advanced (фирма IBM). Однако в последнее время лидирующие позиции начинает занимать ОС Windows 2000 Server фирмы Microsoft.

Для расширения возможностей операционных систем и предоставления набора дополнительных услуг используются ***сервисные программы.*** Их можно разделить на следующие группы:

- интерфейсные системы;
- оболочки операционных систем;
- утилиты.

Интерфейсные системы являются естественным продолжением операционной системы и модифицируют как пользовательский, так и программный интерфейсы, а также реализуют дополнительные возможности по управлению ресурсами ЭВМ. В связи с тем, что развитая интерфейсная система может изменить весь пользовательский интерфейс, часто их также называют операционными системами. Это относится, например, к Windows 3.11 и Windows 3.11 for WorkGroups (для рабочих групп).

Оболочки операционных систем, в отличие от интерфейсных систем, модифицируют только пользовательский интерфейс, предоставляя пользователю качественно новый интерфейс по сравнению с реализуемым операционной системой. Такие системы существенно упрощают выполнение часто запрашиваемых функций, например, таких операций с файлами, как копирование, переименование и уничтожение, а также предлагают пользователю ряд дополнительных услуг. В целом, программы-оболочки заметно повышают уровень пользовательского интерфейса, наиболее полно удовлетворяя потребностям пользователя.

На ПЭВМ широко используются такие программы-оболочки, как Norton Commander, FAR Manager и Windows Commander.

Утилиты предоставляют пользователям средства обслуживания компьютера и его ПО. Они обеспечивают реализацию следующих действий:

- обслуживание магнитных дисков;
- обслуживание файлов и каталогов;
- предоставление информации о ресурсах компьютера;
- шифрование информации;
- защита от компьютерных вирусов;
- архивация файлов и др.

Существуют ***отдельные утилиты,*** используемые для решения одного из перечисленных действий, и ***многофункциональные комплекты утилит***. В настоящее время для ПЭВМ среди многофункциональных утилит одним из наиболее совершенных является комплект утилит Norton Utilities. Существуют его версии для использования в среде DOS и Windows.

Средства разработки программ используются для разработки нового программного обеспечения как системного, так и прикладного.

Прикладным называется ПО, предназначенное для решения определенной целевой задачи из проблемной области. Часто такие программы называют *приложениями.*

Спектр проблемных областей в настоящее время весьма широк и включает в себя по крайней мере следующие: промышленное производство, инженерную практику, научные исследования, медицину, управление (менеджмент), делопроизводство, издательскую деятельность, образование и т. д.

Из всего разнообразия прикладного ПО выделяют группу наиболее распространенных программ (типовые пакеты и программы), которые можно использовать во многих областях человеческой деятельности.

К *типовому прикладному ПО* относят следующие программы:

- текстовые процессоры;
- табличные процессоры;
- системы иллюстративной и деловой графики (графические процессоры);
- системы управления базами данных;
- экспертные системы;
- программы математических расчетов, моделирования и анализа экспериментальных данных.

Предлагаемые на рынке ПО приложения, в общем случае, могут быть выполнены как отдельные программы либо как интегрированные системы. Интегрированными системами обычно являются экспертные системы, программы математических расчетов, моделирования и анализа экспериментальных данных, а также офисные системы. Примером мощной и широко распространенной интегрированной системы является офисная система Microsoft Office.

Поскольку разработка ПО любого назначения, как правило, является довольно сложным и трудоемким процессом, дальнейший материал настоящего раздела посвятим общим вопросам разработки программ и инструментальному ПО.

Операционная система MS-DOS

Операционная система MS-DOS – это однопользовательская, однозадачная, не сетевая 16-разрядная операционная система (ОС), ориентированная на использование на ПЭВМ с микропроцессором Intel 8088(80286).

Краткая история появления данной операционной системы такова. В октябре 1980 г. менеджеры фирмы IBM занялись поисками ОС для своего 16-разрядного персонального компьютера (ПК), находящегося в стадии разработки. В тот период на ПЭВМ наиболее широко применялась ОС CP/M (Control Program for MicroComputers) фирмы Digital Research. Не достигнув приемлемых соглашений с Digital Research, фирма IBM обратилась к фирме Microsoft (президент – Билл Гейтс). В тот момент у Microsoft не было соответствующей ОС, но ей была известна небольшая фирма Seattle Computer Products, которая имела такую ОС. За 50 000$ Билл Гейтс приобрел права на эту ОС. В дальнейшем эта ОС послужила основой для MS-DOS. В ноябре 1980 г. Microsoft и IBM подписали договор на разработку ОС для IBM PC. В феврале 1981 г. появилась первая версия PC/MS-DOS, которая работала на IBM PC, в августе того же года – PC DOS 1.0 (эта версия была утверждена для применения на IBM PC).

Операционная система MS-DOS позволяет полностью использовать возможности процессоров Intel 8088 и Intel 80286, работающих в реальном режиме.

Основными характеристиками данной ОС являются:

– максимальный объем адресуемой физической памяти – 640 Кбайт;

– максимальный объем памяти, доступный из прикладных программ 640 Кбайт. Последние версии MS-DOS (начиная с 5.0) могут использовать адресное пространство между 640 Кбайт и 1 Мбайт для размещения своих составных частей и некоторых драйверов, освобождая тем самым память в адресном пространстве 0-640 Кбайт для использования прикладными

программами;

– представление всех ресурсов персонального компьютера для одной, активной в настоящий момент, программы;

– развитая файловая система и процессор командного языка;

– слабая поддержка интерактивных средств взаимодействия с пользователем;

– занимаемый объем на диске, в зависимости от версии, от 1 Мбайта до 6 Мбайт. (минимум, при котором можно работать – 100 Кбайт).

Основные составные части MS-DOS

MS-DOS состоит из следующих компонент:

– блок начальной загрузки (размещается в 1-м секторе 0-дорожки 0-стороны системной дискеты);

– модуль расширения BIOS (IO.SYS для версии 5.0 и выше);

– модуль обработки прерываний (MSDOS.SYS для версии 5.0 и выше),

– командный процессор (COMMAND.COM);

– внешние команды (программы) MS-DOS;

– драйверы устройств;

– файл Config.SYS;

– файл Autoexec.bat.

Ядро MS-DOS включает блок начальной загрузки и файлы IO.SYS, MSDOS.SYS.

Блок начальной загрузки размещается в 1-м секторе 0-дорожки 0-стороны системной дискеты и/или в 1-м секторе HDD-диска, в разделе, отведенном под DOS. Выполняет следующие функции: просматривает корневой каталог системного диска и проверяет, являются ли **первые два файла** в каталоге – файлами **IO.SYS** и **MSDOS.SYS**. Если *ДА* – загружает их в ОЗУ и передает управление MS-DOS, если *НЕТ* – выдает сообщение на экране и ожидает нажатия какой-либо клавиши пользователем:

Non-System disk or disk error

Replace and press any key when ready

Не системный диск или ошибка диска
Замените и нажмите какую-либо клавишу, когда будет готово

Именно поэтому , при создании *системной дискеты* необходимо переносить на неё файлы *IO.SYS* и *MSDOS.SYS* с помощью специальной программы **SYS.COM**.

Модуль расширения BIOS IO.SYS

Это *резидентный модуль* (всегда находится в ОЗУ после загрузки, пока включен ПК). Взаимодействует с *BIOS*. Расширяет возможности BIOS или изменяет ее свойства (там, где необходимо) с помощью дополнительных драйверов.

Модуль обработки прерываний MSDOS.SYS

Это *резидентный модуль*. Обеспечивает интерфейс высокого уровня для прикладных программ, содержит программные средства для управления файлами, устройствами ввода-вывода, обработки исключительных ситуаций (ошибок) и др. Прикладная программа вызывает функции этого модуля через механизм программных прерываний, передавая (принимая) информацию к (от) MS-DOS через регистры CPU или (и) области памяти ОЗУ. MSDOS.SYS транслирует (переводит) запрос прикладной программы в один или несколько вызовов IO.SYS + BIOS.

Командный процессор COMMAND.COM

Отдельный модуль MS-DOS. Этот модуль может быть заменен на другой, более удобный. Предназначен для приема команд с клавиатуры или из *.bat - файлов и их выполнения; выполнения команд файла Autoexec.bat при загрузке MS-DOS; загрузки в ОЗУ и запуск на выполнение прикладных программ в среде MS-DOS.

Командный процессор состоит из 3-х частей :

– резидентной (размещается в ОЗУ сразу после MSDOS.SYS, включает процедуры обслуживания некоторых прерываний, процедуры обработки стандартных ошибок MS-DOS, процедуру загрузки транзитной части командного процессора);
– инициализирующей (в ОЗУ следует сразу за резидентной частью; во время загрузки ОС ей передается управление; она выполняет файл Autoexec.bat и некоторые другие действия; эта часть командного процессора стирается из ОЗУ первой же загруженной программой);
– транзитной (загружается в старшие адреса ОЗУ; обрабатывает все внутренние команды, команды с клавиатуры и из *.bat-файлов; выдает системную подсказку MS-DOS; загружает в ОЗУ программы и передает им управление).

Драйверы устройств

Специальные резидентные программы, которые управляют внешними устройствами. Драйверы загружаются в ОЗУ в том порядке, как они указаны в файле CONFIG.SYS.

Файл CONFIG.SYS

Специальный текстовый файл, где содержится информация о подгружаемых дополнительных драйверах и некоторая другая информация, касающаяся непосредственно MS-DOS и выполняемых в ее среде прикладных программ. MS-DOS выполняет этот файл автоматически, сразу после загрузки COMMAND.COM.

Файл AUTOEXEC.BAT

Специальный текстовый файл, в котором содержится дополнительная настроечная информация. MS-DOS выполняет этот файл автоматически, сразу после выполнения файла CONFIG.SYS

ВВЕДЕНИЕ В ОПЕРАЦИОННУЮ СИСТЕМУ WINDOWS

Появление Windows 95 ознаменовало переход из эпохи операционной системы MS-DOS к новой эре в мире персональных компьютеров.

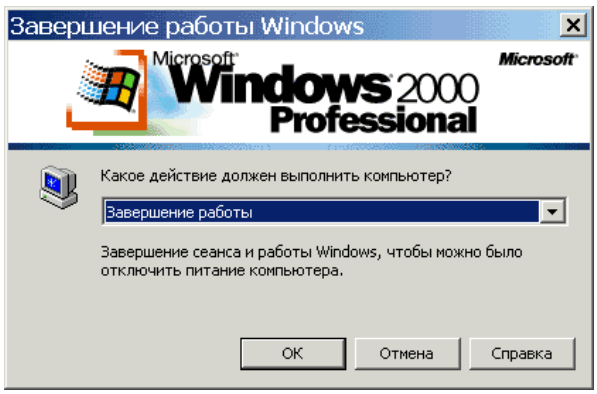
Следующая версия — Windows 98 предоставила пользователям ещё больше средств и возможностей для повышения производительности работы и дружественный интерфейс для работы в Internet.

В Windows 2000 Professional соединились возможности промышленной операционной системы Windows NT 4 с простотой интерфейса и мощью Windows 98. Поэтому в ходе дальнейшего изложения материала будут рассматриваться работа в двух операционных системах — Windows 98/2000.

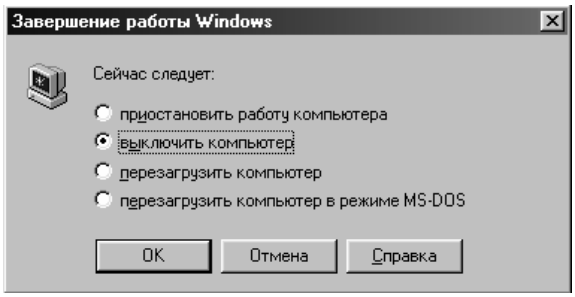
Запуск Windows 2000 Professional

Если вы установили Windows 2000, модернизировав предыдущую версию этой операционной системы, процедура регистрации новых учетных записей будет несколько отличаться от обычной. При модернизации программа установки переносит ваш пользовательский профиль из старой версии и с его помощью создает новую учетную запись в Windows 2000 Professional. В этом случае вам достаточно зарегистрироваться, используя свое старое имя пользователя и пароль.

Завершив работу с Windows, необходимо аккуратно выйти из системы. Только в этом случае вы можете быть уверены в правильном сохранении работы и отсутствии неполадок с системой в будущем. Если же компьютером используются несколько пользователей, нужно завершить сеанс своей работы, чтобы остальные имели возможность зарегистрироваться. Кроме того, завершение сеанса защитит ваши документы и параметры от других пользователей.



а)



б)

Рис. 15. Окно ***Завершение работы:*** *а)* ***- Windows 2000****, б)* ***- Windows 98***

Управление Windows 98/2000

В Windows 98/2000 большую часть команд можно выполнять с помощью мыши. С мышью связан активный элемент управления — ***указатель мыши***. При перемещении мыши по плоской поверхности *указатель* перемещается по *Рабочему столу*, и его можно позиционировать на значках объектов или на пассивных элементах управления приложений.

Основными приемами управления с помощью мыши являются:

- ***щелчок*** (быстрое нажатие и отпускание левой кнопки мыши);
- ***двойной щелчок*** — два щелчка, выполненные с малым интервалом времени между ними;
- ***щелчок правой кнопкой*** (то же, что и *щелчок*, но с использованием правой кнопки);
- ***перетаскивание (drag-and-drop)*** — выполняется путем перемещения мыши при нажатой левой кнопке (обычно сопровождается перемещением экранного объекта, на котором установлен указатель);
- ***протягивание мыши (drag)*** — выполняется, как и *перетаскивание,* но при этом происходит не перемещение экранного объекта, а изменение его формы;

• ***специальное перетаскивание*** — выполняется как и *перетаскивание,* но при нажатой правой кнопке мыши, а не левой;

• ***зависание*** — наведение указателя мыши на значок объекта или на элемент управления и задержка его на некоторое время (при этом обычно на экране появляется *всплывающая подсказка,* кратко характеризующая свойства объекта).

Рабочий стол

То, что появляется на экране по окончании загрузки операционной системы Windows 98/2000, есть не что иное, как своего рода письменный стол. Поэтому такой виртуальный письменный стол называется ***Рабочим столом*** (от английского desktop).

В Windows 98/2000 на рабочем столе располагаются *пиктограммы (значки)* и *ярлыки*. С их помощью вы получаете доступ к соответствующим приложениям или документам. Рабочий стол постоянно находится перед глазами пользователя (если, конечно, он не закрыт окном какого-нибудь приложения), поэтому расположенные на нем пиктограммы и ярлыки всегда доступны.

После инсталляции Windows на рабочем столе отображается несколько значков (например, *Мои компьютер, Сетевое окружение и Корзина*). С помощью первых двух можно получить доступ к локальным, а также выделенным для совместного использования в сети запоминающим устройствам и принтерам. Перетаскивая файлы и папки на значок *Корзина*, вы сможете быстро удалять их. Кроме того, *Корзина* позволяет восстанавливать удаленные документы.

Любая расположенная на рабочем столе пиктограмма (или ярлык) может быть удалена с него. Исключение из этого правила составляют лишь пиктограммы, созданные операционной системой, такие, как *Мой компьютер, Сетевое окружение, Корзина.*

Стартовое меню

С появлением операционной системы Windows 95 пользователи получили возможность тратить гораздо меньше времени на запуск программ по сравнению с DOS и многими другими операционными системами.

В структуру ***Стартового меню*** (рис. 16) входят два раздела — ***обязательный*** *и* ***произвольный***.

Произвольный раздел расположен выше разделительной черты. Пункты этого раздела пользователь может создавать по собственному желанию.

Иногда эти пункты образуются автоматически при установке некоторых приложений.

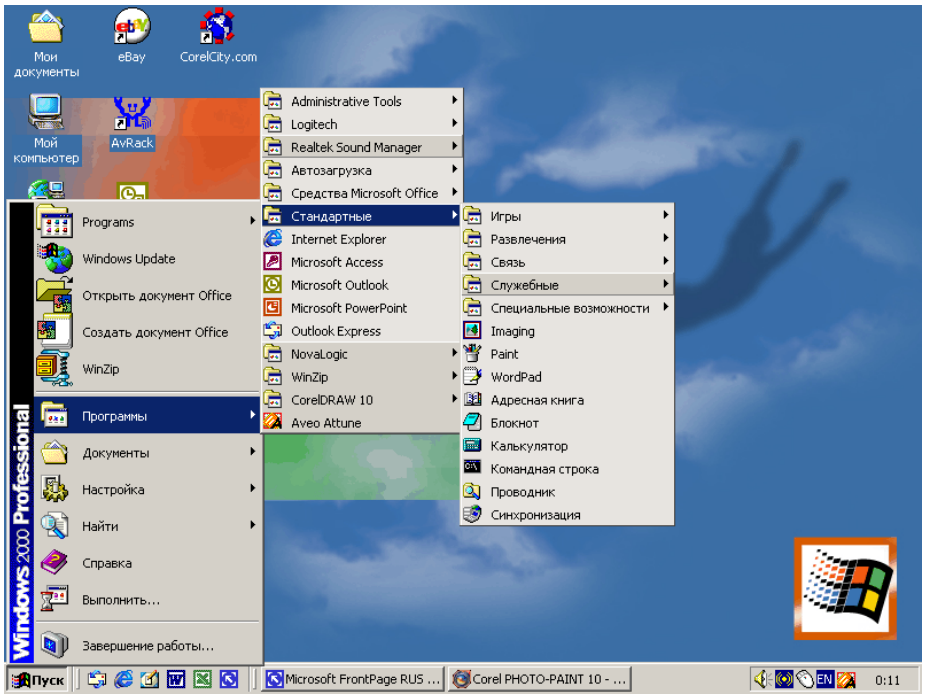


Рис. 16. Стартовое меню Windows 2000

Стартовое меню предназначено, прежде всего, для запуска программ. В нем находятся ***меню*** и ***команды***. Команды служат для запуска различных программ, а меню являются средством упорядочения стартового меню.

В меню могут храниться как команды, так и другие меню — ***подменю***. Благодаря этому, вы можете сконфигурировать стартовое меню таким образом, чтобы определенные категории приложений и средств запускались из различных подменю. Ниже кратко описано назначение команд и меню, расположенных на первом уровне стартового меню.

Меню Программы стартового меню содержит команды, позволяющие запускать как стандартные приложения Windows, так и другие приложения. В Windows 98/2000 вы можете изменять порядок, в котором расположены команды и подменю меню программы.

Меню Избранное содержит созданные вами ссылки на различную информацию сети Интернет.

Меню Документы стартового меню содержит ссылки на 15 последних вызывавшихся пользователем документов. Таким образом, Windows 98/2000 предоставляет вам возможность быстро получить доступ к информации, с которой вы работаете наиболее часто, и помогает избежать отнимающего много времени поиска документов в папках.

Меню Настройка стартового меню содержит, помимо прочих, команды *Панель управления* и *Принтеры*, посредством которых вы можете открыть соответствующие окна**.** С помощью первого окна можно конфигурировать аппаратные и программные средства компьютера, а второе позволяет устанавливать, удалять и конфигурировать драйверы локальных или сетевых принтеров.

Меню Найти. Система Windows предоставляет в распоряжение пользователя несколько утилит поиска информации. Меню *Найти* стартового меню содержит команды: *Файлы и папки, Компьютер, В Интернете и Людей,* которые позволяют соответственно осуществлять поиск файлов и папок, поиск компьютера в локальной сети, поиск данных Internet и поиск информации об интересующих вас персонах.

Команда Справка. Выбрав команду *Справка* в стартовом меню, вы можете запустить справочную систему Windows, которая поможет вам решить различные проблемы и лучше разобраться с программами и средствами Windows.

Команда Выполнить стартового меню вызывает одноименное окно, которое, в свою очередь, позволяет пользователю вводить команды в режиме командной строки.

Команда Завершение работы позволяет пользователю корректно завершить сеанс работы с Windows 98/2000, перегрузить систему или запустить ее в режиме эмуляции MS-DOS

В Windows 2000 из Office 2000 перенесен *адаптируемый интерфейс пользователя*. Он позволяет пометить в меню те приложения, которые используются чаще всего и перемещает их в начало меню *Пуск* (рис. 17).

Рис. 17. Персонифицированное меню Windows 2000

Панель задач

Назначение панели задач — сделать операцию переключения между многими приложениями такой же простой, как переключение каналов телевизора. Каждому открытому в Windows окну на панели задач соответствует определенная кнопка. Можно открыть нужное окно, щелкнув мышью на соответствующей кнопке *Панели задач*.

В Windows 98/2000, в дополнение к кнопкам приложений, на панели задач могут находиться одна или несколько панелей инструментов

АППАРАТНЫЕ СРЕДСТВА ЛВС

Основными аппаратными компонентами ЛВС являются:

1. Рабочие станции (РС) – это, как правило, персональные ЭВМ, которые являются рабочими местами пользователей сети.

Требования, предъявляемые к составу РС, определяются характеристиками решаемых в сети задач, принципами организации вычислительного процесса, используемой ОС и некоторыми другими факторами. Иногда в РС, непосредственно подключенной к сетевому кабелю, могут отсутствовать накопители на магнитных дисках. Такие РС называют ***бездисковыми рабочими станциями.*** Однако в этом случае для загрузки в РС операционной системы с файл-сервера нужно иметь в сетевом адаптере этой станции микросхему дистанционной загрузки. Последняя поставляется отдельно, намного дешевле накопителей и используется как расширение базовой системы ввода-вывода BIOS. В микросхеме записана программа загрузки ОС в оперативную память РС. Основным *преимуществом* бездисковых РС является низкая стоимость, а также высокая защищенность от несанкционированного проникновения в систему пользователей и компьютерных вирусов. *Недостаток* бездисковой РС заключается в невозможности работать в автономном режиме (без подключения к серверу), а также иметь свои собственные архивы данных и программ.

2. Серверы в ЛВС выполняют функции распределения сетевых ресурсов. Обычно его функции возлагают на достаточно мощный ПК, мини-ЭВМ, большую ЭВМ или специальную ЭВМ-сервер. В одной сети может быть один или несколько серверов. Каждый из серверов может быть отдельным или совмещенным с РС. В последнем случае не все, а только часть ресурсов сервера оказывается общедоступной.

При наличии в ЛВС нескольких серверов каждый из них управляет работой подключенных к нему РС. Совокупность компьютеров сервера и

относящихся к нему РС часто называют ***доменом.*** Иногда в одном домене находится несколько серверов. Обычно один из них является главным, а другие – выполняют роль резерва (на случай отказа главного сервера) или логического расширения основного сервера.

Важнейшими параметрами, которые должны учитываться при выборе компьютера-сервера, являются тип процессора, объем оперативной памяти, тип и объем жесткого диска и тип дискового контроллера. Значения указанных характеристик, так же как и в случае РС, существенно зависят от решаемых задач, организации вычислений в сети, загрузки сети, используемой ОС и других факторов.

Оперативная память в сервере используется не только для собственно выполнения программ, а и для размещения в ней буферов, дискового ввода вывода. Определив оптимально количество и размер буферов, можно существенно ускорить выполнение операций ввода-вывода.

Объем выбираемого накопителя должен быть достаточным для размещения на нем необходимого программного обеспечения (особенно при бездисковых РС), а также совместно .используемых файлов и баз данных.

3. Линии передачи данных соединяют РС и серверы в районе размещения сети друг с другом. В качестве линий передачи данных чаще всего выступают ***кабели***. Наибольшее распространение получили кабели на витой паре (рис. 18,а) и коаксиальный кабель (рис. 18,б). Более перспективным и прогрессивным является оптоволоконный кабель. В последнее время стали появляться беспроводные сети, средой передачи данных в которых является радиоканал. В подобных сетях компьютеры устанавливаются на небольших расстояниях друг от друга: в пределах одного или нескольких соседних помещений.

а) б)

Рис. 18. Сетевые кабели: а – кабель на основе скрученных пар (витая пара); б – коаксиальный кабель

4. Сетевые адаптеры (интерфейсные платы) используются для подключения компьютеров к кабелю (рис. 19). Функцией сетевого адаптера является передача и прием сетевых сигналов из кабеля. Адаптер воспринимает команды и данные от сетевой операционной системы (ОС), преобразует эту информацию в один из стандартных форматов и передает ее в сеть через подключенный к адаптеру кабель.

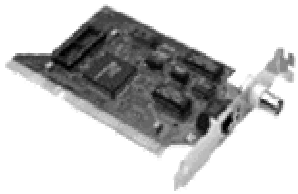


Рис. 19. Сетевой адаптер

Используемые сетевые адаптеры имеют три основные характеристики: ***тип шины*** компьютера, к которому они подключаются (ISA, EISA, Micro Channel и пр.), ***разрядность*** (8, 16, 32, 64) и ***топология*** образуемой сети (Ethernet, Arcnet, Token-Ring).

К ***дополнительному оборудованию*** ЛВС относят источники бесперебойного питания, модемы, трансиверы, репитеры, а также различные разъемы (коннекторы, терминаторы).

Источники бесперебойного питания (ИБП) служат для повышения устойчивости работы сети и обеспечения сохранности данных на сервере. При сбоях по питанию ИБП, подключаемый к серверу через специальный адаптер, выдает сигнал серверу, обеспечивая в течение некоторого времени стабильное напряжение. По этому сигналу сервер выполняет процедуру завершения своей работы, которая исключает потерю данных. Основным критерием выбора ИБП является мощность, которая должна быть не меньше мощности, потребляемой подключаемым к ИБП сервером.

Трансивер – это устройство подключения РС к толстому коаксиальному кабелю. ***Репитер*** предназначен для соединения сегментов сетей. ***Коннекторы*** (соединители) необходимы для соединения сетевых адаптеров компьютеров с тонким кабелем, а также для соединения кабелей друг с другом. ***Терминаторы*** служат для подключения к открытым кабелям сети, а также для заземления (так называемые терминаторы с заземлением).

Модем используется в качестве устройства подключения ЛВС или отдельного компьютера к глобальной сети через телефонную связь.

Топология ЛВС

Топология – это конфигурация соединения элементов в сеть. Топология во многом определяет такие важнейшие характеристики сети, как ее надежность, производительность, стоимость, защищенность и т.д.

Одним из подходов к классификации топологий ЛВС является выделение двух основных классов топологий: ***широковещательных*** **и** ***последовательных.***

В *широковещательных* конфигурациях каждый персональный компьютер передает сигналы, которые могут быть восприняты остальными компьютерами. К таким конфигурациям относятся топологии «общая шина», «дерево», «звезда с пассивным центром». Сеть типа «звезда с пассивным центром» можно рассматривать как разновидность «дерева», имеющего корень с ответвлением к каждому подключенному устройству.

В *последовательных* конфигурациях каждый физический подуровень передает информацию только одному персональному компьютеру. Примерами последовательных конфигураций являются: произвольная (произвольное соединение компьютеров), иерархическая, «кольцо», «цепочка», «звезда с интеллектуальным центром», «снежинка» и др.

Коротко рассмотрим три наиболее широко распространенные (базовые) топологии ЛВС: «звезда», «общая шина» и «кольцо».

В случае ***топологии «звезда»*** каждый компьютер через специальный сетевой адаптер подключается отдельным кабелем к центральному узлу (рис. 19). Центральным узлом служит пассивный соединитель или активный повторитель.

Рис. 19. Топология «звезда»

Недостатком такой топологии является низкая надежность, так как выход из строя центрального узла приводит к остановке всей сети, а также обычно большая протяженность кабелей (это зависит от реального размещения компьютеров). Иногда для повышения надежности в центральном узле ставят специальное реле, позволяющее отключать вышедшие из строя кабельные лучи.

Топология «общая шина» предполагает использование одного кабеля, к которому подключаются все компьютеры. Информация по нему передается компьютерами поочередно (рис. 20).

Рис. 20. Топология «общая шина»

Достоинством такой топологии является, как правило, меньшая протяженность кабеля, а также более высокая надежность чем у «звезды», так как выход из строя отдельной станции не нарушает работоспособности сети в целом. Недостатки состоят в том, что обрыв основного кабеля приводит к неработоспособности всей сети, а также слабая защищенность информации в системе на физическом уровне, так как сообщения, посылаемые одним компьютером другому, в принципе, могут быть приняты и на любом другом компьютере.

При ***кольцевой топологии*** данные передаются от одного компьютера другому по эстафете (рис. 21). Если некоторый компьютер получает данные, предназначенные не ему, он передает их дальше по кольцу. Адресат предназначенные ему данные никуда не передает.

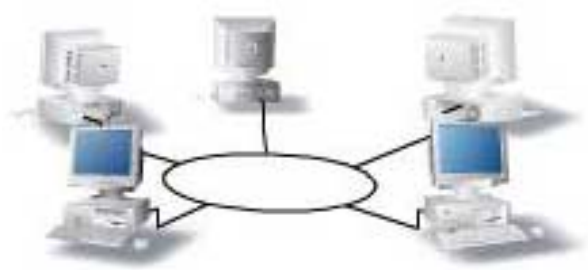


Рис. 21. Кольцевая топология

Достоинством кольцевой топологии является более высокая надежность системы при разрывах кабелей, чем в случае топологии с общей шиной, так как к каждому компьютеру есть два пути доступа. К недостаткам топологии следует отнести большую протяженность кабеля, невысокое быстродействие по сравнению со «звездой» (но соизмеримое с «общей шиной»), а также слабую защищенность информации, как и при топологии с общей шиной.

Топология реальной ЛВС может в точности повторять одну из приведенных выше или включать их комбинацию. Структура сети в общем случае определяется следующими факторами: количеством объединяемых компьютеров, требованиями по надежности и оперативности передачи информации, экономическими соображениями и т. д.

Принципы управления

Существует два основных принципа управления в локальных сетях: централизация и децентрализация.

В сетях с ***централизованным управлением*** функции управления обменом данными возложены на файл-серверы. Файлы, хранящиеся на сервере, доступны рабочим станциям сети. Одна РС к файлам другой РС доступа не имеет. Правда, обмен файлами между РС может происходить и в обход основных путей с помощью специальных программ.

Существует множество сетевых ОС, реализующих централизованное управление. Среди них Microsoft Windows NT/2000 Server, Novell NetWare (версии 3.X и 4.X), Microsoft Lan Manager, OS/2 Warp Server Advanced, VINES 6.0 и др.

Преимуществом централизованных сетей является высокая защищенность сетевых ресурсов от несанкционированного доступа, удобство администрирования сети, возможность создания сетей с большим числом узлов. Основной недостаток состоит в уязвимости системы при нарушении работоспособности файл-сервера (это преодолевается при наличии нескольких серверов или в результате принятия некоторых других мер), а также в предъявлении довольно высоких требований к ресурсам серверов.

В централизованной схеме управления все вычислительные ресурсы, данные и программы их обработки были сконцентрированы в одной ЭВМ. Пользователи имели доступ к ресурсам машины с помощью терминалов (дисплеев). Терминалы подключались к ЭВМ через интерфейсные соединения или удаленные телефонные линии связи (так называемые *удаленные терминалы)*. Основной функцией терминала было отображение информации, представляемой пользователю. К достоинствам этой схемы можно отнести удобство администрирования, модификации программного обеспечения и защиты информации. Недостатком схемы является ее низкая надежность (выход из строя ЭВМ влечет за собой разрушение вычислительного процесса), сложность масштабирования (наращивания мощности) модификации аппаратного и программного обеспечения, как правило, резкое снижение оперативности при увеличении числа пользователей системы и др.

Децентрализованные (одноранговые) сети не содержат в своем составе выделенных серверов. Функции управления сетью в них поочередно передаются от одной РС к другой. Ресурсы одной РС (диски, принтеры и другие устройства) оказываются доступными другим РС.

Наиболее распространенными программными продуктами, позволяющими строить одноранговые сети, являются следующие программы и пакеты: Novell NetWare Lite, Windows for Workgroups, Artisoft LANtastic, LANsmart, Invisible Software NET-30 и др. Все они могут работать под управлением DOS. Для одноранговой сети могут быть использованы также ОС Windows 95/98 и Windows NT/2000 Prof.

Развертывание одноранговой сети для небольшого числа РС часто позволяет построить более эффективную и живучую распределенную вычислительную среду. Сетевое программное обеспечение в них является более простым по сравнению с централизованными сетями. Здесь не требуется установка файл-сервера (как компьютера, так и соответствующих

программ), что существенно удешевляет систему. Однако такие сети слабее с точки зрения защиты информации и администрирования.

Технология «клиент-сервер»

Технология «клиент-сервер» пришла на смену централизованной схеме управления вычислительным процессом на базе средней или большой ЭВМ *(мэйнфрейма)*.

В архитектуре *клиент-сервера* место терминала заняла ПЭВМ (клиентская), а мэйнфрейма – один или несколько мощных компьютеров, специально выделенных для решения общих задач обработки информации (компьютеры-серверы). Достоинством этой модели является высокая живучесть и надежность вычислительной системы, легкость масштабирования, возможность одновременной работы пользователя с несколькими приложениями, высокая оперативность обработки информации, обеспечение пользователя высококачественным интерфейсом и т. д.

Заметим, что эта весьма перспективная и далеко не исчерпавшая себя технология получила свое дальнейшее развитие. Совсем недавно стали говорить о технологии ***intranet,*** которая появилась в результате перенесения идей сети Internet в среду корпоративных систем. В отличие от технологии «клиент-сервер» эта технология ориентирована не на данные, а на информацию в ее окончательно готовом к потреблению виде. Технология Intranet объединяет в себе преимущества двух предыдущих схем. Вычислительные системы, построенные на ее основе, имеют в своем составе центральные серверы информации и распределенные компоненты представления информации конечному пользователю (программы-навигаторы, или броузеры). Детальное рассмотрение этой технологии выходит за рамки настоящего пособия.

При взаимодействии любых двух объектов в сети всегда можно выделить сторону, предоставляющую некоторый ресурс (сервис, услугу), и сторону, потребляющую этот ресурс. Потребителя ресурса традиционно называют ***клиентом,*** а поставщика – ***сервером.***

В качестве ресурса можно рассматривать аппаратный компонент (диск, принтер, модем, сканер и т. д.), программу, файл, сообщение, информацию или даже ЭВМ в целом. Отсюда происхождение множества терминов: файл-сервер или диск-сервер, принт-сервер или сервер печати, сервер сообщений, SQL-сервер (программа обработки запросов к базе данных, сформулированных на языке SQL), компьютер-сервер и т. д. Очевидно, все эти серверы имеют соответствующих клиентов.

С точки зрения программного обеспечения, технология «клиент-сервер» подразумевает наличие программ-клиентов и программ-серверов.

Клиентскими программами обычно являются такие программы, как текстовые и табличные процессоры. В роли серверных программ чаще всего выступают системы управления базами данных. Примером типичной пары программ вида «клиент-сервер» можно считать программу текстового процессора, обрабатывающую документ, в котором содержится таблица с информацией из базы данных.

Некоторая программа, выполняемая в сети, по отношению к одним программам может выступать в роли клиента и в то же время являться сервером для других программ. Более того, за некоторый интервал времени роли клиента и сервера между одними и теми же программами могут меняться.

Разновидностью более сложных клиент-серверных моделей является ***трехзвенная модель*** «сервера приложений» **–** ***AS-модель*** (Application Server). Эта модель описывает процесс функционирования сетей, использующих базы данных. Согласно as-модели, каждая их трех основных функций (управление данными, прикладная обработка и представление информации конечному пользователю) реализуется на отдельном компьютере.

Программное обеспечение технологии «клиент-сервер»

Для успешного применения технологии «клиент-сервер» должно использоваться соответствующее программное обеспечение, включающее клиентскую и серверную части. В частности, широко используемый пакет Microsoft Office представляет собой комплекс программ ***для клиентского компьютера.*** В его состав входят: текстовый процессор Word, табличный процессор Excel, система подготовки презентаций PowerPoint, система управления базами данных Access и программа управления информацией Outlook.

В связи с успехом распространения этого пакета корпорация Microsoft решила собрать воедино комплекс программ ***для сервера*** **–** так появился пакет MS BackOffice.

В состав названного пакета входят следующие компоненты:

- Windows NT Server – сетевая операционная система;
- System Management Server – система администрирования сети;
- SQL Server – сервер управления базами данных;
- SNA Server – сервер для соединения с хост-компьютерами;
- Exchange Server – сервер системы электронной почты;
- Internet Information Server – сервер для работы с Internet.

Windows NT/2000 Server способна обеспечить совместное использование файлов, печатающих устройств, предоставить услуги по соединению с рабочими станциями (клиентскими компьютерами) и другой сервис.

Существуют следующие две разновидности Windows NT/2000:

- Windows NT/2000 Workstation предназначена для использования на ***автономном компьютере;***
- Windows NT/2000 Server предназначена для использования в качестве ***сетевой операционной системы*** и может использоваться на рабочей станции для реализации дополнительных возможностей.

Windows NT Server *целесообразно* использовать в случаях, когда предполагается наличие нескольких процессоров (обычно до четырех). Кроме того, Windows NT Server обеспечивает совместное использование ресурсов многими пользователями, возможность соединения с удаленными сетями через сервис удаленного доступа – RAS (Remote Access Service), а также через средства связи с сетями других фирм (Novell, Digital Pathworks и Apple).

System Management Server (SMS) позволяет сетевому администратору централизованно управлять всей сетью. При этом обеспечивается возможность администрирования каждого компьютера, подключенного к сети, включая установленное на нем программное обеспечение. SMS предоставляет следующий сервис:

- управление инвентаризацией программного и аппаратного обеспечения;
- автоматизация установки и распространения программного обеспечения, включая его обновление;
- удаленное устранение неисправностей и предоставление полного контроля администратору за клавиатурой, мышью и экранами всех компьютеров в сети, работающих под управлением MS-DOS или Windows;
- управление сетевыми приложениями.

SQL Server представляет собой систему управления реляционными базами данных, использующую принципы технологии «клиент-сервер». MS SQL Server поддерживает систему обработки транзакций, систему сохранения ссылочной целостности, механизм распределенных транзакций, тиражирование данных.

SNA Server обеспечивает возможность связи с IBM AS/400 и мэйнфреймами IBM (ЕС ЭВМ). Этот продукт позволяет нескольким настольным ПЭВМ, работающим под управлением MS-DOS, Windows, Windows NT, Macintosh, Unix или OS/2, «видеть» хост-компьютеры.

Exchange Server обеспечивает средства передачи и приема сообщений в информационной сети организации. Этот сервис включает электронную почту (E-mail) и обмен информационными сообщениями для рабочих групп. Microsoft Exchange Server построен на принципах технологии «клиент-сервер» и масштабируется в соответствии с возрастанием вычислительных возможностей сети.

Internet Information Server обеспечивает возможность создания Web-, FTP- и Gopher-серверов для сети Internet, поддерживает управление ими с помощью встроенной программы Internet Service Manager.

Работа пользователя в сети

Рассмотрим работу пользователя в сети средствами ОС Windows 98/2000 Prof, которая может использоваться в качестве:

- ОС ***одноранговой сети*** *с* узлами, функционирующими под управлением как Windows 98, так и Windows 2000 Professional.
- ***клиентской ОС*** в сетях с выделенным сервером, управляемых ОС Windows NT/2000 Server, а также NetWare.

Использование Windows 98/2000 Prof в качестве ОС *одноранговой сети* позволяет:

- осуществлять обмен данными между рабочими станциями сети;
- совместно использовать ресурсы рабочих станций (диски, папки, принтеры, факсы). Для удобства пользования сетевыми папками на рабочих станциях можно назначать им имена дисков. После этого они будут отображаться в виде сетевых дисков в папке *Мой компьютер;*
- обеспечивать парольную защиту ресурсов рабочих станций, выделенных в общее пользование;
- создавать *рабочие группы* пользователей.

Рабочей группой называют совокупность пользователей, объединенных одной общей задачей (например, разработкой проекта) и имеющих общее имя. В рамках каждой рабочей группы действует электронная почта Microsoft Mail, позволяющая обмениваться членам группы почтой. В одной сети может быть организовано несколько рабочих групп, имеющих доступ ко всем сетевым ресурсам.

Клиент сети – это программное обеспечение, позволяющее использовать общие ресурсы сети (папки, принтеры и т.д.). Клиент для сетей NetWare позволяет подключаться к серверам Novell NetWare. Клиент для сетей Microsoft позволяет использовать общие ресурсы компьютеров, работающих под управлением Microsoft Windows 98, Windows NT/2000 и LAN Manager.

Служба дает возможность делать файлы, принтеры и прочие ресурсы доступными для других компьютеров, а также автоматизирует архивацию файлов на сервер сети. В качестве примера можно привести службу доступа к файлам и принтерам сетей Microsoft.

Протокол – это набор конкретных правил обмена информацией между устройствами передачи данных. Существует множество различных протоколов. Чтобы два компьютера могли общаться между собой, они должны использовать одинаковый сетевой протокол.

АЛГОРИТМЫ И СПОСОБЫ ИХ ОПИСАНИЯ

Понятие алгоритма

Для составления программы, предназначенной для решения на ЭВМ какой-либо задачи, требуется составление алгоритма ее решения.

Алгоритм — это точное предписание, которое определяет процесс, ведущий от исходных данных к требуемому конечному результату. Алгоритмами, например, являются правила сложения, умножения, решения алгебраических уравнений, умножения матриц и т.п. Слово алгоритм происходит от *algoritmi*, являющегося латинской транслитерацией арабского имени хорезмийского математика IX века **аль-Хорезми**. Благодаря латинскому переводу трактата **аль-Хорезми** европейцы в XII веке познакомились с позиционной системой счисления, и в средневековой Европе алгоритмом называлась десятичная позиционная система счисления и правила счета в ней.

Применительно к ЭВМ алгоритм определяет вычислительный процесс, начинающийся с обработки некоторой совокупности возможных исходных данных и направленный на получение определенных этими исходными данными результатов. Термин ***вычислительный процесс*** распространяется и на обработку других видов информации, например, символьной, графической или звуковой.

Если вычислительный процесс заканчивается получением результатов, то говорят, что соответствующий алгоритм применим к рассматриваемой совокупности исходных данных. В противном случае говорят, что алгоритм неприменим к совокупности исходных данных. Любой применимый алгоритм обладает следующими ***основными свойствами****:*

- результативностью;
- определенностью;
- массовостью.

Результативность означает возможность получения результата после выполнения конечного количества операций.

Определенность состоит в совпадении получаемых результатов независимо от пользователя и применяемых технических средств.

Массовость заключается в возможности применения алгоритма к целому классу однотипных задач, различающихся конкретными значениями исходных данных.

Для задания алгоритма необходимо описать следующие его элементы:

- набор объектов, составляющих совокупность возможных исходных данных, промежуточных и конечных результатов;
- правило начала;
- правило непосредственной переработки информации (описание последовательности действий);
- правило окончания;
- правило извлечения результатов.

Алгоритм всегда рассчитан на конкретного исполнителя. В нашем случае таким исполнителем является ЭВМ. Для обеспечения возможности реализации на ЭВМ алгоритм должен быть описан на языке, понятном компьютеру, то есть на языке программирования.

Таким образом, можно дать следующее определение программы.

Программа для ЭВМ представляет собой описание алгоритма и данных на некотором языке программирования, предназначенное для последующего автоматического выполнения.

Способы описания алгоритмов

К основным способам описания алгоритмов можно отнести следующие:

- словесно-формульный;
- структурный или блок-схемный;
- с помощью граф-схем;
- с помощью сетей Петри.

Перед составлением программ чаще всего используются словесно-формульный и блок-схемный способы. Иногда перед составлением программ на низкоуровневых языках программирования типа языка Ассемблера алгоритм программы записывают, пользуясь конструкциями некоторого высокоуровнего языка программирования. Удобно использовать программное описание алгоритмов функционирования сложных программных систем. Так, для описания принципов функционирования ОС использовался Алголоподобный высокоуровневый язык программирования.

При ***словесно-формульном способе*** алгоритм записывается в виде текста с формулами по пунктам, определяющим последовательность действий.

Пусть, например, необходимо найти значение следующего выражения:

$$y = 2a – (x+6).$$

Словесно-формульным способом алгоритм решения этой задачи может

быть записан в следующем виде:

1. Ввести значения *a* и *x.*
2. Сложить x и 6.
3. Умножить *a* на 2.
4. Вычесть из *2a* сумму (x+6).
5. Вывести *y* как результат вычисления выражения.

При ***блок-схемном*** описании алгоритм изображается геометрическими фигурами (блоками), связанными по управлению линиями (направлениями потока) со стрелками. В блоках записывается последовательность действий.

Данный способ по сравнению с другими способами записи алгоритма имеет ряд преимуществ. Он наиболее нагляден: каждая операция вычислительного процесса изображается отдельной геометрической фигурой. Кроме того, графическое изображение алгоритма наглядно показывает разветвления путей решения задачи в зависимости от различных условий, повторение отдельных этапов вычислительного процесса и Другие детали.

Оформление программ должно соответствовать определенным требованиям. В настоящее время действует единая система программной документации (ЕСПД), которая устанавливает правила разработки, оформления программ и программной документации. В ЕСПД определены и правила оформления блок-схем алгоритмов (ГОСТ 10.002-80 ЕСПД, ГОСТ 10.003-80 ЕСПД).

Операции обработки данных и носители информации изображаются на схеме соответствующими ***блоками.*** Большая часть блоков по построению условно вписана в прямоугольник со сторонами *a и b.* Минимальное значение ***a* = 10 мм**, увеличение *a* производится на число, кратное **5 мм**. Размер **b=1,5a**. Для от дельных блоков допускается соотношение между *a и b,* равное 1:2. В пределах одной схемы рекомендуется изображать блоки одинаковых размеров. Все блоки нумеруются. Виды и назначение основных блоков приведены в табл. 1.

Таблица 1. Условные обозначения блоков схем алгоритмов

Наименование	***Обозначение***	**Функции**
Процесс		Выполнение операции или группы операций, в результате которых изменяется значение, форма представления или расположение данных.
Ввод-вывод		Преобразование данных в форму, пригодную для обработки (ввод) или отображения результатов обработки (вывод).

Решение		Выбор направления выполнения алгоритма в зависимости от некоторых переменных условий.
Предопределенный процесс		Использование ранее созданных и отдельно написанных программ (подпрограмм).
Документ		Вывод данных на бумажный носитель.
Магнитный диск		Ввод-вывод данных, носителем которых служит магнитный диск.
Пуск-останов		Начало, конец, прерывание процесса обработки данных.
Соединитель		Указание связи между прерванными линиями, соединяющими блоки.
Межстраничный соединитель		Указание связи между прерванными линиями, соединяющими блоки, расположенные на разных листах.
Комментарий		Связь между элементом схемы и пояснением.

Линии, соединяющие блоки и указывающие последовательность связей между ними, должны проводится параллельно линиям рамки. Стрелка в конце линии может не ставиться, если линия направлена слева направо или сверху вниз. В блок может входить несколько линий, то есть блок может являться преемником любого числа блоков. Из блока (кроме логического) может выходить только одна линия. Логический блок может иметь в качестве продолжения один из двух блоков, и из него выходят две линии. Если на схеме имеет место слияние линий, то место пересечения выделяется точкой. В случае, когда одна линия подходит к другой и слияние их явно выражено, точку можно не ставить.

Схему алгоритма следует выполнять как единое целое, однако в случае необходимости допускается обрывать линии, соединяющие блоки.

Если при обрыве линии продолжение схемы находится на этом же листе, то на одном и другом конце линии изображается специальный символ ***соединитель*** — окружность диаметром **0,5 а**. Внутри парных окружностей указывается один и тот же идентификатор. В качестве идентификатора, как правило, используется порядковый номер блока, к которому направлена соединительная линия.

Если схема занимает более одного листа, то в случае разрыва линии вместо окружности используется ***межстраничный соединитель***. Внутри каждого, соединителя указывается адрес — откуда и куда направлена соединительная линия. Адрес записывается в две строки: в первой указывается номер листа, во второй — порядковый номер блока.

Блок-схема должна содержать все разветвления, циклы и обращения к подпрограммам, содержащиеся в программе.

Структурные схемы алгоритмов

Одним из свойств алгоритма является ***дискретность*** — возможность расчленения процесса вычислений, предписанных алгоритмом, на отдельные этапы, возможность выделения участков программы с определенной структурой. Можно выделить и наглядно представить графически три простейшие структуры:

- последовательность двух или более операций;
- выбор направления;
- повторение.

Любой вычислительный процесс может быть представлен как комбинация этих элементарных алгоритмических структур. Соответственно, вычислительные процессы, выполняемые на ЭВМ по заданной программе, можно разделить на три основных вида:

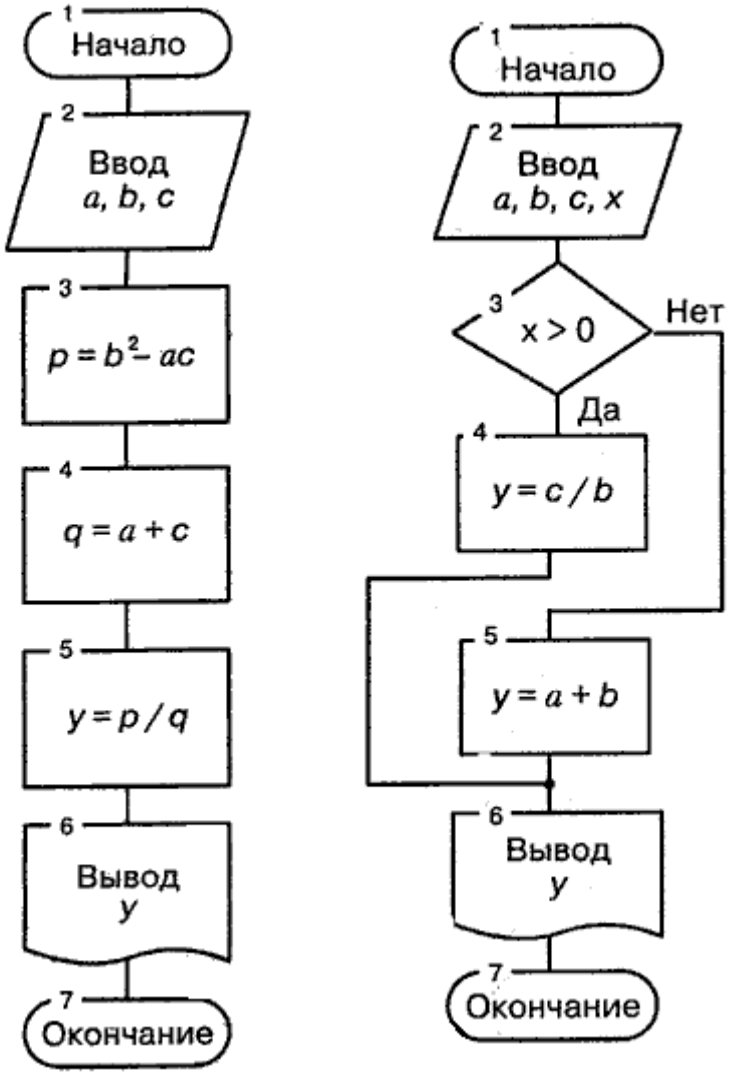
- линейные;
- ветвящиеся;
- циклические.

Линейным принято называть вычислительный процесс, в котором операции выполняются последовательно, в порядке их записи. Каждая операция является самостоятельной, независимой от каких-либо условий. На схеме блоки, отображающие эти операции, располагаются в линейной последовательности.

Линейные вычислительные процессы имеют место, например, при вычислении арифметических выражений, когда имеются конкретные числовые данные и над ними выполняются соответствующие условию задачи действия. На рис. 22, а показан пример линейного алгоритма, определяющего процесс вычисления арифметического выражения $y=(b^2-ac):(a+c)$.

Вычислительный процесс называется ***ветвящимся,*** если для его реализации предусмотрено несколько направлений (ветвей). Каждое отдельное направление процесса обработки данных является отдельной ветвью вычислений. Ветвление в программе — это выбор одной из нескольких последовательностей команд при выполнении программы. Выбор направления зависит от заранее определенного признака, который может относиться к исходным данным, к промежуточным или конечным результатам. Признак характеризует свойство данных и имеет два или более значений.

Ветвящийся процесс, включающий в себя две ветви, называется простым, более двух ветвей — сложным. Сложный ветвящийся процесс можно представить с помощью простых ветвящихся процессов.



а б

Рис. 22. Примеры алгоритмов: а) линейный алгоритм; б) ветвящийся алгоритм

Направление ветвления выбирается логической проверкой, в результате которой возможны два ответа: «да» — условие выполнено и «нет» — условие не выполнено.

Следует иметь в виду, что, хотя на схеме алгоритма должны быть показаны все возможные направления вычислений в зависимости от выполнения определенного условия (или условий), при однократном прохождении программы процесс реализуется только по одной ветви, а остальные исключаются. Любая ветвь, по которой осуществляются вычисления, должна приводить к завершению вычислительного процесса.

На рис. 22,б показан пример алгоритма с разветвлением для вычисления

следующего выражения:

Y = (a+b), если X <0;

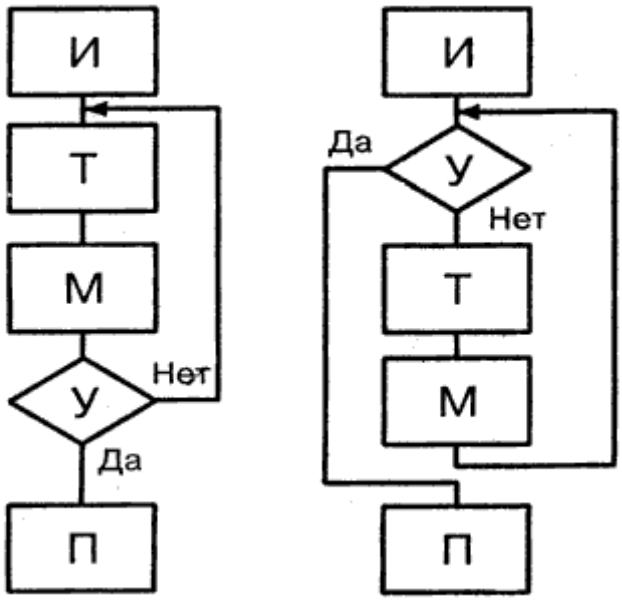
c/b, если X>0.

Циклическими называются программы, содержащие циклы. Цикл — это многократно повторяемый участок программы.

В организации цикла можно выделить следующие ***этапы****:*

- • подготовка (инициализация) цикла (**И**);
- • выполнение вычислений цикла (тело цикла) (**Т**);
- • модификация параметров (**М**);
- • проверка условия окончания цикла (**У**).

Порядок выполнения этих этапов, например, **Т** и **М**, может изменяться. В зависимости от расположения проверки условия окончания цикла различают циклы с нижним и верхним окончаниями (рис. 23). Для цикла с нижним окончанием (рис. 23, а) тело цикла выполняется как минимум один раз, так как сначала производятся вычисления, а затем проверяется условие выхода из цикла. В случае цикла с верхним окончанием (рис. 23, б) тело цикла может не выполниться ни разу в случае, если сразу соблюдается условие выхода.



а б

Рис. 23. Примеры циклических алгоритмов

Цикл называется ***детерминированным,*** если число повторений тела

цикла заранее известно или определено. Цикл называется ***итерационным,*** если число повторений тела цикла заранее неизвестно, а зависит от значений параметров (некоторых переменных), участвующих в вычислениях.

На рис. 24 показан пример циклического алгоритма вычисления суммы десяти чисел.

Рис. 24. Алгоритм нахождения суммы 10-ти чисел

ЭТАПЫ ПОДГОТОВКИ И РЕШЕНИЯ ЗАДАЧ НА ЭВМ

На ЭВМ могут решаться задачи различного характера, например: научно-инженерные; разработки системного программного обеспечения; обучения; управления производственными процессами и т. д. В процессе подготовки и решения на ЭВМ научно -инженерных задач можно выделить следующие ***этапы:***

- постановка задачи;
- математическое описание задачи;
- выбор и обоснование метода решения;
- алгоритмизация вычислительного процесса;
- составление программы;
- отладка программы;
- решение задачи на ЭВМ и анализ результатов.

В задачах другого класса некоторые этапы могут отсутствовать, например, в задачах разработки системного программного обеспечения отсутствует математическое описание. Перечисленные этапы связаны друг с другом. Например, анализ результатов может показать необходимость внесения изменений в программу; алгоритм или даже в постановку задачи. Для уменьшения числа подобных изменений необходимо на каждом этапе по возможности учитывать требования, предъявляемые последующими этапами. В некоторых случаях связь между различными этапами, например, между постановкой задачи и выбором метода решения, между составлением алгоритма и программированием, может быть настолько тесной, что разделение их становится затруднительным.

Постановка задачи. На данном этапе формулируется цель решения задачи и подробно описывается ее содержание. Анализируются характер и сущность всех величин, используемых в задаче, и определяются условия, при которых она решается. Корректность постановки задачи является важным моментом, так как от нее в значительной степени зависят другие этапы.

Математическое описание задачи. Настоящий этап характеризуется математической формализацией задачи, при которой существующие соотношения между величинами, определяющими результат, выражаются посредством математических формул. Так формируется математическая модель явления с определенной точностью, допущениями и ограничениями. При этом в зависимости от специфики решаемой задачи могут быть использованы различные разделы математики и других дисциплин. |

Математическая модель должна удовлетворять по крайней мере двум требованиям: реалистичности и реализуемости. Под ***реалистичностью*** понимается правильное отражение моделью наиболее существенных *черт* исследуемого явления.

Реализуемость достигается *разумной* абстракцией, отвлечением от второстепенных деталей, чтобы свести задачу к проблеме с известным решением. Условием реализуемости является возможность практического выполнения необходимых вычислений за отведенное время при доступных затратах требуемых ресурсов.

Выбор и обоснование метода решения. Модель решения задачи с учетом ее особенностей должна быть доведена до решения при помощи конкретных методов решения. Само по себе математическое описание задачи в большинстве случаев трудно перевести на язык машины. Выбор и использование метода решения задачи позволяет привести решение задачи к конкретным машинным операциям. При обосновании выбора метода необходимо учитывать различные факторы и условия, в том числе точность вычислений, время решения задачи на ЭВМ, требуемый объем памяти и другие.

Одну и ту же задачу можно решить различными методами, при этом в рамках каждого метода можно составить различные алгоритмы.

Алгоритмизация вычислительного процесса. На данном этапе составляется алгоритм решения задачи согласно действиям, задаваемым выбранным методом решения. Процесс обработки данных разбивается на отдельные относительно самостоятельные блоки, и устанавливается последовательность выполнения блоков. Разрабатывается блок-схема алгоритма.

Составление программы. При составлении программы алгоритм решения задачи переводится на конкретный язык программирования. Для программирования обычно используются языки высокого уровня, поэтому составленная программа требует перевода ее на машинный язык ЭВМ. После такого перевода выполняется уже соответствующая машинная программа.

Отладка программы. Отладка заключается в поиске и устранении синтаксических и логических ошибок в программе.

В ходе синтаксического контроля программы транслятором выявляются конструкции и сочетания символов, недопустимые с точки зрения правил их построения или написания, принятых в данном языке. Сообщения об ошибках ЭВМ выдает программисту, при этом вид и форма выдачи подобных сообщений зависят от вида языка и версии используемого транслятора.

После устранения синтаксических ошибок проверяется логика работы программы в процессе ее выполнения с конкретными исходными данными. Для этого используются специальные методы, например, в программе выбираются контрольные точки, для которых вручную рассчитываются промежуточные результаты. Эти результаты сверяются со значениями, получаемыми ЭВМ в данных точках при выполнении отлаживаемой программы. Кроме того, для поиска ошибок могут быть использованы отладчики, выполняющие специальные действия на этапе отладки, например, удаление, замена или вставка отдельных операторов или целых фрагментов программы, вывод или изменение значений заданных переменных.

Решение задачи на ЭВМ и анализ результатов. После отладки программы ее можно использовать для решения прикладной задачи. При этом обычно выполняется многократное решение задачи на ЭВМ для различных наборов исходных данных. Получаемые результаты интерпретируются и анализируются специалистом или пользователем, поставившим задачу.

Разработанная программа длительного использования устанавливается на ЭВМ, как правило, в виде готовой к выполнению машинной программы. К программе прилагается документация, включая инструкцию для пользователя.

Чаще всего при установке программы на диск для ее последующего использования помимо файлов с исполняемым кодом устанавливаются различные вспомогательные программы (утилиты, справочники, настройщики и т. д.), а также необходимые для работы программ разного рода файлы с текстовой, графической, звуковой и другой информацией.

Компиляция и интерпретация программ

ЭВМ непосредственно выполняет программы на ***машинном языке*** программирования данной ЭВМ. При этом программа представляет собой набор отдельных команд компьютера. Эти команды являются достаточно «простыми», например, сложение, умножение, сравнение или пересылка отдельных данных. Каждая команда содержит в себе сведения о том, какая операция должна быть выполнена (код операции), с какими операндами (адреса данных или непосредственно сами данные) выполняются вычисления и куда (адрес) должен быть помещен результат.

Машинные языки были первыми языками программирования. Программирование на них затруднительно ввиду того, что, во-первых, эти языки различны для каждого типа ЭВМ, во-вторых, являются трудоемкими для большинства пользователей по причине необходимости знания особенностей конкретной ЭВМ и большого количества реализуемых ею операций (команд). Данные языки обычно используются для разработки системных программ, при этом чаще всего применяются специальные символические языки — Ассемблеры, близкие к соответствующим машинным языкам.

Человеку свойственно формулировать и решать задачи в выражениях более общего характера, чем команды ЭВМ. Поэтому с развитием программирования появились языки, ориентированные на более высокий уровень абстракции при описании решаемой на ЭВМ задачи. Эти языки получили название языков высокого уровня. Их теоретическую основу составляют алгоритмические языки, например, Паскаль, Си, Бейсик, Фортран, PL/1.

Для перевода программы, написанной на языке высокого уровня, в соответствующую машинную программу используются ***языковые процессоры***. Различают два вида языковых процессоров: интерпретаторы и трансляторы.

Интерпретатор — это программа, которая получает исходную программу и по мере распознавания конструкций входного языка реализует действия, описываемые этими конструкциями.

Транслятор — это программа, которая принимает исходную программу и порождает на своем выходе программу, записываемую на объектном языке программирования (объектную программу). В частном случае объектным может служит машинный язык, и в этом случае полученную на выходе транслятора программу можно сразу же выполнить на ЭВМ. В общем случае объектный язык необязательно должен быть машинным или близким к нему (автокодом). В качестве объектного языка может служить и некоторый промежуточный язык.

Для промежуточного языка может быть использован другой транслятор или интерпретатор — с промежуточного языка на машинный. Транслятор, использующий в качестве входного язык, близкий к машинному (автокод или язык Ассемблера) традиционно называют Ассемблером.

Транслятор с языка высокого уровня называют ***компилятором***.

Стили программирования

Одним из важнейших признаков классификации языков программирования является принадлежность их к одному из стилей, основными из которых являются следующие: процедурный, функциональный, логический и объектно-ориентированный.

Процедурное программирование

Процедурное (императивное) программирование является отражением архитектуры традиционных ЭВМ, которая была предложена фон Нейманом в 40-х годах. Теоретической моделью процедурного программирования служит алгоритмическая система под названием «машина Тьюринга».

Программа на процедурном языке программирования состоит из последовательности операторов (инструкций), задающих процедуру решения задачи. Основным является оператор присваивания, служащий для изменения содержимого областей памяти. Концепция памяти как хранилища значений, содержимое которого может обновляться операторами программы, является фундаментальной в императивном программировании.

Выполнение программы сводится к последовательному выполнению операторов с целью преобразования исходного состояния памяти, то есть значений исходных данных, в заключительное, то есть в результаты. Таким образом, с точки зрения программиста имеются программа и память, причем первая последовательно обновляет содержимое последней.

Процедурные языки характеризуются следующими особенностями:

- необходимостью явного управления памятью, в частности, описанием переменных;
- малой пригодностью для символьных вычислений;
- отсутствием строгой математической основы;
- высокой эффективностью реализации на традиционных ЭВМ.

Одним из важнейших классификационных признаков процедурного языка является его уровень. Уровень языка программирования определяется семантической (смысловой) емкостью его конструкций и степенью его ориентации на программиста. Язык программирования частично ликвидирует разрыв между методами решения различного рода задач человеком и вычислительной машиной. Чем более язык ориентирован на человека, тем выше его уровень. Дадим краткую характеристику реализованным на ПЭВМ языкам программирования в порядке возрастания их уровня.

Двоичный язык является непосредственно машинным языком. В настоящее время такие языки программистами практически не применяются.

Язык Ассемблера — это язык, предназначенный для представления в удобочитаемой символической форме программ, записанных на машинном языке. Он позволяет программисту пользоваться мнемоническими кодами операций, присваивать удобные имена ячейкам и областям памяти, а также задавать наиболее удобные схемы адресации.

Язык Макроассемблера является расширением языка Ассемблера путем включения в него макросредств. С их помощью в программе можно описывать последовательности инструкций с параметрами — макроопределения. После этого программист может использовать снабженные аргументами макрокоманды, которые в процессе ассемблирования программы автоматически замещаются макрорасширениями. Макрорасширение представляет собой макроопределение с подставленными вместо параметров аргументами.

Другими словами, язык Макроассемблера представляет средства определения и использования новых, более мощных команд как последовательности базовых инструкций, что несколько повышает его уровень.

Языки Ассемблера и Макроассемблера применяются системными программистами-профессионалами с целью использования всех возможностей оборудования ЭВМ и получения эффективной по времени выполнения и по требуемому объему памяти программы. На этих языках обычно разрабатываются относительно небольшие программы, входящие в состав системного программного обеспечения: драйверы, утилиты и другие.

Язык программирования С (Си) первоначально был разработан для реализации операционной системы UNIX в начале 70-х годов. В последующем приобрел высокую популярность среди системных и прикладных программистов. В настоящее время этот язык реализован на большинстве ЭВМ.

В С сочетаются достоинства современных высокоуровневых языков в части управляющих конструкций и структур данных с возможностями доступа к аппаратным средствам ЭВМ на уровне, который обычно ассоциируется с языком низкого уровня типа языка Ассемблера. Язык С имеет синтаксис, обеспечивающий краткость программы, а компиляторы способны генерировать эффективный объектный код.

Одна из наиболее существенных особенностей С состоит в нивелировании различий между выражениями и операторами, что приближает его к функциональным языкам. В частности, выражение может обладать побочным эффектом присваивания, а также может использоваться в качестве оператора. Нет также четкой границы между процедурами и функциями, более того, понятие процедуры не вводится вообще.

Синтаксис языка затрудняет программирование и восприятие составленных программ. Отсутствует и строгая типизация данных, что предоставляет дополнительные возможности программисту, но не способствует написанию надежных программ.

Basic(Бэйсик) (Beginners All-purpose Symbolic Instruction Code) — многоцелевой язык символических инструкций для начинающих) представляет собой простой язык программирования, разработанный в 1964 году для использования новичками. Он был разработан как простейший язык для непосредственного общения человека с вычислительной машиной. Поэтому первоначально работа велась в интерактивном режиме с использованием интерпретаторов. В настоящее время для этого языка имеются также и компиляторы.

Согласно концепциям, заложенным в Basic, этот язык в смысле строгости и стройности является антиподом языка Pascal. В частности, в нем широко распространены различные правила умолчания, что считается плохим тоном в большинстве языков программирования подобного типа.

Basic широко распространен на ЭВМ различных типов и очень популярен в среде программистов, особенно начинающих. Существует множество диалектов этого языка, мало совместимых между собой. Basic активно поглощает многие концепции и новинки из других языков. Поэтому он достаточно динамичен, и нельзя однозначно определить его уровень.

Pascal (Паскаль) является одним из наиболее популярных среди прикладных программистов процедурным языком программирования, особенно для ПЭВМ. Разработанный в 1970 году швейцарским специалистом в области вычислительной техники профессором Н. Виртом, язык назван в честь французского математика и по замыслу автора предназначался для обучения программированию. Однако язык получился настолько удачным, что стал одним из основных инструментов прикладных и системных программистов при решении задач вычислительного и информационно-логического характера. В 1979 году был подготовлен проект описания языка — Британский стандарт языка программирования Pascal BS6192, который стал также и международным стандартом ISO 7185.

В языке Pascal реализован ряд концепций, рассматриваемых как основа «дисциплинированного» программирования и заимствованных впоследствии разработчиками многих языков. Одним из существенных признаков языка Pascal является последовательная и достаточно полная реализация концепции структурного программирования. Причем это осуществляется не только путем упорядочивания связей между фрагментами программы по управлению, но и за счет структуризации данных. Кроме того, в языке реализована концепция определения новых типов данных на основе уже имеющихся. Этот язык, в отличие от языка С, является строго типизированным. Pascal характеризуется:

- высоким уровнем;
- широкими возможностями;
- стройностью, простотой и краткостью;
- строгостью, способствующей написанию эффективных и надежных программ;
- высокой эффективностью реализации на ЭВМ.

Pascal реализован на ЭВМ различных типов, но наиболее распространен и развит для ПЭВМ. В настоящее время широко используются такие версии этого языка для ПЭВМ, как Borland Pascal и Turbo Pascal.

ЗНАКОМСТВО СО СРЕДОЙ ТУРБО ПАСКАЛЯ

Система программирования Турбо Паскаль представляет собой единство двух в известной степени самостоятельных начал: компилятора с языка

программирования Паскаль (язык назван в честь выдающегося французского математика и философа Блеза Паскаля (1623-1662)) и некоторой инструментальной программной оболочки, способствующей повышению эффективности создания программ. Для краткости условимся в дальнейшем называть реализуемый компилятором язык программирования Паскаль - *языком Турбо Паскаля*, а разнообразные сервисные услуги, представляемые программной оболочкой, - *средой Турбо Паскаля*.

Как начать работу с турбо паскалем

Система Турбо Паскаль довольна значительна по объему. Она поставляется на нескольких дистрибутивных дискетах и устанавливается на жесткий диск. При развертывании системы на жестком диске обычно создается каталог с именем *TP* (или *PAS, TURBOPAS, PASCAL* и т.п.), в который помещаются все файлы с дистрибутивных дискет. Для вызова Турбо Паскаля необходимо отыскать в древовидной структуре каталогов ПК этот каталог и в нем файл *TURBO.EXE*. Этот файл содержит готовую к работе диалоговую систему программирования Турбо Паскаль. В него входят минимально необходимые части Турбо Паскаля (текстовый редактор, компилятор, компоновщик, загрузчик). Для нормальной работы в диалоговой среде понадобятся также основная библиотека, располагающаяся в файле *TURBO. TPL*, и справочная служба (файл *TURBO.HLP*).

Пусть перечисленные файлы располагаются в каталоге *TP* на диске *D*. Тогда для вызова Турбо Паскаля следует дать команду:

```
D:\TP\TURBO
```

По этой команде операционная система *MS-DOS* поставит на исполнение программу из файла *TURBO.EXE*: загрузит программу в оперативную память и передаст ей управление.

После успешного вызова системы экран ПК приобретает вид, показанный на рис. 25.

Сразу же скажем, что для выхода из Турбо Паскаля следует нажать клавишу *Alt* и, не отпуская ее, - клавишу с латинской буквой *X*, после чего можно отпустить обе клавиши.

Верхняя строка содержит «меню» возможных режимов работы Турбо Паскаля, нижняя - краткую справку о назначении основных функциональных клавиш. Вся остальная часть экрана принадлежит окну редактора, очерченному двойной рамкой и предназначенному для ввода и коррекции текстов программ. В его верхней строке приводятся имя того дискового файла, откуда был прочитан текст программы (новому файлу присваивается имя *NONAME00.PAS*), два специальных поля, используемых при работе с

устройством ввода «мышь» (эти поля выделены квадратными скобками), и цифра 1 - номер окна. В Турбо Паскале можно работать одновременно с несколькими программами (или частями одной крупной программы), каждая из которых может располагаться в отдельном окне редактора. Среда позволяет использовать до 9-ти окон редактора одновременно.

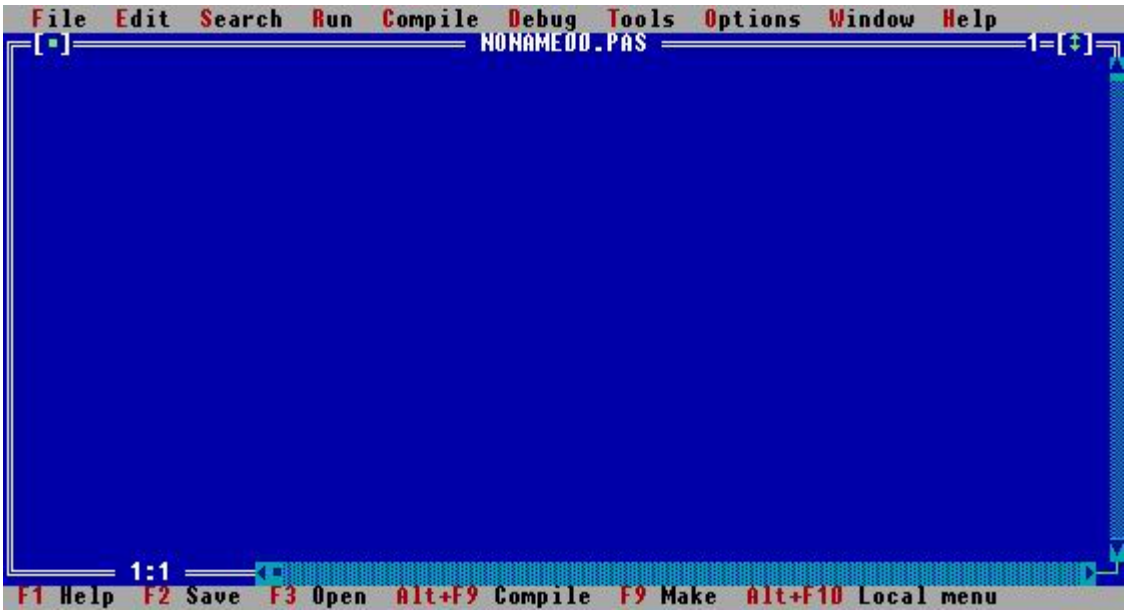


Рис. 25. Вид экрана после вызова Турбо Паскаля

Кроме окна (окон) редактора в Турбо Паскале используются также окна отладочного режима, вывода результатов работы программы, справочной службы, стека, регистров. По желанию они могут вызываться на экран поочередно или присутствовать на нем одновременно.

Для знакомства с языком Турбо Паскаля попробуем составить несложную программу, осуществляющую вывод какого-либо сообщения на экран ПК. Пусть это будет фраза «Я программирую на Турбо Паскале». Вот возможный вариант такой программы:

Пример

```
Program My_First_Program;
const
Text = 'Я программирую на Турбо Паскале';
begin
WriteLn(Text);
end.
```

Прежде всего проанализируем форму представления текста. В программе шесть строк. Строки программы обычно выделяют некоторые смысловые фрагменты текста и могут не связываться с конкретными действиями в программе: расположение текста программы по строкам - дело

вкуса программиста, а не требование синтаксиса языка. Ту же программу можно было бы написать, например, так:

```
Program My_First_Program; const Text =
'Я программирую на Турбо Паскале';begin WriteLn(Text); end.
```

В отличие от некоторых других языков программирования пробел в языке Турбо Паскаль используется как разделитель отдельных конструкций языка, поэтому программа

```
PROGRAMMy_First_Program;constText=
'Я программирую на Турбо Паскале';BEGINWriteLn(Text);end.
```

будет неверной.

В Турбо Паскале игнорируется различие в высоте букв (заглавные или строчные), если только это не связано с текстовыми константами. Начало программы могло бы, например, выглядеть так:

```
program my_first_program;
```

Теперь о смысле отдельных строк. Первая строка

```
Program My_First_Program;
```

начинается словом Program и содержит объявление имени программы. Слово Program зарезервировано в Турбо Паскале, т.е. не может использоваться ни в каких иных целях, кроме как для объявления имени программы. В Турбо Паскале имеется множество зарезервированных слов (см. гл.3). Любое из них нельзя использовать в качестве идентификатора (имени) какого-либо объекта программы - переменной, константы и т.д. Замечу, что редактор среды Турбо Паскаля обычно выделяет зарезервированные слова цветом. В связи с этим в тексте книги эти слова выделены жирным шрифтом. Поскольку имя программы никак в дальнейшем не используется, требование его объявления кажется излишним. В Турбо Паскале можно опускать объявление имени оператором Program без каких-либо последствий для программы.

В рассматриваемом примере имя My_First_Program есть не что иное, как английская фраза «Моя Первая Программа», но только написанная без пробелов - пробел является разделителем и не может использоваться произвольно (вместо пробелов в идентификаторах разрешается использовать символ подчеркивания).

Первая строка заканчивается особым разделителем - точкой с запятой. Этот разделитель в языке Турбо Паскаль отмечает конец оператора или описания. Использование особого разделителя позволяет располагать несколько операторов на одной строке.

Вторая строка `const` содержит единственное зарезервированное слово const, означающее, что далее будут описаны одна или несколько констант (CONSTants - *константы*). Константами в языке считаются такие объекты программы, которые не могут изменять своего значения. В отличие от многих других языков программирования, константа в Турбо Паскале может иметь собственное имя, что соответствует принятой в научных и инженерных расчетах практике именования часто используемых констант. Например, со школы мы помним о существовании константы п—3.14159265. При обработке программы имя константы pi будет заменяться компилятором на ее значение.Описать константу в Турбо Паскале - значит указать ее имя и значение. Такое указание содержится в третьей строке

```
Text = 'Я программирую на Турбо Паскале';
```

в которой константе с именем Text присваивается в качестве значения строка символов «Я программирую на Турбо Паскале».

В Турбо Паскале могут использоваться константы разного типа - целые или вещественные числа, символы, строки символов, массивы и т.д. Признаком того, что Text является константой типа строка символов, служат два апострофа, обрамляющих строку, причем сами апострофы этой строке не принадлежат, а лишь указывают компилятору на то, что все заключенные в них символы следует рассматривать как единое целое - текстовую константу. Если понадобится включить сам апостроф в текстовую константу, достаточно его написать дважды подряд. Например, описание

```
Text = 'Турбо' 'Паскаль';
```

создаст константу со значением `Турбо'Паскаль`

Все три первые строки не связаны с какими-либо конкретными действиями при работе программы. Они сообщают компилятору некоторые сведения о самой программе и использующихся в ней объектах. Эта часть программы называется разделом описаний. Зарезервированное слово begin в четвертой строке сигнализирует компилятору о начале другой части программы - раздела операторов. В нашем примере этот раздел содержит оператор `WriteLn(Text);` который, собственно, и выводит сообщение на экран компьютера.

Завершает всю программу зарезервированное слово end с точкой. Точка оповещает компилятор о конце текста программы. За сочетанием end. можно размещать какой угодно текст - он не будет обрабатываться компилятором.

Перед тем как попробовать откомпилировать и исполнить нашу программу, обсудим ее единственный исполняемый оператор `WriteLn(Text);`

Любопытно, что в Паскале вообще и Турбо Паскале, в частности, нет специальных операторов ввода-вывода. Для обмена информацией с окружающим миром в программах, написанных на языке Турбо Паскаль, используются специальные стандартные процедуры. Таким образом, по своей сути оператор `WriteLn(Text);` является оператором обращения к встроенной процедуре вывода данных (свое название она получила от WRITE LiNe - записать строку).

Процедура `WriteLn` относится к стандартным или встроенным процедурам Турбо Паскаля. Стандартная процедура не нуждается в предварительном описании, она доступна любой программе, в которой содержится обращение к ней. Разница между оператором вывода и обращением к процедуре вывода состоит в том, что имя процедуры вывода, как и любой другой процедуры Турбо Паскаля, не является зарезервированным словом, а следовательно, пользователь может написать свою собственную процедуру с именем `WriteLn`. Впрочем, эта возможность для большинства пользователей остается лишь языковой тонкостью и очень редко используется на практике.

Процедура `WriteLn` - одна из немногих процедур Турбо Паскаля, при обращении к которым допускается использование произвольного числа параметров. Параметры передаются процедуре в виде списка, располагающегося в круглых скобках сразу за именем процедуры. В нашем примере процедуре передается единственный параметр -константа Text.

Анализируя всю программу в целом, мы обнаружим, что четыре использовавшихся в ней слова (Program, const, begin и end) являются зарезервированными. Слово WriteLn, как уже отмечалось, не относится к зарезервированным, но вряд ли может возникнуть необходимость переопределить его, так как в этом случае программа лишится мощного и удобного средства вывода данных. Два слова My_First_Program и Text служат идентификаторами (именами) некоторых объектов программы. Программист может использовать в качестве идентификаторов любые последовательности символов, которые удовлетворяют следующим ограничениям:

- идентификатор может состоять из букв латинского алфавита, цифр, знака подчеркивания; никакие другие символы в идентификаторе недопустимы;
- идентификатор не может начинаться с цифры;
- идентификатор не может совпадать ни с одним из зарезервированных слов;
- длина идентификатора может быть произвольной, но значащими считаются первые 63 символа.

Как и всюду в программе, в идентификаторах игнорируется разница в высоте букв, поэтому, например, идентификаторы Text, text и TEXT с точки зрения компилятора идентичны.

ТИПЫ ДАННЫХ

Структура рассмотренной программы имеет следующий вид:

```
Program MyFirstProgram;
{Раздел описаний}
begin
{Раздел операторов}
end.
```

Слова Program, begin и end выделяют две части программы - раздел описаний и раздел операторов. Такая структура обязательна для любой программы, что является следствием жесткого требования языка: любой нестандартный идентификатор, используемый в исполняемых операторах, должен быть предварительно описан в разделе описаний. (Стандартные идентификаторы связаны с предварительно объявленными объектами и входят в стандартную библиотеку Турбо Паскаля. Таким, например, является идентификатор WriteLn. Стандартные идентификаторы, если они используются в программе, описывать не нужно).

Любые данные, т.е. константы, переменные, значения функций или выражения, в Турбо Паскале характеризуются своими типами. *Тип* определяет множество допустимых значений, которые может иметь тот или иной объект, а также множество допустимых операций, которые применимы к нему. Кроме того, тип определяет также и формат внутреннего представления данных в памяти ПК.

Турбо Паскаль характеризуется разветвленной структурой типов данных (рис.26).

В Турбо Паскале предусмотрен механизм создания новых типов данных, благодаря чему общее количество типов, используемых в программе, может быть сколь угодно большим.

Требование предварительного описания идентификаторов кажется чрезмерно строгим и делающим язык менее свободным. На самом деле в нем проявляется тенденция развития языков программирования в сторону повышения надежности создаваемых программ. Кто программировал на Фортране или Бэйсике (в этих языках не требуется предварительное описание идентификаторов), знает, как порой бывает трудно обнаружить в большой программе ошибочно введенный или пропущенный символ в идентификаторе. Если, например, всюду в программе используется переменная с именем EPSILON, а в одном месте ошибочно написано

EPSLION, то программа может благополучно откомпилироваться и даже давать почти правдоподобный результат для некоторых наборов данных, но в какой-то момент начнет вести себя странно. Обязательное предварительное описание идентификаторов в Турбо Паскале защищает программы от такого рода ошибок и повышает их надежность.

Рис. 26. Структура типов данных

Описать идентификатор - это значит указать тип связанного с ним объекта программы (константы или переменной). Понятие типа - одно из фундаментальных понятий Турбо Паскаля. Тип определяет, во-первых, способ внутреннего для компьютера представления объекта и, во-вторых, действия, которые разрешается над ним выполнять.

Основные типы данных:

- INTEGER - целочисленные данные, во внутреннем представлении занимают 2 байта; диапазон возможных значений - от -32768 до +32767; данные представляются точно;
- REAL - вещественные данные, занимают 6 байт; диапазон возможных значений модуля - от 2.9E-39 до 1.7E+38; точность представления данных - 11...12 значащих цифр;
- CHAR - символ, занимает 1 байт;
- STRING - строка символов, занимает MAX+1 байт, где MAX - максимальное число символов в строке;
- BOOLEAN - логический тип, занимает 1 байт и имеет два значения: FALSE (ложь) и TRUE (истина).

Тип константы определяется способом записи ее значения. Например:

```
const
c1 = 17;
```

```
c2 = 3 .14 ;
c3 = 'A';
c4 = '3.14 ' ;
c5 = False;
```

При анализе этого фрагмента программы компилятор отнесет первую константу к типу INTEGER, вторую - к типу REAL, третью - к CHAR, четвертую - к STRING и последнюю - к BOOLEAN. Признаком, позволяющим отнести константу к REAL или к INTEGER, является наличие или отсутствие десятичной точки в ее значении. Разумеется, константы С2 и С4 относятся к разным типам: С2 - к REAL (в константе есть десятичная точка), а С4 - к STRING (константа обрамлена апострофами). Константу С3 компилятор будет считать относящейся к типу CHAR: одиночный символ в апострофах относится к CHAR, в то время как несколько символов - к STRING.

В отличие от константы переменная именует объект программы, который может изменять свое значение в ходе счета. При описании переменных за идентификатором ставятся двоеточие и имя типа. Несколько однотипных переменных можно объединять в список, разделяя их запятыми. В начале раздела описания переменных должно стоять зарезервированное слово VAR (VARiables - переменные). Например:

```
var
sigma :Real; a,b,c,d :Char;
textl :String[15];
text2 :String;
flag :Boolean;.
```

Как уже говорилось, тип данных определяет длину внутреннего представления соответствующих переменных. В частности, длина внутреннего представления переменных типа STRING (строка символов) зависит от максимального числа символов, которые могут составлять строку. В приведенном выше примере переменная text 1 описана с указанием ее максимальной длины (15 символов), а в описании переменной text2 максимальная длина не указана и компилятор установит для нее предельно допустимую в Турбо Паскале длину - 255 символов.

Рассмотрим еще одну несложную программу. Ее назначение: ввести с клавиатуры два целых числа, найти результат деления первого числа на второе и вывести полученный результат на экран.

Пример

Program Input_0utput; *{Программа вводит два целых числа*
и выводит частное от деления 1-го на 2-е}
var
n1,n2 : Integer; *{n1 и n2 - вводимые целые}*

```
x : Real;{x - результат}
BEGIN
Write( 'n1 = ');{Сообщаем о вводе n1}
ReadLn (n1) ;{Вводим n1}
Write ( 'n2 = ');{Сообщаем о вводе n2}
ReadLn (n2);{Вводим n2}
x := n1/n2;{Находим результат}
WriteLn('n1/n2 =',x);{Выводим его}
END.
```

Прежде всего бросается в глаза появление в программе поясняющих комментариев. Комментарий в Турбо Паскале - это произвольная последовательность любых символов, обрамленная фигурными скобками. Комментарий разрешается вставлять в любое место программы, где по смыслу может стоять пробел. В качестве ограничителей комментария допускается использование фигурных скобок «{» и «}», а также пары символов: «(*» - слева от комментария и «*)» - справа от него:

```
{ Это – комментарий }
(* Это – тоже комментарий *)
```

Редактор Турбо Паскаля выделяет комментарии наклонным шрифтам (курсивом).

Комментарии с однотипными ограничителями нельзя вкладывать друг в друга, т.е. недопустимы последовательности вида

```
{...{...}...} или (*...(*...*)...*)
```

Однако можно вкладывать комментарии с ограничителями разных типов (не более одной глубины вложения):

```
{ ... (* ...*)...} или (* ... { ... } ... *)
```

Последнее обстоятельство проясняет кажущуюся странной избыточность ограничителей: если всюду в программе будут использоваться ограничители одного типа, то для того, чтобы временно исключить из программы какой-либо фрагмент текста, достаточно заключить его в ограничители другого типа.

Наличие комментариев в программе избавляет меня от необходимости пояснять назначение отдельных строк программы. Несколько слов о вводе данных. Пары операторов

```
Write (..);
ReadLn(..);
```

работают следующим образом. Вначале оператор Write выводит строку на экран и оставляет курсор в конце только что выведенной строки текста. Заметим, что оператор

```
WriteLn(Text);
```

в примере после вывода текста осуществлял перевод строки и устанавливал курсор в начало следующей строки экрана. Именно в этом простом действии (переводе строки) заключается единственное отличие в работе процедуры WriteLn от процедуры Write.

Затем по оператору ReadLn вызывается встроенная процедура ввода данных и программа останавливается в ожидании ввода. В этот момент необходимо набрать на клавиатуре нужное число и нажать клавишу *Enter*. Сразу после этого программа продолжит работу: проанализирует введенное число и перейдет к вводу следующего числа или вычислению результата. Таким образом, сигналом окончания подготовки очередного числа является нажатие на клавишу *Enter*, до этого момента можно стирать любой ошибочно введенный символ клавишей *Backspace*.

Для вычисления отношения введенных чисел используется один из основных операторов Турбо Паскаля - оператор присваивания. В его левой части указывается имя переменной, правая часть представляет собой выражение того же типа, что и переменная. Пара символов «: =», связывающая левую и правую части оператора присваивания, означает «присвоить значение». Запомним: в операторах присваивания Турбо Паскаля всегда используются символы «: =», в то время как при описании констант - одиночный символ «=». С точки зрения синтаксиса языка, два символа «: =» рассматриваются как один специальный символ и обязательно пишутся слитно.

Оператор присваивания используется практически во всех языках программирования. В некоторых языках, например в Фортране или Бейсике, символом присваивания является знак равенства, однако новичка, привыкшего к строгости математических формул, может озадачить типичная форма записи фортран-оператора присваивания, например, такая:

```
X = X + 1
```

Вариант записи этого же оператора на Турбо Паскале:

```
X := X + 1;
```

в этом смысле кажется более логичным. Разумеется, вряд ли кому-нибудь придет в голову видеть уравнения там, где их нет и не может быть. Конечно же, и в том, и в другом случае реализуется одно и то же алгоритмическое действие: к содержимому X прибавляется 1 и полученный результат вновь

присваивается переменной X. Обратите внимание на оператор вывода результатов

```
WriteLn('n1/n2 = ',x);
```

В нем в качестве одного из параметров явно указывается константа типа строка символов 'nl/n2 = '. Конечно же, константы (в отличие от переменных) вовсе не обязательно описывать в разделе описаний, так как их тип легко определяется компилятором по форме записи константы. С учетом этого можно было бы записать программу из примера предельно лаконично:

```
begin WriteLn('Я программирую на Турбо Паскале'); end.
```

ПРЕОБРАЗОВАНИЯ ТИПОВ И ДЕЙСТВИЯ НАД НИМИ

Как уже говорилось, тип переменной позволяет не только устанавливать длину ее внутреннего представления, но и контролировать те действия, которые выполняются над ней в программе. Контроль за использованием переменных еще на этапе компиляции программы - важное преимущество Турбо Паскаля перед другими языками программирования, в которых допускается автоматическое преобразование типов. В Турбо Паскале почти невозможны неявные (автоматические) преобразования типов. Исключение сделано только в отношении констант и переменных типа INTEGER (целые), которые разрешается использовать в выражениях типа REAL (вещественные). Если, например, переменные X и Y описаны следующим образом:

```
var
x: Integer;
y: Real;
```

то оператор

```
y := x + 2;
```

будет синтаксически правильным: хотя справа от знака присваивания стоит целочисленное выражение, а слева - вещественная переменная, компилятор сделает необходимые преобразования автоматически. В то же время оператор

```
x := 2.0;
```

будет неверным, так как автоматическое преобразование типа REAL (константа 2.0 содержит десятичную точку и, следовательно, принадлежит к

типу REAL) в тип INTEGER в Турбо Паскале запрещено.

Разумеется, запрет на автоматическое преобразование типов еще не означает, что в Турбо Паскале нет средств преобразования данных. Они, конечно же, есть, но их нужно использовать явно. Для преобразования данных в языке существуют встроенные функции, которые получают в качестве параметра значение одного типа, а возвращают результат в виде значения другого типа. В частности, для преобразования REAL в INTEGER имеются даже две встроенные функции такого рода: ROUND округляет REAL до ближайшего целого, а TRUNC усекает REAL путем отбрасывания дробной части.

Например, ошибочным будет оператор `x := y/x;` но правильным `x := round(y/x);`

Понятие функции в Турбо Паскале близко к понятию процедуры. Как и процедура, функция вызывается своим именем и может содержать произвольное число операторов Турбо Паскаля и даже внутренних процедур и функций. Существенным отличием функции от процедуры является то обстоятельство, что функция имеет собственное значение и, следовательно, может использоваться наравне с переменными в выражениях соответствующего типа.

Для преобразования данных типа CHAR (символ) в целое число предназначена функция ORD, обратное преобразование INTEGER в CHAR осуществляет функция CHR.

С помощью следующей несложной программы можно узнать внутренний код произвольного символа.

Пример

Program Code_pf_Char;

```
{Программа читает символ с клавиатуры и выводит на экран
этот символ несоответствующий ему внутренний код}
var
ch: Char; {В эту переменную читается символ}
begin
Write('Введите любой символ: ');
ReadLn(ch); {Читаем один символ}
WriteLn(ch, ' = ',ord(ch)); {Преобразуем его к целому и выводим на экран}
END.
```

Обратите внимание: при вызове

```
WriteLntch,' = ',ord(ch));
```

третьим параметром обращения указан вызов функции ORD (CH) , что с

точки зрения языка является выражением; как мы увидим дальше (см. гл.8), во многих случаях при вызове процедур и функций в качестве параметров вызова можно указывать не только переменные или константы, но и выражения с их участием.

По мере надобности мы будем знакомиться с другими функциями преобразования типов данных, а сейчас - о тех операциях, которые разрешены над различными типами.

Конечно же, в Турбо Паскале есть все четыре арифметические операции над переменными REAL И INTEGER:

+ - сложение;
 - - вычитание;
* - умножение;
/ - деление вещественное;
div - деление целочисленное.

Наличие двух операций деления есть еще одно проявление основополагающего принципа Турбо Паскаля: программист должен явно подтверждать компилятору, что он готов к возможным последствиям преобразования типов. Если, например, в языке Фортран используется выражение 1/2 , то результат этого выражения будет зависеть от того, переменной какого типа он будет присвоен: если N есть переменная целого типа, а X- вещественного, то в программе на Фортране присваивания

```
N = 1/2
X = 1/2
```

дадут значения 0 для N и 0.5 для X. В Турбо Паскале такой двусмысленности нет: выражение 1/2 всегда имеет значение 0.5 и поэтому оператор

```
var
N :Integer;
begin
N := 1/2;
```

просто недопустим. В то же время допустимый в Турбо Паскале оператор

```
var
X : Real;
begin
X := 1 div 2;
```

самим фактом использования операции целочисленного деления DIV свидетельствует о том, что программист сознательно отбрасывает дробную часть результата. (Надеюсь, что читатель извинит явную искусственность этих примеров, которая вызвана лишь стремлением проиллюстрировать обсуждаемые особенности языка).

Для данных типа INTEGER в Турбо Паскале есть еще одна операция MOD - получение остатка от целочисленного деления. Например:

```
5 mod 2 = 1
31 mod 16 = 15
18 mod 3 = 0
```

В Турбо Паскале отсутствует операция возведения в степень, что, очевидно, будет вызывать определенные неудобства при реализации вычислительных алгоритмов. Некоторым утешением может служить наличие встроенной функции SQR, возвращающей квадрат от значения параметра, причем тип результата определяется типом параметра.

И еще об одном существенном недостатке Турбо Паскаля: в нем отсутствуют комплексный тип и соответствующие операции над ним. Вообще, в отношении реализации разнообразных вычислительных процедур Турбо Паскаль значительно уступает некоторым другим языкам программирования, в частности, тому же Фортрану. В частности, в нем намного беднее набор встроенных математических функций (см. гл. 4).

При работе с целыми числами могут оказаться полезными две процедуры (здесь и далее в квадратных скобках указываются необязательные параметры):

DEC (X [, N]) - уменьшает содержимое переменной X на значение выражения N (если N не задано, то на 1); тип переменной X и выражения N - INTEGER (точнее, любой целый, см. гл. 4);

INC (X [, N]) - увеличивает значение X на N (если N не задано, то на 1).

Над символами и строками символов определена единственная операция - сцепление двух строк. Операция обозначается символом «+». Например, программа

```
var
st: String;
begin
st := 'Турбо'+'-'+'Паскаль';
WriteLn(st);
end.
```

напечатает строку

```
Турбо-Паскаль
```

Все остальные действия над строками и символами реализуются с помощью встроенных процедур и функций.

И, наконец, об операциях отношения и логических операциях.

Над данными типа REAL, INTEGER, CHAR, STRING определены

следующие операции отношения (сравнения):
= - равно;
<> - не равно;
< - меньше;
> - больше;
<= - меньше или равно,
>= - больше или равно.

В операциях сравнения должны участвовать однотипные операнды. Исключение сделано опять-таки в отношении REAL и INTEGER, которые могут сравниваться друг с другом. Результат применения операции отношения к любым операндам имеет тип BOOLEAN.

Сравнение двух строк осуществляется следующим образом. Символы строк сравниваются попарно друг с другом так, что первый символ первой строки сравнивается с первым символом второй строки, второй символ первой строки - со вторым символом второй и т.д. Символы сравниваются путем сравнения их кодов во внутреннем представлении (см. гл. 4). Если одна строка короче другой, недостающие символы заменяются нулем. Отношение первой несовпадающей друг с другом пары символов и принимается за отношение двух строк.

При сравнении данных типа BOOLEAN учитывается внутреннее соглашение Турбо Паскаля, в соответствии с которым FALSE есть нулевой байт, а TRUE - байт с единицей в младшем разряде. Заметим, что функция ORD преобразует к целому не только символы, но и логические величины, поэтому

```
ord(false) = 0,
ord(true) = 1.
```

В Турбо Паскале определены следующие логические операции:

not - логическое НЕ; or - логическое ИЛИ;
and - логическое И; xor - исключающее ИЛИ.

Логические операции применимы к операндам целого и логического типов. Если операнды - целые числа, то результат логической операции есть тоже целое число (подробнее об этом сказано в гл.4). Логические операции над логическими данными дают результат логического типа.

При вычислении выражений любого типа приоритет вычислений определяется расставленными скобками, а при их отсутствии - по табл. (в порядке убывания приоритета).

Приоритет	**Операция**
1	not, @

2	*, /, div, mod, and, shl , shr
3	+, - , or, xor
4	=, <>, >, >=, <, <=, in

Следует учесть, что в отличие от многих других языков программирования в Турбо Паскале логические операции имеют более высокий приоритет, чем операции отношения. В связи с этим, в сложных логических выражениях обычно необходимо расставлять скобки. Если, например, b и c имеют тип INTEGER , то выражение

```
a = b and c < d
```

вызовет сообщение о синтаксической ошибке, так как сначала выполнится операция b and c. Правильным будет выражение:

```
(a = b) and (c < d)
```

ОПЕРАТОРЫ ЯЗЫКА

С одним из наиболее часто используемых операторов языка Турбо Паскаль - оператором присваивания мы уже познакомились. Ниже рассматриваются остальные операторы языка.

Составной оператор и пустой оператор

Составной оператор - это последовательность произвольных операторов программы, заключенная в операторные скобки - зарезервированные слова begin . . . end. Составные операторы - важный инструмент Турбо Паскаля, дающий возможность писать программы по современной технологии структурного программирования (без операторов перехода GOTO).

Язык Турбо Паскаль не накладывает никаких ограничений на характер операторов, входящих в составной оператор. Среди них могут быть и другие составные операторы - Турбо Паскаль допускает произвольную глубину их вложенности:

```
begin
.......
begin
.......
begin
......
......
end;
......
end;
......
end;
```

Фактически, весь раздел операторов, обрамленный словами begin . . .

end, представляет собой один составной оператор. Поскольку зарезервированное слово end является закрывающей операторной скобкой, оно одновременно указывает и конец предыдущего оператора, поэтому ставить перед ним символ «;» необязательно, и далее во всех примерах мы не будем этого делать. Наличие точки с запятой перед end в предыдущих примерах означало, что между последним оператором и операторной скобкой end располагается пустой оператор. Пустой оператор не содержит никаких действий, просто в программу добавляется лишняя точка с запятой. В основном пустой оператор используется для передачи управления в конец составного оператора.

Условный оператор

Условный оператор позволяет проверить некоторое условие и в зависимости от результатов проверки выполнить то или иное действие. Таким образом, условный оператор - это средство ветвления вычислительного процесса.

Структура условного оператора имеет следующий вид:

```
IF <условие> THEN <оператор1> ELSE <оператор2>,
```

где IF, THEN, ELSE - зарезервированные слова (если, то, иначе); <условие> - произвольное выражение логического типа; <оператор1>, <оператор2> - любые операторы языка Турбо Паскаль.

Условный оператор работает по следующему алгоритму. Вначале вычисляется условное выражение <условие>. Если результат есть TRUE (истина), то выполняется <оператор1>, а <оператор2> пропускается; если результат есть FALSE (ложь), наоборот, <оператор1> пропускается, а выполняется <оператор2>. Например:

```
var
x, y, max: Integer;
begin
.......
if x > max then
y := max else
y := x;
```

При выполнении этого фрагмента переменная Y получит значение переменной X, если только это значение не превышает MAX, в противном случае Y станет равно MAX.

Часть ELSE <оператор2> условного оператора может быть опущена. Тогда при значении TRUE условного выражения выполняется <оператор1>, в противном случае этот оператор пропускается:

```
var
```

```
x, y, max: Integer;
begin
.......
if x > max then
max := x;
Y := x;
```

В этом примере переменная Y всегда будет иметь значение переменной X,а в MAX запоминается максимальное значение X.

Поскольку любой из операторов <оператор1> и <оператор2> может быть любого типа, в том числе и условным, а в то же время не каждый из «вложенных» условных операторов может иметь часть ELSE <оператор2>, то возникает неоднозначность трактовки условий. Эта неоднозначность в Турбо Паскале решается следующим образом: любая встретившаяся часть ELSE соответствует ближайшей к ней «сверху» части THEN условного оператора. Например:

```
var
a,b,c,d : Integer; begin
a := 1; b := 2; c= 3; d := 4;
if a > b then
if c < d then
if c < 0 then
c := 0 else
a := b; {a равно 1}
if a > b then
if c then
if c then
c := 0
else
else
else
a := b; {a равно 2}
```

Рассмотрим далее программу, которая вводит произвольное десятичное целое число в диапазоне 0...15, преобразует его к шестнадцатеричному и выводит на экран полученный результат.

Пример

```
Program Hex;
{Программа вводит с клавиатуры целое число в диапазоне от 0 до 15,
преобразует его к шестнадцатеричной системе счисления и выводит результат на
экран}
var
n : Integer; {Вводимое число}
ch : Char; {Результат}
begin
Write ( ',n = ' ) ;
ReadLn(n); { Вводим число }
{Проверяем число на принадлежность к диапазону 0...15}
if (n >= 0) and (n <= 15) then
begin {Да, принадлежит диапазону}
if n < 10 then
ch := chr(ord('0') + n)
else
```

```
ch := chr(ord('A') + n- 10);
WriteLn('n = ',ch)
end
else {Не принадлежит диапазону}
WriteLn('Ошибка')
end.
```

В шестнадцатеричной системе счисления используется 16 цифр в каждом разряде: цифры 0...9 обозначают первые 10 возможных значений разряда, буквы A...F - остальные шесть.

В программе учитывается непрерывность и упорядоченность множеств цифр 0...9, букв A...F и их кодов.

Операторы повторений

В языке Турбо Паскаль имеются три различных оператора, с помощью которых можно запрограммировать повторяющиеся фрагменты программ.

Счетный оператор цикла FOR имеет такую структуру:

```
FOR <пар_цик> := <нач_знач> TO <кон_знач> DO <оператор>.
```

Здесь FOR, TO, DO - зарезервированные слова (для, до, выполнить);

<пар_цик> - параметр цикла - переменная типа INTEGER (точнее, любого порядкового типа;

<нач_знач> - начальное значение - выражение того же типа;

<кон_знач> - конечное значение - выражение того же типа;

<оператор> - произвольный оператор Турбо Паскаля.

При выполнении оператора FOR вначале вычисляется выражение <нач_знач> и осуществляется присваивание <пар_цик> : = <нач_знач>. После этого циклически повторяется:

- проверка условия <пар_цик> <= <кон_знач>; если условие не выполнено, оператор FOR завершает свою работу;
- выполнение оператора <оператор>;
- наращивание переменной <пар_цик> на единицу.

В качестве иллюстрации применения оператора FOR рассмотрим программу, осуществляющую ввод с клавиатуры произвольного целого числа N и вычисление суммы всех целых чисел от 1 до N.

Пример

Program Summ_of_Integer;

```
{Программа вводит целое положительное число N и подсчитывает сумму всех целых чисел от 1 до N}
var
i, n, s : Integer;
begin
Write('N = ');
ReadLn(n); . {Вводим N}
s := 0; {Начальное значение суммы}
for i : = 1 to n do {Цикл подсчета суммы}
s : = s + i;
writeln('Сумма = ',s) {Выводим результат}
end.
```

Отметим два обстоятельства. Во-первых, условие, управляющее работой оператора FOR, проверяется перед выполнением оператора <оператор>: если условие не выполняется в самом начале работы оператора FOR, исполняемый оператор не будет выполнен ни разу. Другое обстоятельство - шаг наращивания параметра цикла строго постоянен и равен (+1). Существует другая форма оператора:

```
FOR<пар_цик>: = <нач_знач> DOWNTO <кон_знач> DO <оператор>
```

Замена зарезервированного слова TO на DOWNTO означает, что шаг наращивания параметра цикла равен (-1), а управляющее условие приобретает вид <пар_цик> = <кон_знач>.

Пример можно модифицировать так, чтобы сделать его пригодным для подсчета любых сумм - положительных и отрицательных:

```
.................
s := 0;
if n >= 0 then
for i := 1 to n do
s := s + i else
for i := -1 downto n do s : = s + i ;
..............
```

Два других оператора повторений лишь проверяют условие выполнения или повторения цикла, но не связаны с изменением счетчика цикла.

Оператор цикла WHILE с предпроверкой условия:

```
WHILE <условие> DO <оператор>.
```

Здесь WHILE, DO - зарезервированные слова (пока [выполняется условие], делать);

<условие> - выражение логического типа;

<оператор> - произвольный оператор Турбо Паскаля.

Если выражение <условие> имеет значение TRUE, то выполняется <оператор>, после чего вычисление выражения <условие> и его проверка повторяются. Если <условие> имеет значение FALSE , оператор WHILE прекращает свою работу.

Рассмотрим далее пример, иллюстрирующий использование оператора WHILE. Найдем так называемое «машинное эпсилон» - такое минимальное, не равное нулю вещественное число, которое после прибавления его к 1.0 еще дает результат, отличный от 1.0.

Пример

```
Program EpsilpnDetect;
{Программа вычисляет и выводит на экран значение "машинного эпсилон"}
var
epsilon: Real;
begin
epsilon := 1;
while epsilon/2 + 1 > 1 do
epsilon := epsilon/2
WriteLn('Машинное эпсилон = ',epsilon)
end.
```

Оператор выбора

Оператор выбора позволяет выбрать одно из нескольких возможных продолжений программы. Параметром, по которому осуществляется выбор, служит ключ выбора -выражение любого порядкового типа (любого из рассмотренных, кроме типов REAL и STRING, см. гл. 4).

Структура оператора выбора такова:

```
CASE <ключ_выбора> OF <список_выбора> [ELSE <операторы>] END
```

Здесь CASE, OF, ELSE, END - зарезервированные слова (случай, из, иначе, конец);

<ключ_выбора> - ключ выбора;

<список_выбора> - одна или более конструкций вида:

<константа_выбора> : <оператор>;

<константа_выбора> - константа того же типа, что и выражение<ключ_выбора> ;

<операторы> - произвольные операторы Турбо Паскаля.

Оператор выбора работает следующим образом. Вначале вычисляется значение выражения <ключ_выбора>, а затем в последовательности операторов <список_выбора> отыскивается такой, которому предшествует константа, равная вычисленному значению. Найденный оператор выполняется, после чего оператор выбора завершает свою работу. Если в списке выбора не будет найдена константа, соответствующая вычисленному значению ключа выбора, управление передается операторам, стоящим за словом ELSE. Часть ELSE <оператор> можно опускать. Тогда при отсутствии в списке выбора нужной константы ничего не произойдет и оператор выбора просто завершит свою работу.

Составим программу, имитирующую работу микрокалькулятора. Программа вводит две строки: первая содержит два произвольных числа, разделенных пробелом, вторая - символ арифметического действия, например:

```
2 2
 *
```

или

```
18.35 0.12
     /
```

Над введенными числами осуществляется соответствующее действие и результат выводится на экран. Признаком конца работы программы служит ввод любого символа, отличного от +,-,*, /.

Пример

```
Program Calc;
{Программа вводит два числа в первой строке и один из знаков +, -, *, / - во
второй и выводит на экран результат соответствующего арифметического
действия}
var
operation : Char; {Знак операции}
x, y, z : Real; {Операнды и результат}
stop : Boolean; {Признак ошибочной операции и останова}
begin
stop := false;
repeat
WriteLn; {Пустая строка-разделитель}
Write('x,y= ' ) ;
ReadLn(x,y);
Write('операция: ') ;
ReadLn(operation);
case operation of
' + ' : z : = x + y;
' - ' : z : = x - y;
' * ' : z : = x * y;
' / ' : z : = x / y;
else
stop    := true;
end;
```

```
if not  stop  then
WriteLn('результат=',z)
until stop
end.
```

Любому из операторов списка выбора может предшествовать не одна, а несколько констант выбора, разделенных запятыми. Например, следующая программа при вводе одного из символов: y или Y выведет на экран слово «Да», а при вводе n или N - слово «Нет»:

```
var
ch : Char ;
begin
ReadLn (ch) ;
case ch of
'n','N' : WriteLn ('Нет' );
'y','Y' : WriteLn ('Да')
end
end.
```

Метки и операторы перехода

Можно теоретически показать, что рассмотренных операторов вполне достаточно для написания программ любой сложности. В этом отношении наличие в языке операторов перехода кажется излишним. Более того, современная технология структурного программирования основана на принципе «программировать без GOTO»: считается, что злоупотребление операторами перехода затрудняет понимание программы, делает ее запутанной и сложной в отладке.

Тем не менее, в некоторых случаях использование операторов перехода может упростить программу.

Оператор перехода имеет вид:

```
GOTO <метка>.
```

Здесь GOTO - зарезервированное слово (перейти [на метку]); <метка> - метка.

Метка в Турбо Паскале - это произвольный идентификатор, позволяющий именовать некоторый оператор программы и таким образом ссылаться на него. В целях совместимости со стандартным языком Паскаль в языке Турбо Паскаль допускается в качестве меток использование также целых чисел без знака.

Метка располагается непосредственно перед помечаемым оператором и отделяется от него двоеточием. Оператор можно помечать несколькими метками, которые в этом случае отделяются друг от друга двоеточием. Перед тем как появиться в программе, метка должна быть описана. Описание меток

состоит из зарезервированного слова LABEL (метка), за которым следует список меток:

```
label
loop, 1b1, 1b2;
begin
.......
goto 1b1;
.......
 loop: ......
.......
1b1:1b2: ......
........
goto 1b2;
.......
```

Действие оператора GOTO состоит в передаче управления соответствующему меченному оператору.

При использовании меток необходимо руководствоваться следующими правилами:

− метка, на которую ссылается оператор GOTO, должна быть описана в разделе описаний и она обязательно должна встретиться где-нибудь в теле программы;

− метки, описанные в процедуре (функции), локализуются в ней, поэтому передача управления извне процедуры (функции) на метку внутри нее невозможна.

МАССИВЫ

Рассмотренные выше простые типы данных позволяют использовать в программе одиночные объекты - числа, символы, строки и т.п. В Турбо Паскале могут использоваться также объекты, содержащие множество однотипных элементов. Это массивы -формальное объединение нескольких однотипных объектов (чисел, символов, строк и т.п.), рассматриваемое как единое целое. К необходимости применения массивов мы приходим всякий раз, когда требуется связать и использовать целый ряд родственных величин. Например, результаты многократных замеров температуры воздуха в течение года удобно рассматривать как совокупность вещественных чисел, объединенных в один сложный объект - массив измерений.

При описании массива необходимо указать общее число входящих в массив элементов и тип этих элементов. Например:

```
var
a : array [1..10] of Real;
b : array [0..50] of Char;
```

```
c : array [-3..4] of Boolean;
```

Как видим, при описании массива используются зарезервированные слова ARRAY и OF (массив, из). За словом ARRAY в квадратных скобках указывается тип-диапазон, с помощью которого компилятор определяет общее число элементов массива. Тип-диапазон задается левой и правой границами изменения индекса массива, так что массив A состоит из 10 элементов, массив B - из 51, а массив C - из 8 элементов. За словом OF указывается тип элементов, образующих массив.

Доступ к каждому элементу массива в программе осуществляется с помощью индекса - целого числа (точнее, выражения порядкового типа, служащего своеобразным именем элемента в массиве (если левая граница типа-диапазона равна 1, индекс элемента совпадает с его порядковым номером). При упоминании в программе любого элемента массива сразу за именем массива должен следовать индекс элемента в квадратных скобках, например:

```
var
a: array [1..10] of Integer;
b: array [0..40] of Char;
c: array [-2..2] of Boolean;
k: Integer; begin
b[17] := 'F1;
c[-2] := a[l] > [2] ;
for k : = 1 to 10 do a[k] := 0;
...
end.
```

В правильно составленной программе индекс не должен выходить за пределы, определенные типом-диапазоном. Например, можно использовать элементы A[1], B[38], C[0], но нельзя A[0] или C[38] (определение массивов см. выше). Турбо Паскаль может контролировать использование индексов в программе на этапе компиляции и на этапе счета программы.

Для иллюстрации приемов работы с массивами составим программу, которая создает массив случайных целых чисел, подсчитывает их среднее арифметическое, а также определяет и выводит на экран минимальное и максимальное из этих чисел.

Пример

```
Program Average;
{Программа создает массив из N случайных целых чисел, равномерно
распределенных в диапазоне от 0 до MAX_VALUE-1, подсчитывает
среднее арифметическое этих чисел, а также минимальное и
максимальное из них.}
const
```

```
N = 1000;
MAX_VALUE = 100+1; {Диапазон значений случайных чисел}
var
m : array [1..N] of Integer; {Массив чисел}
i : Integer; {Индекс массива}
max, min : Integer; {Максимальное и минимальное число}
s : Real; {Сумма чисел}
begin
{Наполняем массив случайными числами:}
for i := 1 to N do
m[i] := random(MAX_VALUE); {Задаем начальные значения
переменных:}
s := 0;
max : = m [ 1 ] ;
min : = m [ 1 ] ;
{Цикл вычисления суммы всех случайных чисел и поиска
минимального и максимального:}
for i := 1 to N do
begin
s : = s + m [ i ] ;
if m[i] < min then
min := m[i]
else if m[i] > max then
max := m[i]
end;
{Вычисляем среднее значение и печатаем результат:}
WriteLn('ММН = ',min,' Макс = ', max, ' Среднее = ',s/N)
end.
```

Для создания массива используется встроенная функция RANDOM (MAX) , которая возвращает случайное целое число, равномерно распределенное в диапазоне от 0 до MAX-1 (MAX- параметр обращения).

ПРОЦЕДУРЫ И ФУНКЦИИ

Процедуры и функции представляют собой важный инструмент Турбо Паскаля, позволяющий писать хорошо структурированные программы. В структурированных программах обычно легко прослеживается основной алгоритм, их нетрудно понять любому читателю, они проще в отладке и менее чувствительны к ошибкам программирования. Все эти свойства являются следствием важной особенности процедур (функций), каждая из которых представляет собой во многом самостоятельный фрагмент программы, связанный с основной программой лишь с помощью нескольких параметров. Самостоятельность процедур (функций) позволяет локализовать в них все детали программной реализации того или иного алгоритмического действия и поэтому изменение этих деталей, например, в процессе отладки обычно не приводит к изменениям основной программы.

Процедурой в Турбо Паскале называется особым образом оформленный

фрагмент программы, имеющий собственное имя. Упоминание этого имени в тексте программы приводит к активизации процедуры и называется ее вызовом. Сразу после активизации процедуры начинают выполняться входящие в нее операторы, после выполнения последнего из них управление возвращается обратно в основную программу и выполняются операторы, стоящие непосредственно за оператором вызова процедуры (рис. 27).

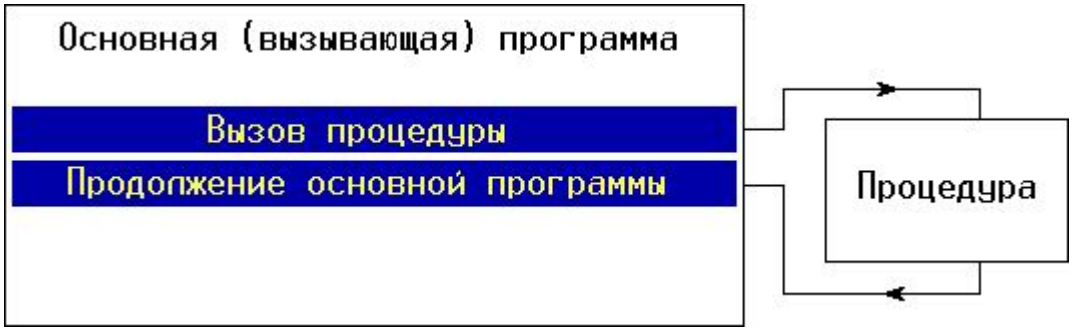


Рис.27. Взаимодействие вызывающей программы и процедуры

Для обмена информацией между основной программой и процедурой используется один или несколько параметров вызова. Процедуры могут иметь и другой механизм обмена данными с вызывающей программой, так что параметры вызова могут и не использоваться. Если они есть, то они перечисляются в круглых скобках за именем процедуры и вместе с ним образуют оператор вызова процедуры.

Функция отличается от процедуры тем, что результат ее работы возвращается в виде значения этой функции, и, следовательно, вызов функции может использоваться наряду с другими операндами в выражениях.

С примерами процедур и функций мы уже сталкивались - это стандартные процедуры чтения И записи READ, READLN, WRITE, WRITELN, функции ORD, CHR, математические функции и др. Стандартными они называются потому, что созданы одновременно с системой Турбо Паскаль и являются ее неотъемлемой частью. В Турбо Паскале имеется много стандартных процедур и функций. Наличие богатой библиотеки таких программных заготовок существенно облегчает разработку прикладных программ. Однако в большинстве случаев некоторые специфичные для данной прикладной программы действия не находят прямых аналогов в библиотеках Турбо Паскаля, и тогда программисту приходится разрабатывать свои, нестандартные процедуры и функции.

Нестандартные процедуры и функции необходимо описать, чтобы компилятор мог установить связь между оператором вызова и теми действиями, которые предусмотрены в процедуре (функции). Описание процедуры (функции) помещается в разделе описаний и внешне выглядит как программа, но вместо заголовка программы фигурирует заголовок

процедуры (функции).

Не вдаваясь в дальнейшие подробности, попробуем составить собственную процедуру, чтобы пояснить сказанное. Пусть в этой процедуре преобразуется некоторая символьная строка таким образом, чтобы все строчные буквы заменялись соответствующими прописными. В Турбо Паскале имеется стандартная функция UPCASE (см. гл.4), которая выполняет аналогичные действия над одиночным символом. Наша процедура (назовем ее UPSTRING) будет преобразовывать сразу все символы строки, причем сделаем ее пригодной не только для латинских букв, но и для букв русского алфавита.

Разработку программы проведем в два этапа. Сначала сконструируем основную (вызывающую) часть программы. Ее действия очень просты: она должна ввести входную строку (назовем ее Sinp) с клавиатуры, преобразовать ее с помощью процедуры UpString в выходную строку Sout и напечатать результат. Эти действия нетрудно запрограммировать, например:

```
Program CharsConvert;
Procedure UpString(si: String; var s2: String);
begin {UpString}
s2 := s1 {Пока еще нет преобразования!}
end; {UpString}
var
Sinp, Sout : String; {Исходная и преобразованная строки}
begin {Начало основной (вызывающей) программы}
Write('Введите строку: ');
ReadLn(Sinp); {Вводим исходную строку}
UpString(Sinp,Sout); {Преобразуем ее.к прописным буквам}
WriteLn (' Результат: ',Sout) {Печатаем результат}
end. {Конец вызывающей программы}
```

В этой программе используется замещение процедуры UPSTRING так называемой «заглушкой», т.е. процедурой, в которой на самом деле не осуществляется нужных нам действий, а выходная строка просто копирует входную. (Однако эта программа синтаксически абсолютно правильна и при желании ее можно запустить на счет.) Заглушка понадобилась нам по двум причинам. Во-первых, приведенная программа очень проста, в ней отсутствует детальная реализация процедуры и это позволяет наглядно проиллюстрировать механизм ее описания. Во-вторых, на ее примере мы знакомимся с универсальным методом конструирования сложных программ, получившим название нисходящее программирование. В соответствии с этим методом создание программы начинается «сверху», т.е. с разработки самого главного, генерального алгоритма. На верхнем уровне обычно еще не ясны детали реализации той или иной части программы, поэтому эти части следует заменить временными заглушками. Желательно, чтобы временный вариант программы был синтаксически правильным, тогда можно его

откомпилировать и убедиться в отсутствии в нем синтаксических ошибок. Такой прогон даст определенную уверенность перед разработкой и реализацией алгоритмов нижнего уровня, т.е. перед заменой заглушек реально работающими процедурами. Если реализуемый в заглушке алгоритм достаточно сложен, его вновь структурируют, выделяя главный алгоритм и применяя новые заглушки, и т.д. Процесс продолжается «вниз» до тех пор, пока не будет создан полностью работоспособный вариант программы.

В дальнейшем мы еще не раз будем использовать метод нисходящего программирования, а сейчас вернемся к описанию нашей процедуры. Как видим, это описание начинается зарезервированным словом Procedure, за которым следуют имя процедуры и список формальных параметров. Список параметров заключается в круглые скобки и содержит перечень параметров с указанием их типа. Заметим, что перед параметром s2, с помощью которого в вызывающую программу возвращается результат преобразования, стоит зарезервированное слово VAR. Именно таким способом компилятору указываются те параметры, в которых процедура возвращает вызвавшей ее программе результат своей работы. Зарезервированное слово Procedure, имя процедуры и список ее параметров образуют заголовок процедуры. За заголовком следует тело процедуры, содержащее новый раздел описаний (этот раздел пока еще пуст) и раздел исполняемых операторов (оператор s2 := s1).

Приступим к разработке алгоритма процедуры. Для этого обратимся к таблице кодировки символов, используемой в ПК. В соответствии с этой таблицей коды символов латинских строчных букв от a до z образуют непрерывный массив монотонно нарастающих чисел от 97 до 122, а коды соответствующих им прописных букв - непрерывный массив чисел от 65 до 90. Преобразование строчных латинских букв в прописные, следовательно, состоит в уменьшении кода буквы на 32. Сложнее обстоит дело с символами русского алфавита (кириллицей). В зависимости от принятого способа кодировки русские строчные буквы могут образовывать один сплошной массив (кодировки ГОСТ и MIC), два массива (альтернативная кодировка), несплошной массив (кодировка типа ЕСТЕЛ), неупорядоченный массив (кодировка КОИ-8). Если исключить два последних варианта кодировки, использовавшихся на устаревших ПК, то задача преобразования буквы состоит в том, чтобы к внутреннему коду русской буквы А (для букв от а до п) или к коду буквы Р (для букв от р до я) прибавить разницу в кодах текущего символа и кодах букв а и и. Например, если преобразуется буква б, то к коду А нужно прибавить разницу между кодами а и б, т.е. единицу, в результате получим код буквы Б. Точно так же при преобразовании буквы ф к коду буквы П будет прибавлено число 5 (как разница кодов ф и п), поэтому в результате получится код буквы Ф. С учетом этого можно составить следующий алгоритм реализации процедуры: для каждого символа исходной строки s1 определить, к какому подмассиву a...z, а...р или п...я принадлежит

код этого символа, и затем изменить его, добавив к кодам букв А (латинская), А (русская) или Я соответствующую разницу. Если символ не принадлежит ни к какому из подмассивов, нужно поместить его код в выходную строку без изменений.

Вот возможный вариант процедуры:

```
Procedure UpString(sl: String; var s2: String);
var
i: Integer; {Счетчик цикла преобразования}
c: Char; {Рабочая переменная преобразования}
begin {UpString}
s2 := ' '; {Вначале выходная строка пуста}
{Цикл посимвольного анализа исходной строки}
for i := 1 to Length(si) do
begin
{Берем из входной строки очередной символ}
c := sl[i];
{Проверяем символ на принадлежность к одному из трех
подмассивов}
if (c >= 'a') and (c <= 'z') then
c := chr(ord('A')+ord(c)-ord('a1')) {A,a - латинские!}
else if (c >= 'a') and (c <= 'n') then
c := chr(ord('A')+ord(c)-ord('a')) {A,a - русские!}
else if (c >= 'p') and (c <= 'я') then
c := chr(ord('PI)+ord(c)-ord('p'));
s2 := s2+c
end
end; {UpString}
```

В процедуре вначале с помощью оператора `s2 := '';` подготавливается «пустая» выходная строка, т.е. строка нулевой длины. Затем используется цикл от 1 до длины входной строки s1 (эта длина получается с помощью стандартной функции Length), в ходе которого проверяется принадлежность очередного символа указанным подмассивам и осуществляется необходимая коррекция его внутреннего кода. Для доступа к отдельным символам строки используется замечательное свойство типа данных STRING, позволяющее рассматривать строку как набор (массив) символов. Первый символ этого набора имеет индекс 1, второй - 2 и т.д. Индекс указывается сразу за именем строки в квадратных скобках. Таким образом, s1 [i] -это i-ый символ строки s1. Преобразованный символ добавляется в конец выходной строки.

Добавив комментарии и поместив тело процедуры вместо заглушки в первоначальный вариант программы, получим окончательно ее рабочий вариант:

Пример

```
Program CharsConvert;
```

```
{Программа вводит произвольную текстовую строку, преобразует все
входящие в нее буквы в прописные и печатает результат
преобразования}
PROCEDURE UpString(sl : String; var s2 : String);
{Эта процедура преобразует буквы входной строки si в прописные
буквы латинского или русского алфавита и помещает результат
преобразования в выходную строку s2. Используется предположение
о том, что последовательности латинских букв от «a» до «z» и
русских букв. от «а» до «п» и от «р» до «я», а также
последовательности соответствующих им прописных букв образуют
непрерывные массивы}
var
i: Integer; {Счетчик цикла преобразования}
c: Char; {Рабочая переменная преобразования}
begin {UpString}
s2 :=' ' ; {Вначале выходная строка пуста}
{Цикл посимвольного анализа исходной строки}
for i := 1 to Length(s1) do
begin
{Берем из входной строки очередной символ}
c := s1[i] ;
{Проверяем символ на принадлежность к одному из трех
подмассивов}
if (c >= 'a') and (c <= 'z') then
c := chr(ord('A')+ord(c)-ord('a')) {A,a - латинские!}
else if (c >= 'a') and (c <= 'n') then
c := chr(ord('A')+ord(c)-ord('a')) {A,a -русские!}
else if (c >= 'p') and (c <= 'я') then
c := chr(ord('P')+ord(c)-ord('p'));
s2 := s2+c
end
end; {UpString}
var
Sinp, Sout : String; {Исходная и преобразованная строки}
begin {Начало основной (вызывающей) программы}
Write('Введите строку: ');
ReadLn(Sinp); {Вводим исходную строку}
UpString(Sinp,Sout); {Преобразуем ее к прописным буквам}
WriteLn(' Результат: ',Sout) {Печатаем результат}
end. {Конец вызывающей программы}
```

Рассмотрим иной способ реализации той же программы: оформим алгоритм преобразования в виде функции. Кроме того, с помощью стандартной функции UPCASE преобразуем каждый очередной символ (это преобразование осуществляется только для букв латинского алфавита) и тем самым исключим проверку принадлежности символа к строчным латинским буквам:

```
Function UpString(s1: String): String;
var
i : Integer; c : Char;
s2: String; {Результат преобразования}
```

```
begin {UpString}
s2 := ' ';
for i := 1 to Length(si) do
begin
{Получаем и преобразуем очередной символ}
c := UpCase(si [i]);
if (c >= 'a') and (c <= 'п') then
c := chr(ord('A')+orcKcJ-ord('a'))
else
if (c >= 'p') and (c <= 'я') then
c := chr(ord('P')+ord(c)-ord('p'));
 s2 := s2+c
end;
UpString := s2 {Присваиваем значение функции UpString}
end; {UpString}
var
Sinp: String;
begin {Начало основной программы}
Write('Введите строку: ') ;
ReadLn(Sinp);
WriteLn(' Результат: ',UpString(Sinp))
 end. {Конец основной программы}
```

Программа получилась несколько проще за счет того, что функцию можно использовать в качестве параметра обращения к другой процедуре (в нашем случае к WriteLn). Обратите внимание: в теле любой функции нужно осуществить присваивание ей вычисленного значения (см. оператор UpString := s2). В левой части оператора присваивания в этом случае указывается имя функции.

ЭЛЕМЕНТЫ ЯЗЫКА

Алфавит

Алфавит языка Турбо Паскаль включает буквы, цифры, шестнадцатеричные цифры, специальные символы, пробелы и зарезервированные слова.

Буквы - это буквы латинского алфавита от a до z и от A до Z, а также знак подчеркивания _ (код ASCII 95). В Турбо Паскале нет различия между прописными и строчными буквами алфавита, если только они не входят в символьные и строковые выражения.

Цифры - арабские цифры от 0 до 9.

Каждая шестнадцатеричная цифра имеет значение от 0 до 15. Первые 10 значений обозначаются арабскими цифрами 0...9, остальные шесть - латинскими буквами A...F или a...f.

Специальные символы Турбо Паскаля - это символы

```
+ - * / = ,' . : ; < > [ ] ( ) { } ^ @ $ #
```

К специальным символам относятся также следующие пары символов:

```
<> <= >= := (* *) (. .)
```

В программе эти пары символов нельзя разделять пробелами, если они используются как знаки операций отношения или ограничители комментария. Символы (. и .) могут употребляться соответственно вместо [и].

Особое место в алфавите языка занимают пробелы, к которым относятся любые символы ASCII в диапазоне кодов от 0 до 32. Эти символы рассматриваются как ограничители идентификаторов, констант, чисел, зарезервированных слов. Несколько следующих друг за другом пробелов считаются одним пробелом (последнее не относится к строковым константам).

В Турбо Паскале имеются следующие зарезервированные слова:

```
and           end             nil         shr
asm           file            not         string
array         for             object      then
begin         function        of          to
case          goto            or          type
const         if              packed      unit
constructor   implementation  procedure   until
destructor    in              program     uses
div           inline          record      var
do            interface       repeat      while
downto        label           set         with
else          mod             shl         xor
```

Зарезервированные слова не могут использоваться в качестве идентификаторов. Стандартные директивы первоначально связаны с некоторыми стандартными объявлениями в программе. К ним относятся:

```
absolute    far         near
assembler   forward     private
external    interrupt   virtual
```

Как и зарезервированные слова, стандартные директивы в окне редактора Турбо Паскаля выделяются цветом, тем не менее Вы можете переопределить любую стандартную директиву, т.е. объявить одноименный идентификатор. Стандартные директивы PRIVATE и VIRTUAL действуют только в пределах объявления объектов.

Идентификаторы

Идентификаторы в Турбо Паскале - это имена констант, переменных, меток, типов, объектов, процедур, функций, модулей, программ и полей в записях. Идентификаторы могут иметь произвольную длину, но значащими (уникальными в области определения) являются только первые 63 символа.

Идентификатор всегда начинается буквой, за которой могут следовать буквы и цифры. Напомню, что буквой считается также символ подчеркивания, поэтому идентификатор может начинаться этим символом и даже состоять только из одного или нескольких символов подчеркивания. Пробелы и специальные символы алфавита не могут входить в идентификатор.

Примеры правильных идентификаторов:

```
a
ALPHA
MyProgramIsBestProgram
date_27_sep_39
external
_beta
```

Примеры неправильных идентификаторов:

```
1Program {начинается цифрой}
block#1 {содержит специальный символ}
My Prog {содержит пробел}
mod {зарезервированное слово}
```

Константы

В качестве констант в Турбо Паскале могут использоваться целые, вещественные и шестнадцатеричные числа, логические константы, символы, строки символов, конструкторы множеств и признак неопределенного указателя NIL.

Целые числа записываются со знаком или без него по обычным правилам и могут иметь значение от -2147483648 до +2147483647. Следует учесть, что, если целочисленная константа выходит за указанные границы, компилятор дает сообщение об ошибке. Такие константы должны записываться с десятичной точкой, т.е. определяться как вещественные числа.

Вещественные числа записываются со знаком или без него с использованием десятичной точки и/или экспоненциальной части. Экспоненциальная часть начинается символом e или E, за которым могут

следовать знаки «+» или «-» и десятичный порядок. Символ e (E) означает десятичный порядок и имеет смысл «умножить на 1.0 в степени». Например,

```
3.14E5 - 3.14 умножить на 10 в степени 5;
-17e-2 - минус 17 умножить на 10 в степени минус 2.
```

Если в записи вещественного числа присутствует десятичная точка, перед точкой и за ней должно быть хотя бы по одной цифре. Если используется символ экспоненциальной части e (E), за ним должна следовать хотя бы одна цифра десятичного порядка.

Шестнадцатеричное число состоит из шестнадцатеричных цифр, которым предшествует знак доллара $ (код 36 в ASCII). Диапазон шестнадцатеричных чисел - от $00000000 ДО $FFFFFFFF.

Логическая константа - это либо слово FALSE (ложь), либо слово TRUE (истина).

Символьная константа - это любой символ ПК, заключенный в апострофы:

```
'z' - символ z;
'Ф' - символ Ф.
```

Если необходимо записать собственно символ апострофа, он удваивается:

```
'''' - символ ' (апостроф).
```

Допускается использование записи символа путем указания его внутреннего кода, которому предшествует символ # (код 35), например:

```
#97 - символ a;
#90 - символ Z;
#39 - символ ';
#13 - символ CR.
```

Строковая константа - любая последовательность символов (кроме символа CR -возврат каретки), заключенная в апострофы. Если в строке нужно указать сам символ апострофа, он удваивается, например:

```
'Это - строка символов;
 'That' 's string.'.
```

Строка символов может быть пустой, т.е. не иметь никаких символов в обрамляющих ее апострофах. Строку можно составлять из кодов нужных символов с предшествующими каждому коду символами #, например, строка #83#121#109#98#11#108 эквивалентна строке ' Symbol'.

Наконец, в строке можно чередовать части, записанные в обрамляющих апострофах, с частями, записанными кодами. Таким способом можно вставлять в строки любые управляющие символы, в том числе и символ CR (код 13), например:

```
#7'Ошибка !'#13'Нажмите любую клавишу ...'#7 .
```

Конструктор множества - список элементов множества, обрамленный квадратными скобками, например:

```
[1,2,4..7,12]
[blue, red]
[]
[true]
```

В отличие от стандартного Паскаля, в Турбо Паскале разрешается в объявлении констант использовать произвольные выражения, операндами которых могут быть ранее объявленные нетипизированные константы, имена типов и объектов, а также следующие функции от них;

```
abs     lo     ptr     swap
chr     odd    rpund   trunc
hi      ord    sizeof
length  pred   succ
```

Например:

```
const
MaxReal = Maxlnt div SizeOf(real);
NumChars = ord('Z') - ord('a') + 1;
Ln10 = 2.302585092994;
Ln10R = 1 / Lnl0;.
```

Выражения

Основными элементами, из которых конструируется исполняемая часть программы, являются константы, переменные и обращения к функциям. Каждый из этих элементов характеризуется своим значением и принадлежит к какому-либо типу данных. С помощью знаков операций и скобок из них можно составлять выражения, которые фактически представляют собой правила получения новых значений.

Частным случаем выражения может быть просто одиночный элемент, т.е. константа, переменная или обращение к функции. Значение такого выражения имеет, естественно, тот же тип, что и сам элемент. В более общем случае выражение состоит из нескольких элементов (операндов) и знаков операций, а тип его значения определяется типом операндов и видом примененных к ним операций. Примеры выражений:

```
Y

21

(a + b) * c

sin(t)

a > 2

not Flag and (a = b)

NIL

[1, 3..7] * set1
```

Операции

В Турбо Паскале определены следующие операции:

унарные	`not, @;`
мультипликативные	`*, /, div, mod, and, shl, shr;`
аддитивные	`+, -, or, xor;`
отношения	`=, <>, <, >, <=, >=,in.`

Приоритет операций убывает в указанном порядке, т.е. наивысшим приоритетом обладают унарные операции, низшим - операции отношения. Порядок выполнения нескольких операций равного приоритета устанавливается компилятором из условия оптимизации кода программы и не обязательно слева направо. При исчислении логических выражений операции равного приоритета всегда вычисляются слева направо, причем будут вычисляться все или только достаточные операции в зависимости от установленной в среде Турбо Паскаля опции OPTIONS/COMPILER/ COMPLETE BOOLEAN EVAL: при установленном значении этой опции вычисляются все операции отношения, при не установленном - только те, которые достаточны для получения результата.

Это обстоятельство необходимо учитывать при использовании операций отношения с функциями, в которых изменяются глобальные переменные или параметры, передаваемые по имени, например:

```
Function AddI(var x: Integer): Integer;
begin {AddI}
inc(x);
AddI := x end {AddI} ;
var
a,b : Integer;
begin {main}
if (a > b) or (Addl (a) > 100) then b := a;
```

. . . . . .

При выполнении этого фрагмента значение переменной A будет зависеть от настройки опции: если опция активизирована, значение A всегда наращивается на 1, если не активизирована - только в случае A <= B .

Правила использования операций с операндами различного типа приведены в таблице.

Таблица

Операция	Действие	Тип операндов	Тип результата
not	Отрицание	Логический	Логический
not	То же	Любой целый	Тип операнда
@	Адрес	Любой	Указатель
*	Умножение	Любой целый	Наименьший целый
*	То же	Любой вещественный	Exended
*	Пересечение множеств	Множественный	Множественный
/	Деление	Любой вещественный	Extended
div	Целочисленное деление	Любой целый	Наименьший целый
mod	Остаток от деления	То же	То же
and	Логическое И	Логический	Логический
and	То же	Любой целый	Наименьший целый
shl	Левый сдвиг	То же	То же
shr	Правый сдвиг	То же	То же
+	Сложение	То же	То же
+	То же	Любой вещественный	Extended
+	Объединение множеств	Множественный	Множественный
+	Сцепление строк	Строковый	Строковый
-	Вычитание	Любой целый	Наименьший целый
-	То же	Любой вещественный	Extenden
or	Логическое ИЛИ	Логический	Логический
or	Тоже	Любой целый	Наименьший целый
=	Равно	Любой простой или строковый	Логический
<>	Не равно	То же	То же
<	Меньше	Логический	Логический
<=	Меньше или равно	То же	То же
>	Больше	То же	То же
>=	Больше или равно	То же	То же

При действиях с вещественным типом одним из операндов может быть значение любого целого типа. Результат операций имеет указанный в таблице тип EXTENDED только для установленного в среде Турбо Паскаля режима генерации кода, рассчитанного на арифметический сопроцессор или на его эмуляцию. Если этот режим не установлен, результат будет иметь значение типа REAL.

Унарная операция @ применяется к операнду любого типа и возвращает результат типа POINTER, в котором содержится адрес операнда.

Если операция @ применяется к процедуре, функции или методу в объекте, ее результатом будет адрес точки входа в эту процедуру (функцию, метод). Этот адрес можно использовать только в подпрограмме, написанной на ассемблере, или в фрагментах INLINE.

В Турбо Паскале определены следующие логические операции:

not - логическое НЕ;
and - логическое И;
or - логическое ИЛИ;
xor - исключительное ИЛИ.

Логические операции применимы к операндам целого и логического типов. Если операнды - целые числа, то результат логической операции есть тоже целое число, биты которого (двоичные разряды) формируются из битов операндов по правилам, указанным в таблице:

Логические операции над данными типа INTEGER (поразрядно)					
Операнд 1	**Операнд 2**	**not**	**and**	**or**	**xor**
1	-	0	-	-	-
0	-	1	-	-	-
0	0	-	0	0	0
0	1	-	0	1	1
1	0	-	0	1	1
1	1	-	1	1	0

К логическим же в Турбо Паскале обычно относятся и две сдвиговые операции над целыми числами:

i shl j - сдвиг содержимого i на j разрядов влево; освободившиеся младшие

разряды заполняются нулями;

i shr j - сдвиг содержимого i на j разрядов вправо; освободившиеся старшие

разряды заполняются нулями.

В этих операциях i и j - выражения любого целого типа.

С помощью программы следующего примера можно вывести на экран результат применения логических операций к двум целым числам.

Пример

{Программа вводит два целых числа и печатает результат применения к ним логических операций. Для выхода из программы ввести Ctrl-z и нажать Enter}

```
var
n,m : integer; begin
while not EOF do begin
Write('n,m='); ReadLn(n,m);
WriteLn( ' not= 1 , not n, 'not m);
WriteLnC  and= ' , n and m)
WriteLnC  or = 1 , n or m) ;
WriteLnC  xor= 1 , n xor m)
WriteLn( ' shl= 1 ,n shl m)
WriteLn( ' shr= 1 , n shr m)
end
end.
```

В программе организуется ввод двух произвольных целых чисел и печать результата применения к ним всех логических операций. Для выхода из программы следует нажать Ctrl-z, и Enter.

Логические операции над логическими данными дают результат логического типа по правилам, указанным в таблице:

Логические операции над данными типа Boolean					
Операнд 1	**Операнд 2**	**not**	**and**	**or**	**xor**
True	-	False	-	-	-
False	-	True	-	-	-
False	False	-	False	False	False
False	True	-	False	True	True
True	False	-	False	True	True
True	True	-	True	True	False

Операция отношения IN применяется к двум операндам. Первым (левым) операндом должно быть выражение любого порядкового типа,

вторым - множество, состоящее из элементов того же типа, или идентификатор множественного типа. Операция дает TRUE, если левый операнд принадлежит множеству, например:

```
var
c: char; type
digit = set of ' 0 '..' 9 ' ; begin
    if c in digit then .......
```

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 (пример 18). 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 представленном ниже, 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 8.5) на EXTENDED, программа перестанет работать уже при N = 8. Чтобы избежать переполнения стека сопроцессора, следует размещать промежуточные результаты во вспомогательной переменной. Вот правильный вариант этого 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.

ИСПОЛЬЗОВАНИЕ БИБЛИОТЕКИ GRAPH

Начиная с версии 4.0, в состав Турбо Паскаля включена мощная библиотека графических подпрограмм Graph, остающаяся практически неизменной во всех последующих версиях. Библиотека содержит в общей сложности более 50 процедур и функций, предоставляющих программисту самые разнообразные возможности управления графическим экраном. Для облегчения знакомства с библиотекой все входящие в нее процедуры и функции сгруппированы по функциональному принципу.

Переход в графический режим и возврат в

Стандартное состояние ПК после его включения, а также к моменту запуска программы из среды Турбо Паскаля соответствует работе экрана в текстовом режиме, поэтому любая программа, использующая графические средства компьютера, должна определенным образом инициировать графический режим работы дисплейного адаптера. После завершения работы программы ПК возвращается в текстовый режим.

Настройка графических процедур на работу с конкретным адаптером достигается за счет подключения нужного графического драйвера. Драйвер - это специальная программа, осуществляющая управление теми или иными техническими средствами ПК. Графический драйвер, как это не трудно догадаться, управляет дисплейным адаптером в графическом режиме. Графические драйверы разработаны фирмой Borland практически для всех типов адаптеров. Обычно они располагаются на диске в отдельном подкаталоге BGI в виде файлов с расширением BGI (от англ.: Borland Graphics Interface - графический интерфейс фирмы Borland). Например, CGA.BGI - драйвер для CG4-адаптера, EGA VGA.BGI - драйвер для адаптеров EGA и VGA и т.п.

Адаптер CGA (Color Graphics Adapter - цветной графический адаптер) имеет 5 графических режимов. Четыре режима соответствуют низкой разрешающей способности экрана (320 пикселей по горизонтали и 200 по вертикали, т.е. 320x200) и отличаются только набором допустимых цветов - палитрой. Каждая палитра состоит из трех цветов, а с учетом черного цвета несветящегося пикселя - из четырех: палитра 0 (светло-зеленый, розовый, желтый), палитра 1 (светло-бирюзовый, малиновый, белый), палитра 2 (зеленый, красный, коричневый) и палитра 3 (бирюзовый, фиолетовый, светло-серый). Пятый режим соответствует высокому разрешению 640x200, но каждый пиксель в этом случае может светиться либо каким-то одним заранее выбранным и одинаковым для всех пикселей цветом, либо не светиться вовсе, т.е. палитра этого режима содержит два цвета. В графическом режиме адаптер CGA использует только одну страницу.

Адаптер EGA (Enhanced Graphics Adapter - усиленный графический адаптер) может полностью эмулировать графические режимы адаптера CGA. Кроме того, в нем возможны режимы: низкого разрешения (640x200, 16 цветов, 4 страницы) и высокого разрешения (640x350, 16 цветов, 1 страница). В некоторых модификациях используется также монохромный режим (640x350, 1 страница, 2 цвета).

Адаптер MCGA (Multi-Color Graphics Adapter - многоцветный графический адаптер) совместим с CGA и имеет еще один режим - 640x480, 2 цвета, 1 страница. Такими адаптерами оснащались младшие модели серии ПК PS/2 фирмы IBM. Старшие модели этой серии оснащаются более совершенными адаптерами VGA (Video Graphics Array -графический видеомассив. Адаптер VGA эмулирует режимы адаптеров CGA и EGA и дополняет их режимом высокого разрешения (640x480, 16 цветов, 1 страница).

Не так давно появились так называемые cynep-VGA адаптеры (SVGA) с разрешением 800x600 и более, использующие 256 и более цветовых оттенков. В настоящее время эти адаптеры получили повсеместное распространение, однако в библиотеке Graph для них нет драйверов. Поскольку SVGA совместимы с VGA, для управления современными графическими адаптерами приходится использовать драйвер EGAVGA.BGI и довольствоваться его относительно скромными возможностями.

Несколько особняком стоят достаточно популярные адаптеры фирмы Hercules. Адаптер HGC имеет разрешение 720x348, его пиксели могут светиться одним цветом (обычно светло-коричневым) или не светиться вовсе, т.е. это монохромный адаптер. Адаптер HGC+ отличается несущественными усовершенствованиями, а адаптер HICC (Hercules In Color Card) представляет собой 16-цветный вариант HGC+.

Процедуры и функции

Процедура InitGraph. Инициирует графический режим работы адаптера. Заголовок процедуры:

```
Procedure InitGraph(var Driver,Mode: Integer; Path: String);
```

Здесь Driver - переменная типа Integer, определяет тип графического драйвера; Mode - переменная того же типа, задающая режим работы графического адаптера; Path - выражение типа String, содержащее имя файла драйвера и, возможно, маршрут его поиска.

Функция GraphResult. Возвращает значение типа Integer, в котором закодирован результат последнего обращения к графическим процедурам. Если ошибка не обнаружена, значением функции будет ноль, в противном случае - отрицательное число, имеющее следующий смысл:

Функция GraphErrorMsg. Возвращает значение типа String, в котором по указанному коду ошибки дается соответствующее текстовое сообщение. Заголовок функции:

```
Function GraphErrorMsg(Code: Integer): String;
```

Здесь Code - код ошибки, возвращаемый функцией GraphResult.

Процедура CloseGraph. Завершает работу адаптера в графическом режиме и восстанавливает текстовый режим работы экрана. Заголовок:

```
Procedure CloseGraph;
```

Процедура RestoreCRTMode. Служит для кратковременного возврата в текстовый режим. В отличие от процедуры CloseGraph не сбрасываются установленные параметры графического режима и не освобождается память, выделенная для размещения графического драйвера. Заголовок:

```
Procedure RestoreCRTMode;
```

Функция GetGraphMode. Возвращает значение типа Integer, в котором содержится код установленного режима работы графического адаптера. Заголовок:

```
Function GetGraphMode: Integer;
```

Процедура SetGraphMode. Устанавливает новый графический режим работы адаптера. Заголовок:

```
Procedure SetGraphMode(Mode: Integer);
```

Здесь Mode - код устанавливаемого режима.

Следующая программа иллюстрирует переход из графического режима в текстовый и обратно:

```
Uses Graph;
var .
Driver, Mode, Error : Integer;
begin
{Инициируем графический режим}
Driver := Detect;
InitGraph(Driver, Mode, '');
Error := GraphResult; {Запоминаем результат}
if Error <> grOk then {Проверяем ошибку}
WriteLn(GraphErrorMsg(Error)) {Есть ошибка}
else
begin {Нет ошибки}
WriteLn ('Это графический режим');
WriteLn ('Нажмите "Enter"...':20);
ReadLn;
{Переходим в текстовый режим}
RestoreCRTMode;
WriteLn (' А это текстовый...');
ReadLn;
{Возвращаемся в графический режим}
SetGraphMode (GetGraphMode);
WriteLn ('Опять графический режим...');
ReadLn;
CloseGraph
end
end.
```

В этом примере для вывода сообщений как в графическом, так и в текстовом режиме используется стандартная процедура WriteLn. Если Ваш ПК оснащен нерусифицированным адаптером CGA, вывод кириллицы в графическом режиме таким способом невозможен, в этом случае замените соответствующие сообщения так, чтобы использовать только латинские буквы.

Координаты, окна, страницы

Многие графические процедуры и функции используют указатель текущей позиции на экране, который в отличие от текстового курсора невидим. Положение этого указателя, как и вообще любая координата на графическом экране, задается относительно левого верхнего угла, который, в свою очередь, имеет координаты 0,0. Таким образом, горизонтальная координата экрана увеличивается слева направо, а вертикальная - сверху вниз.

Функции GetMaxX и GetMaxY. Возвращают значения типа Word, содержащие максимальные координаты экрана в текущем режиме работы соответственно по горизонтали и вертикали. Например:

```
Uses Graph;
var
a,b: Integer;
```

```
begin
a := Detect; InitGraph(a, b, '');
WriteLn(GetMaxX, GetMaxY:5);
ReadLn;
CloseGraph
end.
```

Функции GetX и GetY. Возвращают значения типа Integer, содержащие текущие координаты указателя соответственно по горизонтали и вертикали. Координаты определяются относительно левого верхнего угла окна или, если окно не установлено, экрана.

Процедура SetViewPort. Устанавливает прямоугольное окно на графическом экране. Заголовок:

```
Procedure SetViewPort(XI,Y1,X2,Y2: Integer; ClipOn: Boolean);
```

Здесь X1...Y2 - координаты левого верхнего (XI,Y1) и правого нижнего (X2,Y2) углов окна; CHpOn - выражение типа Boolean, определяющее «отсечку» не умещающихся в окне элементов изображения.

Координаты окна всегда задаются относительно левого верхнего угла экрана. Если параметр ClipOn имеет значение True, элементы изображения, не умещающиеся в пределах окна, отсекаются, в противном случае границы окна игнорируются. Для управления этим параметром можно использовать такие определенные в модуле константы:

```
const
ClipOn = True; {Включить отсечку}
ClipOff = False; {Не включать отсечку}
```

Процедура MoveTo. Устанавливает новое текущее положение указателя. Заголовок:

```
Procedure MoveTo(X,Y: integer);
```

Здесь X, Y - новые координаты указателя соответственно по горизонтали и вертикали.

Координаты определяются относительно левого верхнего угла окна или, если окно не установлено, экрана.

Процедура ClearDevice. Очищает графический экран. После обращения к процедуре указатель устанавливается в левый верхний угол экрана, а сам экран заполняется цветом фона, заданным процедурой SetBkColor. Заголовок:

```
Procedure ClearDevice;
```

Процедура ClearViewPort. Очищает графическое окно, а если окно не

определено к этому моменту - весь экран. При очистке окно заполняется цветом с номером О из текущей палитры. Указатель перемещается в левый верхний угол окна. Заголовок:

```
Procedure ClearViewPort;
```

Линии и точки

Процедура PutPixel. Выводит заданным цветом точку по указанным координатам. Заголовок:

```
Procedure PutPixel(X,Y: Integer; Color: Word);
```

Здесь X, Y- координаты точки; Color - цвет точки.

Координаты задаются относительно левого верхнего угла окна или, если окно не установлено, относительно левого верхнего угла экрана.

Следующая программа периодически выводит на экран «звездное небо» и затем гасит его. Для выхода из программы нажмите любую клавишу.

```
Uses CRT, Graph;
type
PixelType = record
x, y : Integer; end;
const
N = 5000; {Количество "звезд"}
var
d,r,e,k: Integer;
x1,y1,x2,y2: Integer;
a: array [1..N] of PixelType; {Координаты}
begin
{Инициируем графику}
d := Detect; InitGraph(d, r, ' ') ;
e := GraphResult; if e<>grOk then
WriteLn(GraphErrorMsg(e))
else
begin
{Создаем окно в центре экрана}
x1 := GetMaxX div 4;
y1 := GetMaxY div 4;
x2 := 3*x1;
y2 := 3*y1;
Rectangle(x1,y1,x2,y2);
SetViewPort(x1+1,y1+1,x2-1,y2-1,ClipOn);
{Создаем и запоминаем координаты всех "звезд"}
for k := 1 to N do with a[k] do begin
x := Random(x2-x1);
y := Random(y2-y1)
end;
{Цикл вывода}
repeat
for k := 1 to N do
with a[k] do {Зажигаем "звезду"}
PutPixel(x,y,white);
if not KeyPressed then
for k := N downto 1 do with a[k] do {Гасим "звезду"}
```

```
PutPixel(x,y,black)
until KeyPressed;
while KeyPressed do k := ord(ReadKey);
CloseGraph
end;
end.
```

Функция GetPixel. Возвращает значение типа Word, содержащее цвет пикселя с указанными координатами. Заголовок:

```
Function GetPixel(X,Y: Integer): Word;
```

Здесь X, Y - координаты пикселя.

Процедура Line. Вычерчивает линию с указанными координатами начала и конца. Заголовок:

```
Procedure Line(X1,Y1,X2,Y2: Integer);
```

Здесь XL. .Yl - координаты начала (XI, Y1) и конца (X2, Y2) линии.

Линия вычерчивается текущим стилем и текущим цветом.

Процедура LineTo. Вычерчивает линию от текущего положения указателя до положения, заданного его новыми координатами. Заголовок:

```
Procedure LineTo(X,Y: Integer);
```

Здесь X, Y - координаты нового положения указателя, они же - координаты второго конца линии.

Процедура LineRel. Вычерчивает линию от текущего положения указателя до положения, заданного приращениями его координат. Заголовок:

```
Procedure LineRel (DX, DY: Integer);
```

Здесь DX, DY- приращения координат нового положения указателя. В процедурах LineTo и LineRel линия вычерчивается текущим стилем и текущим цветом.

Процедура SetLineStyle. Устанавливает новый стиль вычерчиваемых линий. Заголовок:

```
Procedure SetLineStyle(Type,Pattern,Thick: Word)
```

Здесь Type, Pattern, Thick - соответственно тип, образец и толщина линии. Тип линии может быть задан с помощью одной из следующих констант:

```
const
SolidLn= 0; {Сплошная линия}
```

```
DottedLn= 1; {Точечная линия}
CenterLn= 2; {Штрих-пунктирная линия}
DashedLn= 3; {Пунктирная линия}
UserBitLn= 4; {Узор линии определяет пользователь}
```

Параметр Pattern учитывается только для линий, вид которых определяется пользователем (т.е. в случае, когда Type = UserBitLn). При этом два байта параметра Pattern определяют образец линии: каждый установленный в единицу бит этого слова соответствует светящемуся пикселю в линии, нулевой бит - несветящемуся пикселю. Таким образом, параметр Pattern задает отрезок линии длиной в 16 пикселей. Этот образец периодически повторяется по всей длине линии.

Параметр Thick может принимать одно из двух значений:

```
const
NormWidth = 1; {Толщина в один пиксель}
ThickWidth = 3; {Толщина в три пикселя}
```

Отметим, что установленный процедурой стиль линий (текущий стиль) используется при построении прямоугольников, многоугольников и других фигур.

В следующем примере демонстрируются линии всех стандартных стилей, затем вводятся слово-образец и линия с этим образцом заполнения (рис. 28).

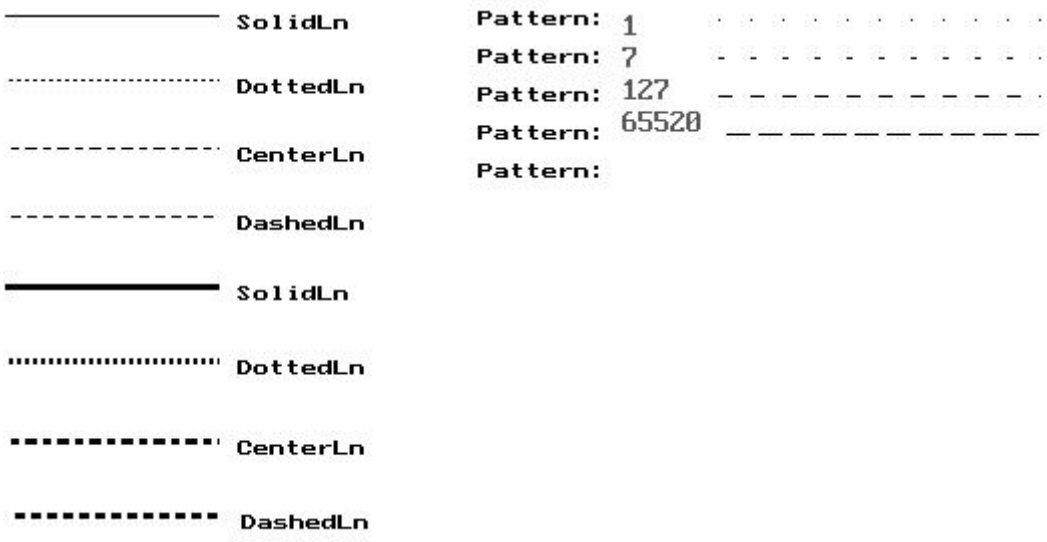


Рис. 28. Образцы линий

Процедура GetLineSettings. Возвращает текущий стиль линий. Заголовок:

```
Procedure GetLineSettings(var StyleInfo: LineSettingsType)
```

Здесь StyleInfo - переменная типа LineSettingsType, в которой возвращается текущий стиль линий.

Тип LineSettingsType определен в модуле Graph следующим образом:

```
type
LineSettingsType = record
LineStyle: Word; {Тип линии}
Pattern : Word; {Образец}
Thickness: Word {Толщина}
end;
```

Многоугольники

Процедура Rectangle. Вычерчивает прямоугольник с указанными координатами углов. Заголовок:

```
Procedure Rectangle(X1,Y1,X2,Y2: Integer);
```

Здесь X1... Y2 - координаты левого верхнего (X1, Y1) и правого нижнего (X2, Y2) углов прямоугольника. Прямоугольник вычерчивается с использованием текущего цвета и текущего стиля линий.

В следующем примере на экране вычерчиваются 10 вложенных друг в друга прямоугольников.

```
Uses Graph, CRT;
var
d,r,e,xl,yl, x2,y2,dx,dy: Integer;
begin
{Инициируем графику}
d := Detect; InitGraph(d, r, ' ') ;
e := GraphResult; if e <> grOK then
WriteLn(GraphErrorMsg(e))
else
begin
{Определяем приращения сторон}
dx := GetMaxX div 20;
dy := GetMaxY div 20;
{Чертим вложенные прямоугольники}
for d := 0 to 9 do
Rectangle(d*dx,d*dy,GetMaxX-d*dx,GetMaxY-d*dy);
if ReadKey=#0 then d := ord(ReadKey);
CloseGraph
end
end.
```

Процедура DrawPoly. Вычерчивает произвольную ломаную линию, заданную координатами точек излома.

```
Procedure DrawPoly(N: Word; var Points)
```

Здесь N - количество точек излома, включая обе крайние точки; Points - переменная типа PointType, содержащая координаты точек излома.

Координаты точек излома задаются парой значений типа Word: первое определяет горизонтальную, второе - вертикальную координаты. Для них можно использовать следующий определенный в модуле тип:

```
type
PointType = record
x, y : Word
end;
```

При вычерчивании используется текущий цвет и текущий стиль линий.

Дуги, окружности, эллипсы

Процедура Circle. Вычерчивает окружность. Заголовок:

```
Procedure Circle(X,Y: Integer; R: Word);
```

ЗдесьX, Y- координаты центра; R - радиус в пикселях.

Окружность выводится текущим цветом. Толщина линии устанавливается текущим стилем, вид линии всегда SolidLn (сплошная). Процедура вычерчивает правильную окружность с учетом изменения линейного размера радиуса в зависимости от его направления относительно сторон графического экрана, т.е. с учетом коэффициента GetAspectRatio. В связи с этим параметр R определяет количество пикселей в горизонтальном направлении.

В следующем примере в центре экрана создается окно, постепенно заполняющееся случайными окружностями. Для выхода из программы нажмите на любую клавишу.

```
Uses Graph, CRT;
var
d,r,e,x,y: Integer;
begin.
{Инициируем графику}
d i= Detect; InitGraph(d, r, '');
e := GraphResult; if e <> grOK then
WriteLn(GraphErrorMsg(e))
else
begin
{Создаем окно в центре экрана}
x := GetMaxX div 4;
y := GetMaxY div 4;
Rectangle(x,y,3*x,3*y);
SetViewPort(x+1,y+1,3*x-1,3*y-1,ClipOn);
{Цикл вывода случайных окружностей}
repeat
SetColor(succ(Random(white))); {Случайный цвет}
SetLineStyle(0,0,2*Random(2)+1); {и стиль линии}
x := Random(GetMaxX); {Случайное положение}
y := Random(GetMaxY); {центра окружности}
Circle(x,y,Random(GetMaxY div 4));
until KeyPressed;
if ReadKey=#0 then x := ord(ReadKey);
```

```
CloseGraph
end
end.
```

Процедура Arc. Чертит дугу окружности. Заголовок:

```
Procedure Arc(X,Y: Integer; BegA,EndA,R: Word);
```

Здесь X, Y - координаты центра; BegA, EndA - соответственно начальный и конечный углы дуги; R - радиус.

Углы отсчитываются против часовой стрелки и указываются в градусах. Нулевой угол соответствует горизонтальному направлению вектора слева направо. Если задать значения начального угла 0 и конечного - 359, то будет выведена полная окружность. При вычерчивании дуги окружности используются те же соглашения относительно линий и радиуса, что и в процедуре Circle.

Вот как выглядят две дуги: одна с углами 0 и 90, вторая 270 и 540 градусов (рис. 29):

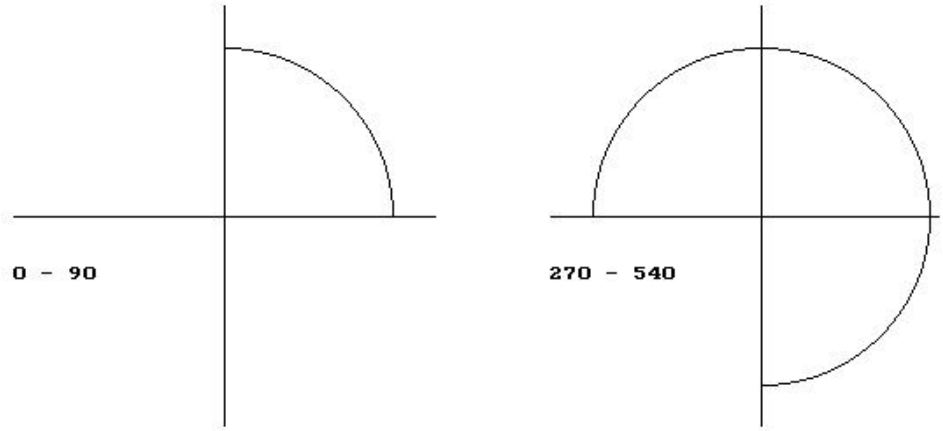


Рис. 29. Иллюстрация процедуры Arc

Процедура GetArcCoords. Возвращает координаты трех точек: центра, начала и конца дуги. Заголовок:

```
Procedure GetArcCoords(var Coords: ArcCoordsType);
```

Здесь Coords - переменная типа ArcCoordsType, в которой процедура возвращает координаты центра, начала и конца дуги.

Тип ArcCoordsType определен в модуле Graph следующим образом:

```
type
ArcCoordsType = record
X,Y : Integer; {Координаты центра}
Xstart,Ystart: Integer; {Начало дуги}
Xend,Yend : Integer; {Конец дуги}
```

```
end;
```

Совместное использование процедур Arc и GetArcCoords позволяет вычерчивать сопряжения двух прямых с помощью дуг.

Процедура Ellipse. Вычерчивает эллипсную дугу. Заголовок:

```
Procedure Ellipse(X,Y: Integer; BegA,EndA,RX,RY: Word);
```

Здесь X, Y - координаты центра; BegA, EndA - соответственно начальный и конечный углы дуги; RX, RY- горизонтальный и вертикальный радиусы эллипса в пикселях.

При вычерчивании дуги эллипса используются те же соглашения относительно линий, что и в процедуре Circle, и те же соглашения относительно углов, что и в процедуре Arc. Если радиусы согласовать с учетом масштабного коэффициента GetAspectRatio, будет вычерчена правильная окружность.

В следующей программе вычерчиваются три эллипсных дуги (рис. 30) при разных отношениях радиусов. Чем выше разрешение графического экрана, тем ближе к единице отношение сторон и тем меньше первый график отличается от третьего.

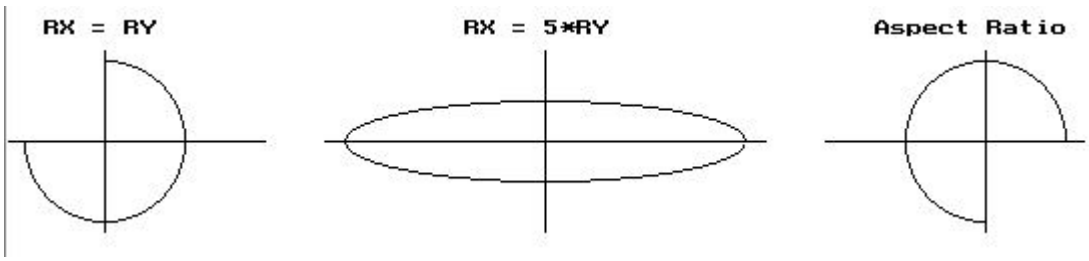


Рис.30. Эллипсные дуги

```
Uses Graph, CRT;
var
d,r,e: Integer;
xa,ya: Word;
begin
{Инициируем графику}
d := Detect; InitGraph(d, r, '');
e := GraphResult; if e <> grOK then
WriteLn(GraphErrorMsg(e))
else
begin
{Первый график}
OutTextXY(5 0,4 0,'RX = RY'); {Надпись}
Line (0,100,160,100); {Ось X}
Line (80,55,80,145); {Ось Y}
Ellipse (80,100,180,90,40,40);
{Второй график}
OutTextXY(260,40,'RX = 5*RY');
Line (190,100,410,100);
Line (300,55,300,145);
```

```
Ellipse (300,100,0,359,100,20);
{Третий график}
OutTextXY(465,40,'Aspect Ratio');
Line (440,100,600,100);
Line (520,55,520,145);
GetAspectRatio(xa, ya);
Ellipse (520,100,0,270,40,round(40*(xa/ya)));
if ReadKey=#0 then
 d := ord(ReadKey);
CloseGraph
end
end.
```

Краски, палитры, заполнения

Процедура SetColor. Устанавливает текущий цвет для выводимых линий и символов. Заголовок:

```
Procedure SetColor(Color: Word);
```

Здесь Color - текущий цвет.

Функция GetColor. Возвращает значение типа Word, содержащее код текущего цвета. Заголовок:

```
Function GetColor: Word;
```

Функция GetMaxColor. Возвращает значение типа Word, содержащее максимальный доступный код цвета, который можно использовать для обращения к SetColor. Заголовок:

```
Function GetMaxColor: Word;
```

Процедура SetBkColor. Устанавливает цвет фона. Заголовок:

```
Procedure SetBkColor(Color: Word);
```

Здесь Color - цвет фона.

В отличие от текстового режима, в котором цвет фона может быть только темного оттенка, в графическом режиме он может быть любым. Установка нового цвета фона немедленно изменяет цвет графического экрана. Это означает, что нельзя создать изображение, два участка которого имели бы разный цвет фона. Для CGA -адаптера в режиме высокого разрешения установка цвета фона изменяет цвет активных пикселей. Замечу, что после замены цвета фона на любой, отличный от 0 (Black) цвет, Вы не сможете более использовать цвет 0 как черный, он будет заменяться на цвет фона, т.к. процедуры модуля Graph интерпретируют цвет с номером 0 как цвет фона. Это означает, в частности, что Вы уже не сможете вернуть фону черный цвет!

Функция GetBkColor. Возвращает значение типа Word, содержащее текущий цвет фона. Заголовок:

```
Function GetBkColor: Word;
```

Процедура SetPalette. Заменяет один из цветов палитры на новый цвет. Заголовок:

```
Procedure SetPalette(N: Word; Color: Shortlnt);
```

Здесь N - номер цвета в палитре; Color - номер вновь устанавливаемого цвета.

Например, если выполнить оператор

```
SetPalette(2,White);
```

то цвет с индексом 2 (первоначально это - бирюзовый цвет Cyan) будет заменен на белый. Замечу, что цвет с индексом 0 отождествляется с цветом фона и может изменяться наряду с любым другим цветом.

Процедура GetPalette. Возвращает размер и цвета текущей палитры. Заголовок:

```
Procedure GetPalette(var Palettelnfo: PaletteType);
```

Здесь Palettelnfo - переменная типа PaletteType, возвращающая размер и цвета палитры.

В модуле Graph определена константа

```
const
MaxColors =15;
```

и тип

```
type
PaletteType = record
Size : Word; {Количество цветов в палитре}
Colors : array [0..MaxColors] of Shortlnt
{Номера входящих в палитру цветов}
end;
```

Процедура SetAllPalette. Изменяет одновременно несколько цветов палитры. Заголовок процедуры:

```
Procedure SetAllPalette(var Palette);
```

Параметр Palette в заголовке процедуры описан как нетипизированный параметр. Первый байт этого параметра должен содержать длину N палитры, остальные N байты - номера вновь устанавливаемых цветов в диапазоне от -1

до MaxColors. Код -1 означает, что соответствующий цвет исходной палитры не меняется.

Функция GetPaletteSize. Возвращает значение типа Integer, содержащее размер палитры (максимальное количество доступных цветов). Заголовок:

```
Function GetPaletteSize: Integer;
```

Процедура GetDefaultPalette. Возвращает структуру палитры, устанавливаемую по умолчанию (в режиме автонастройки). Заголовок:

```
Procedure GetDefaultPalette(var Palette: PaletteType);
```

Здесь Palette - переменная типа PaletteType (см. процедуру GetPalette), в которой возвращаются размер и цвета палитры.

Процедура SetFillStyle. Устанавливает стиль (тип и цвет) заполнения. Заголовок:

```
Procedure SetFillStyle(Fill,Color: Word);
```

Здесь Fill - тип заполнения; Color - цвет заполнения.

С помощью заполнения можно покрывать какие-либо фрагменты изображения периодически повторяющимся узором. Для указания типа заполнения используются следующие предварительно определенные константы:

```
const
EmptyFill = 0;{Заполнение фоном (узор отсутствует)}
SolidFill = 1;{Сплошное заполнение}
LineFill = 2;{Заполнение -------}
LtSlashFill = 3;{Заполнение ///////}
SlashFill = 4;{Заполнение утолщенными ///}
BkSlashFill = 5;{Заполнение утолщенными \\\}
LtBkSlashFill = 6;{Заполнение \\\\\\\}
HatchFill = 7;{Заполнение +++++++}
XHatchFill = 8;{Заполнение xxxxxxx}
InterleaveFill= 9;{Заполнение прямоугольную клеточку}
WideDotFill = 10;{Заполнение редкими точками}
CloseDotFill = 11;{Заполнение частыми точками}
UserFill = 12;{Узор определяется пользователем}
```

Если параметр Fill имеет значение 12 (UserFill), то рисунок узора определяется программистом путем обращения к процедуре SetFillPattern.

Процедура SetFillPattern. Устанавливает образец рисунка и цвет штриховки. Заголовок:

```
Procedure SetFillPattern(Pattern: FillPatternType;Color: Word);
```

Здесь Pattern - выражение типа FillPatternType; устанавливает образец

рисунка для Fill - UserFill в процедуре SetFillStyle; Color - цвет заполнения.

Образец рисунка задается в виде матрицы из 8x8 пикселей и может быть представлен массивом из 8 байт следующего типа:

```
type
FillPatternType = array [1..8] of Byte;
```

Каждый разряд любого из этих байтов управляет светимостью пикселя, причем первый байт определяет 8 пикселей первой строки на экране, второй байт - 8 пикселей второй строки и т.д.

Процедура GetFillPattern. Возвращает образец заполнения, установленный ранее процедурой SetFillPattern. Заголовок:

```
Procedure GetFillPattern(var Pattern: FillPatternType);
```

Здесь Pattern - переменная типа FillPatternType, в которой возвращается образец заполнения.

Если программа не устанавливала образец с помощью процедуры SetFillPattern, массив Pattern заполняется байтами со значением 255 ($FF).

Процедура GetFillSettings. Возвращает текущий стиль заполнения. Заголовок:

```
Procedure GetFillSettings(var Pattlnfo: FillSettingsType);
```

Здесь Pattlnfo - переменная типа FillSettingsType, в которой возвращается текущий стиль заполнения,

В модуле Graph определен тип:

```
type
FillSettingsType = record
Pattern: Word; {Образец}
Color : Word {Цвет}
end;
```

Поля Pattern и Color в этой, записи имеют то же назначение, что и аналогичные параметры при обращении к процедуре SetFillStyle.

Процедура FloodFill. Заполняет произвольную замкнутую фигуру, используя текущий стиль заполнения (узор и цвет). Заголовок:

```
Procedure FloodFill(X,Y: Integer; Border: Word);
```

Здесь X, Y- координаты любой точки внутри замкнутой фигуры; Border - цвет граничной линии.

Если фигура незамкнута, заполнение «разольется» по всему экрану.

Следует учесть, что реализованный в процедуре алгоритм просмотра границ замкнутой фигуры не отличается совершенством. В частности, если выводятся подряд две пустые строки, заполнение прекращается. Такая ситуация обычно возникает при заполнении небольших фигур с использованием типа LtSlashFill.

Процедура Bar. Заполняет прямоугольную область экрана. Заголовок:

```
Procedure Bar(X1,Y1,X2,Y2: Integer);
```

Здесь XJ...Y2 - координаты левого верхнего (X1, Y1) и правого нижнего (X2, Y2) углов закрашиваемой области.

Процедура закрашивает (но не обводит) прямоугольник текущим образцом узора и текущим цветом, которые устанавливаются процедурой SetFillStyle.

Процедура Bar3D. Вычерчивает трехмерное изображение параллелепипеда и закрашивает его переднюю грань . Заголовок:

```
Procedure Bar3D (X1,Y1,X2,Y2,Depth: Integer; Top: Boolean);
```

Здесь X1... Y2 - координаты левого верхнего (X1, Y1) и правого нижнего (X2, Y2) углов передней грани; Depth - третье измерение трехмерного изображения («глубина») в пикселях; Top - способ изображения верхней грани.

Если параметр Top имеет значение True, верхняя грань параллелепипеда вычерчивается, в противном случае - не вычерчивается (этот вариант используется для изображения поставленных друг на друга параллелепипедов). В качестве значения этого параметра может использоваться одна из следующих констант, определенных в модуле Graph:

```
const
TopOn = True;
TopOff = False;
```

При вычерчивании используется текущий стиль линий (SetLineStyle) и текущий цвет (SetColor). Передняя грань заливается текущим стилем заполнения (SetFillStyle).

Процедура обычно применяется при построении столбиковых диаграмм. Следует учесть, что параллелепипед «прозрачен», т.е. за его незакрашенными гранями могут быть видны другие элементы изображения.

Процедура Fill Poly. Обводит линией и закрашивает замкнутый

многоугольник. Заголовок:

```
Procedure FillPoly(N: Word; var Coords);
```

Здесь N - количество вершин замкнутого многоугольника; Coords - переменная типа PointType, содержащая координаты вершин.

Координаты вершин задаются парой значений типа Integer: первое определяет горизонтальную, второе - вертикальную координаты. Для них можно использовать следующий определенный в модуле тип:

```
type
PointType = record
x, y : Integer
end;
```

Стиль и цвет линии контура задаются процедурами SetLineStyle и SetColor, тип и цвет заливки - процедурой SetFillStyle.

Процедура FillEllipse. Обводит линией и заполняет эллипс. Заголовок:

```
Procedure FillEllipse(X,Y,RX,RY: Integer);
```

Здесь X, Y - координаты центра; RX, RY- горизонтальный и вертикальный радиусы эллипса в пикселях.

Эллипс обводится линией, заданной процедурами SetLineStyle и SetColor, и заполняется с использованием параметров, установленных процедурой SetFillStyle.

Процедура Sector. Вычерчивает и заполняет эллипсный сектор. Заголовок: Procedure Sector(X,Y: Integer; BegA,EndA,RX,RY: Word);

Здесь BegA, EndA - соответственно начальный и конечный углы эллипсного сектора. Остальные параметры обращения аналогичны параметрам процедуры FillEllipse.

Процедура PieSlice. Вычерчивает и заполняет сектор окружности. Заголовок:

```
Procedure PieSlice(X,Y: Integer; BegA,EndA,R: Word);
```

В отличие от процедуры Sector, указывается лишь один горизонтальный радиус R, остальные параметры аналогичны параметрам процедуры Sector.

Сектор обводится линией, заданной процедурами SetLineStyle и SetColor, и заполняется с помощью параметров, определенных процедурой SetFillStyle. Процедуру удобно использовать при построении круговых диаграмм (рис. 31).

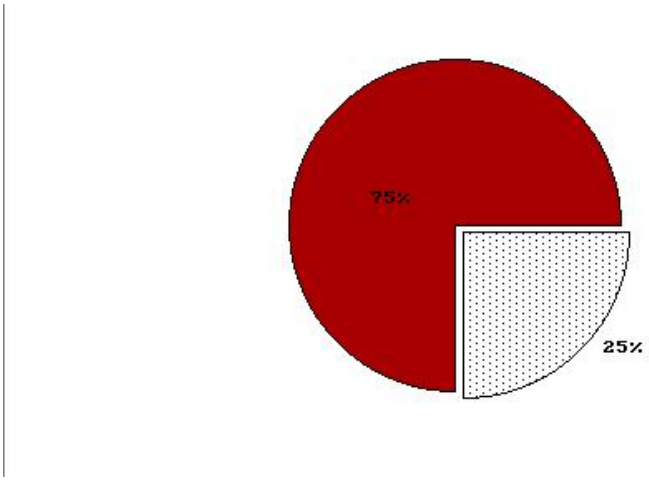


Рис. 31. Иллюстрация процедуры PieSlice

Вывод текста

Описываемые ниже стандартные процедуры и функции поддерживают вывод текстовых сообщений в графическом режиме. Это не одно и то же, что использование процедур Write или WriteLn. Дело в том, что специально для графического режима разработаны процедуры, обеспечивающие вывод сообщений различными шрифтами в горизонтальном или вертикальном направлении, с изменением размеров и т.д. Однако в стандартных шрифтах, разработанных для этих целей фирмой Borland, отсутствует кириллица, что исключает вывод русскоязычных сообщений.

С другой стороны, процедуры Write и WriteLn после загрузки в память второй половины таблицы знакогенератора (а эта операция легко реализуется в адаптерах EGA и VGA) способны выводить сообщения с использованием национального алфавита, но не обладают мощными возможностями специальных процедур.

Стандартные средства модуля Graph для вывода текста:

Процедура OutText. Выводит текстовую строку, начиная с текущего положения указателя. Заголовок:

```
Procedure OutText(Txt: String);
```

Здесь Txt - выводимая строка.

При горизонтальном направлении вывода указатель смещается в конец выведенного текста, при вертикальном - не меняет своего положения. Строка выводится в соответствии с установленным стилем и выравниванием. Если текст выходит за границы экрана, то при использовании штриховых шрифтов

он отсекается, а в случае стандартного шрифта не выводится.

Процедура OutTextXY. Выводит строку, начиная с заданного места. Заголовок:

```
Procedure OutTextXY (X,Y: Integer; Txt: String);
```

Здесь X, Y - координаты точки вывода; Txt - выводимая строка. Отличается от процедуры OutText только координатами вывода. Указатель не меняет своего положения.

Процедура SetTextStyle. Устанавливает стиль текстового вывода на графический экран. Заголовок:

```
Procedure SetTextStyle(Font,Direct,Size: Word);
```

Здесь Font - код (номер) шрифта; Direct - код направления; Size - код размера шрифта.

Для указания кода шрифта можно использовать следующие предварительно определенные константы:

```
const
DefaultFont = 0;{Точечный шрифт 8x8}
TriplexFont = 1;{Утроенный шрифт TRIP.CHR}
SmallFont = 2;{Уменьшенный шрифт LITT.CHR}
SansSerifFont = 3;{Прямой шрифт SANS.CHR}
GothicFont = 4;{Готический шрифт GOTH.CHR}
```

Эти константы определяют все шрифты для версий 4.0, 5.0, 5.5 и 6.0. В версии 7,0 набор шрифтов значительно расширен, однако для новых шрифтов не предусмотрены соответствующие мнемонические константы. В этой версии помимо перечисленных Вы можете при обращении к SetTextStyle использовать такие номера шрифтов:

Номер	Файл	Краткое описание
5	scri.chr	«Рукописный» шрифт
6	simp.chr	Одноштриховый шрифт типа Courier
7	tscr.chr	Красивый наклонный шрифт типа Times Italic
8	Icom.chr	Шрифт типа Times Roman
9	euro . chr	Шрифт типа Courier увеличенного размера
10	bold.chr	Крупный двухштриховый шрифт

Шрифт DefaultFont входит в модуль Graph и доступен в любой момент. Это -единственный матричный шрифт, т.е. его символы создаются из матриц 8x8 пикселей. Все остальные шрифты - векторные: их элементы формируются как совокупность векторов (штрихов), характеризующихся направлением и размером. Векторные шрифты отличаются более богатыми

изобразительными возможностями, но главная их особенность заключается в легкости изменения размеров без существенного ухудшения качества изображения. Каждый из этих шрифтов размещается в отдельном дисковом файле. Если Вы собираетесь использовать какой-либо векторный шрифт, соответствующий файл должен находиться в Вашем каталоге, в противном случае вызов этого шрифта игнорируется и подключается стандартный.

Для задания направления выдачи текста можно использовать константы:

```
const
HorizDir = 0;{Слева направо}
VertDir = 1;{Снизу вверх}
```

Как видим, стандартные процедуры OutText и OutTextXY способны выводить сообщения лишь в двух возможных направлениях - слева направо или снизу вверх. Зная структуру векторных шрифтов, нетрудно построить собственные процедуры вывода, способные выводить сообщения в любом направлении.

Каждый шрифт способен десятикратно изменять свои размеры. Размер выводимых символов кодируется параметром Size, который может иметь значение в диапазоне от 1 до 10 (точечный шрифт - в диапазоне от 1 до 32). Если значение параметра равно 0. устанавливается размер 1, если больше 10 - размер 10. Минимальный размер шрифта. при котором еще отчетливо различаются все его детали, равен 4 (для точечного шрифта - 1).

Процедура SetTextJustify. Задает выравнивание выводимого текста по отношению к текущему положению указателя или к заданным координатам. Заголовок:

```
Procedure SetTextJustify(Horiz,Vert: Word);
```

Здесь Horiz - горизонтальное выравнивание; Vert - вертикальное выравнивание. Выравнивание определяет как будет размещаться текст - левее или правее указанного места, выше, ниже или по центру. Здесь можно использовать такие константы:

```
const
LeftText = 0;{Указатель слева от текста}
CenterText= 1;{Симметрично слева и справа,верху и снизу}
RightText = 2;{Указатель справа от текста}
BottomText= 0;{Указатель снизу от текста}
TopText = 2;{Указатель сверху от текста}
```

Процедура SetUserCharSize. Изменяет размер выводимых символов в соответствии с заданными пропорциями. Заголовок:

```
Procedure SetUserCharSize(XI,X2,Yl,Y2: Word);
```

Здесь X1...Y2 - выражения типа Word, определяющие пропорции по горизонтали и вертикали.

Процедура применяется только по отношению к векторным шрифтам. Пропорции задают масштабный коэффициент, показывающий во сколько раз увеличится ширина и высота выводимых символов по отношению к стандартно заданным значениям. Коэффициент по горизонтали находится как отношение X1 к X2, по вертикали - как отношение Y1 к Y2. Чтобы, например, удвоить ширину символов, необходимо задать X1=2 и X2=1. Стандартный размер символов устанавливается процедурой SetTextStyle, которая отменяет предшествующее ей обращение к SetUserCharSize.

Функция TextWidth. Возвращает длину в пикселях выводимой текстовой строки. Заголовок:

```
Function TextWidth (Txjt: String): Word;
```

Учитываются текущий стиль вывода и коэффициенты изменения размеров символов, заданные соответственно процедурами SetTextStyle и SetUserCharSize.

Функция TextHeight. Возвращает высоту шрифта в пикселях. Заголовок:

```
Function TextHeight(Txt: String): Word;
```

Процедура GetTextSettings. Возвращает текущий стиль и выравнивание текста. Заголовок:

```
Procedure GetTextSettins(var Textlnfo: TextSettingsType);
```

Здесь Textlnfo - переменная типа TextSettingsType, который в модуле Graph определен следующим образом:

```
type
TextSettingsType = record
Font : Word; {Номер шрифта}
Direction: Word; {Направление}
CharSize : Word; {Код размера}
Horiz : Word; {Горизонтальное выравнивание}
Vert : Word; {Вертикальное выравнивание}
end;
```

Функция InstallUserFont. Позволяет программе использовать нестандартный векторный шрифт. Заголовок функции:

```
Function InstallUserFont(FileName: String): Integer;
```

Здесь FileName - имя файла, содержащего векторный шрифт.

Как уже говорилось, в стандартную поставку Турбо Паскаля версий 4.0 - 6.0 включены три векторных шрифта, для версии 7.0 - 10. Функция InstallUserFont позволяет расширить этот набор. Функция возвращает идентификационный номер нестандартного шрифта, который может использоваться при обращении к процедуре SetTextStyle.

Функция InstallUserDriver. Включает нестандартный графический драйвер в систему BGI-драйверов. Заголовок функции:

```
Function InstallUserDriver(FileName: String; AutoDetectPtr: Pointer):
Integer;
```

Здесь FileName - имя файла, содержащего программу драйвера; AutoDetectPtr - адрес точки входа в специальную процедуру автоопределения типа дисплея, которая в числе прочих процедур должна входить в состав драйвера.

Эта функция расширяет и без того достаточно обширный набор стандартных графических драйверов и предназначена в основном для разработчиков аппаратных средств.

ЗАДАНИЯ ДЛЯ ПРОВЕДЕНИЯ ПРАКТИЧЕСКИХ ЗАНЯТИЙ

Задачи на использование одномерных массивов

1. Даны натуральное число n, массив A[n]. Вычислить:

а) $a_1 + \ldots + a_n$; б) $a_1 \times \ldots \times a_n$; в) $\sin |a_1 + \ldots + a_n|$; г) $|a_1| \times |a_2| \times \ldots \times |a_n|$;
д) $a_1 + \ldots + a_n$ и $a_1 \times a_2 \times \ldots \times a_n$; е) a_1, a_1+a_2, …, $a_1+a_2+ \ldots + a_n$;
ж) $a_1 \times a_1$, $a_1 \times a_2$, $a_1 \times a_3$, …, $a_1 \times a_n$; з) $|a_1|$, $|a_1 + a_2|$, $|a_1 + \ldots + a_n|$;
и) $-a_1$, a_2, $-a_3$, …, $(-1)^n \times a_n$.

2. Дано натуральное число n. Получить последовательность b_1, …, b_n, где при i = 1, 2, …, n значение b_i равно:

а) i ; б) i^2 ; в) i! ; г) 2^{i+1} ; д) $2^i + 3^{i+1}$; е) $(-1)^i \times 3^i$; ж) 1+1/2+…1/i; з) 2^i /i!.

3. Вычислить значения многочлена $x^3 - 9x^2 + 3{,}2x - 7{,}5$ для x = 0, 1, …, 5.

4. Даны натуральное число n, действительные числа a, b (a¹b). Получить r_1, r_2, …, r_n, где $r_i = a + ih$, $h = (b - a) / n$.

5. Получить таблицу температур по Цельсию от 0 до 100 градусов и их эквивалентов по шкале Фаренгейта, используя для перевода формулу

$$t_F = \frac{9}{5} t_C + 32 .$$

6. Вычислить значения функции $y = 5x^3 - 3x^2 + 5$ для значений x, изменяющихся -3 … 1, с шагом 0,1.

7. Даны натуральные числа i, n, действительные числа a_1, …,a_n (i £ n). Найти среднее арифметическое всех чисел a_1, …, a_n, кроме a_i.

8. Даны действительные числа a_1, a_2, … Известно, что $a_1>0$ и что среди a_2, a_3,… есть хотя бы одно отрицательное число. Пусть a_1, …, a_n – члены данной последовательности, предшествующие первому отрицательному члену (n заранее неизвестно). Получить:

а) $a_1 + a_2 + \ldots + a_n$; б) среднее арифметическое a_1,…, a_n;
в) a_1, a_1a_2, $a_1a_2a_3$, …, $a_1a_2 \ldots a_n$; г) $a_1a_2 + a_2a_3 + \ldots + a_{n-1}a_n + a_na_1$;
д) $(-1)^n a_n$; е) $n + a_n$;
ж) $|a_1 - a_n|$.

9. Даны натуральное число n, действительные числа a_1, …, a_n. Получить числа b_1, …, b_n, которые связаны с a_1, …, a_n следующим образом:

$b_1 = a_1$, $b_n = a_n$, $b_i = (a_{i+1} - a_i)/3$, i=2, …, n-1.

10. Даны натуральные числа x, y_1, …, y_{100}. ($y_1 < y_2 \ldots < y_{100}$, $y_1 < x < y_{100}$). Найти натуральное k, при котором $y_{k-1} < x$ £ y_k.

11. Даны натуральные числа n, a_1,…, a_n. Определить количество членов a_k последовательности a_1, …, a:

а) являющихся нечетными числами;
б) кратных 3 и не кратных 5;
в) являющихся квадратами четных чисел;
г) имеющих четные порядковые номера и являющихся нечетными числами.

12. Даны целые числа a_1, …, a_{50}. Получить сумму тех чисел данной последовательности, которые а) кратны; б) нечетны и отрицательны.

13. Даны натуральное число n, целые числа a_1, …, a_n. Найти количество и сумму тех членов данной последовательности, которые делятся на 5 и не делятся на 7.

14. Даны натуральные числа n, p, целые числа a_1, …, a_n. Получить произведение членов последовательности a_1, …, a_n, кратных p.

15. Даны натуральное число n, действительные числа a_1, …, a_n. Получить удвоенную сумму всех положительных членов последовательности a_1, …, a_n.

16. Даны натуральное число n, действительные числа x_1, …, x_n. В последовательности x_1, …, x_n все члены, меньшие двух, заменить нулями. Кроме того, получить сумму членов, принадлежащих отрезку [3,7], а также число таких членов.

17. Даны натуральное число n, целые числа a_1, …, a_n. Получить сумму положительных и число отрицательных членов последовательности a_1, …, a_n.

18. Даны натуральное число n, целые числа a_1, …, a_n. Заменить все большие семи члены последовательности a_1, …, a_n числом 7. Вычислить количество таких членов.

19. Даны целые числа a_1, …, a_{45}. Получить число отрицательных членов последовательности a_1, …, a_{35} и число нулевых членов всей последовательности a_1, …, a_{45}.

20. Даны натуральное число n, целые a, x_1, …, x_n. Если в последовательности x_1, …, x_n есть хотя бы один член, равный a, то получить сумму всех членов, следующих за первым таким членом; в противном случае ответом должно быть число -10.

21. Даны натуральное число n, действительные числа a_1, …, a_n.

Получить:

а) $\max(a_1, \ldots, a_n)$; б) $\min(a_1, \ldots, a_n)$; в) $\max(a_2, a_4, \ldots)$; г) $\min(a_1, a_3, \ldots)$; д) $\min(a_2, a_4, \ldots) + \max(|a_1|, |a_3|, \ldots)$; е) $\max(|a_1|, \ldots, |a_n|)$.

22. Даны натуральное число n, действительные числа a_1, …, a_n.

а) Верно ли, что отрицательных членов в последовательности a_1, …, a_n больше, чем положительных?
б) Верно ли, что наибольший член последовательности a_1, …, a_n по модулю больше единицы?

23. Даны натуральное число n, действительные числа a_1, …, a_n. В последовательности a_1, …, a_n определить число соседств:

а) двух положительных чисел;
б) двух чисел разного знака;
в) двух чисел одного знака, причем модуль первого числа должен быть больше модуля второго числа.

24. Даны целые числа c_1, …, c_{80}. Имеются ли в последовательности c_1, …, c_{80}:

а) два идущих подряд нулевых члена;
б) три идущих подряд нулевых члена?

25. Даны целые числа a_1, a_2, … Известно, что $a_1 > 0$ и что среди $a_2, a_3, \ldots$ есть хотя бы одно отрицательное число. Пусть $a_1, \ldots, a_n$ – члены данной последовательности, предшествующие первому отрицательному члену (n заранее неизвестно). Получить:

а) $\max(a_1, \ldots, a_n)$; б) $\min(a_1, 2a_2, \ldots, na_n)$; в) $\min(a_1+a_2, a_2+a_3, \ldots, a_{n-1}+a_n)$; г) количество четных среди a_1, …, a_n; д) $\max(a_1, a_1+a_2, \ldots, a_1+a_2+\ldots+a_n)$;

26. Даны натуральное число n, действительные числа a_1, …, a_n. Выяснить, является ли последовательность a_1, …, a_n упорядоченной по убыванию.

Задачи на построение процедур и функций

1. Даны действительные числа s, t. Получить f(t,-3s,3.22) + f(5.2,t,s-t),

где f (a,b,c) = $\frac{2a - b - \sin c}{5 + |c|}$

2. Даны действительные числа s, t. Получить g(1.2,s) + g(t,s) - g(2s-1,st),

где g(a,b) = $\frac{a^2 + b^2}{a^2 + 2ab + 3b^2 + 4}$

3. Даны действительные числа a, h, натуральное число n. Вычислить

$$f(a) + 2f(a + h) + \ldots + 2f(a + (n-1)h) + f(a + nh),$$

где $f(x) = (x^2 + 1) \cos^2 x$.

4. Даны действительные числа a, b, c. Получить

$$\frac{\max(a, a + b) + \max(a, b + c)}{1 + \max(a + bc, 6.23)}$$

5. Даны действительные числа a, b. Получить

$$u = \min(a, b),\ v = \min(ab, a + b),\ x = \min(u^2 + v, 3.14)$$

6. Даны действительные числа s, t. Получить

$$h(s, t) + \max(h^2(s-t, st), h^3(s-t, s+t)) + h(1,1),$$

где

h (a,b) = $\frac{a}{1 + b^2} + \frac{b}{1 + a^2} - (a - b)^2$.

7. Даны действительные числа x_1, y_1, x_2, y_2, …, x_{10}, y_{10}. Найти периметр десятиугольника, вершины которого имеют соответственно координаты (x_1, y_1), (x_2, y_2), …, (x_{10}, y_{10}). (Определить функцию вычисления расстояния между двумя точками, заданными своими координатами).

8. Даны натуральное число n, действительные числа x_1, y_1, x_2, y_2, …, x_n, y_n. Найти площадь n-угольника, вершины которого при некотором последовательном обходе имеют координаты (x_1, y_1), (x_2, y_2),…, (x_n, y_n). (Определить функцию вычисления площади треугольника по координатам его вершин.)

9. Составить функцию, позволяющую определить позицию самого правого вхождения заданного символа в исходную строку. Если строка не содержит символа, результатом работы процедуры должна быть -1.

10. Составить процедуру, изменяющую в исходной строке символов все единицы нулями и все нули единицами. Замена должна выполняться, начиная с заданной позиции строки.

ПРАКТИЧЕСКИЕ ЗАДАЧИ НА ПРОГРАММИРОВАНИЕ

Задачи на программирование алгоритмов линейной структуры

1. Даны два числа a и b. Получить их сумму, разность и произведение.

2. Даны действительные числа x и y. Получить (|x| – |y|) / (1 + |x·y|).

3. Дана длина ребра куба. Найти площадь грани, площадь полной поверхности и объем этого куба.

4. Даны два действительных положительных числа. Найти среднее арифметическое и среднее геометрическое этих чисел.

5. Даны катеты прямоугольного треугольника. Найти его гипотенузу и площадь.

6. Определить периметр правильного n-угольника, описанного около окружности радиуса r.

7. Даны x, y, z. Вычислить a, b, если

а) $a = \frac{\sqrt{|x-1|} - \sqrt{|y|}}{1 + x^2/2 + x^2/4}, b = x(arctg(z) + e^{-(x+3)})$;

б) $a = \frac{3 + e^{y-1}}{1 + x^2|y - tg(z)|}, b = 1 + |y - x| + \frac{(y-x)^2}{2} + \frac{|y-x|^3}{3}$;

в) $a = (1+y)\frac{x + y\sqrt{(x^2+4)}}{e^{-x-2} + 1/(x^2+4)}, b = \frac{1 + \cos(y-2)}{x^4/2 + \sin^2 z}$;

г) $a = y + \frac{x}{y^2 + \left|\frac{x^2}{y + x^3/3}\right|}, b = \left(1 + tg^2\frac{z}{2}\right)$;

д) $a = \frac{2\cos(x - \pi/6)}{1/2 + \sin^2 y}, b = 1 + \frac{z^2}{3 + z^2/5}$.

е) $a = \dfrac{1+\sin^2(x+y)}{2+\left|x-2x/(1+x^2y^2)\right|}+x,\ b=\cos^2\left(arctg\dfrac{1}{z}\right)$;

ж) $a=\ln\left|\left(y-\sqrt{|x|}\right)\left(x-\dfrac{y}{z+x^2/4}\right)\right|,\ b=x-\dfrac{x^2}{3!}+\dfrac{x^3}{4!}$.

8. Дана сторона равностороннего треугольника. Найти площадь этого треугольника.

9. Известна длина окружности. Найти площадь круга, ограниченного этой окружностью.

10. Найти площадь кольца, внутренний радиус которого равен 20, а внешний – заданному числу r (r>20).

11. Найти площадь равнобочной трапеции с основаниями a и b и углом α при большем основании a.

12. Вычислить расстояние между двумя точками с координатами x_1, y_1 и x_2, y_2.

13. Треугольник задан координатами своих вершин. Найти:

а) периметр треугольника;
б) площадь треугольника.

14. Найти площадь сектора, радиус которого равен 3,7, а дуга содержит заданное число радиан φ.

15. Дано действительное число a. Не пользуясь никакими другими арифметическими операциями, кроме умножения, получить:

а) a^4 за две операции;
б) a^6 за три операции;
в) a^7 за четыре операции;
г) a^8 за три операции;
д) a^9 за четыре операции;
е) a^{10} за четыре операции;
ж) a^{13} за пять операций;
з) a^{15} за пять операций (указание:$a^{15}=(a^5)^3$);
и) a^{21} за шесть операций;
к) a^{28} за шесть операций;
л) a^{64} за шесть операций.

16. Дано действительное число a. Не пользуясь никакими другими арифметическими операциями, кроме умножения, получить:

а) a^3 и a^{10} за четыре операции;
б) a^4 и a^{20} за пять операций;
в) a^5 и a^{13} за пять операций;
г) a^5 и a^{19} за шесть операций;
д) a^2 и a^5, a^{17} за шесть операций;
е) a^4 ,a^{12}, a^{28} за шесть операций.

Задачи на программирование алгоритмов ветвящейся структуры

1. Даны действительные числа x, y. Получить:

а) max(x,y);
б) min(x,y);
в) max(x,y), min(x,y).

2. Даны действительные числа x, y, z. Получить:

а) max(x,y,z);
б) min(x,y,z);
в) max(x,y,z), min(x,y,z).

3. Даны действительные числа x, y, z. Вычислить:

а) max2(x+y+z,xyz);
б) min(x+y+z/2,xyz)+1;
в) min(x+5y+z,2x-y+z) + max(x+y+z,xy+z).

4. Даны действительные числа a, b, c. Удвоить эти числа, если a > b > c, и заменить их абсолютными значениями, если это не так.

5. Даны действительные числа x, y. Вычислить z:

$$y = \begin{cases} x - y, \text{если } x > y, \\ y - x + 1, \text{в противном случае.} \end{cases}$$

6. Даны два действительных числа. Вывести первое число, если оно больше второго, и оба числа, если это не так.

7. Даны два действительных числа. Заменить первое число нулем, если оно меньше или равно второму, и оставить без изменения в противном случае.

8. Даны действительные числа x, y (x ¹ y). Меньшее из этих двух чисел заменить их полусуммой, а большее – их удвоенным произведением.

9. Даны действительные числа x, y, z. Выяснить, существует ли треугольник с длинами сторон x, y, z.

10. Даны действительные числа a, b, c (a ¹ 0). Выяснить, имеет ли уравнение $ax^2 + bx + c = 0$ действительные корни. Если действительные корни имеются, то найти их. В противном случае ответом должно служить сообщение, что действительных корней нет.

11. Дано действительное число a. Вычислить f(a), если

а) $$f(x) = \begin{cases} x^2, \text{при } -2 \le x < 2, \\ 4, \text{в противном случае}; \end{cases}$$

б) $$f(x) = \begin{cases} x^2 + 4x + 5, \text{при } x \le 2, \\ \dfrac{1}{x^2 + 4x + 5}, \text{в противном случае}; \end{cases}$$

в) $$f(x) = \begin{cases} 0, \text{при } x \le 0, \\ x, \text{при } 0 < x \le 1, \\ x^2, \text{в противном случае}, \end{cases}$$

в) $$f(x) = \begin{cases} 0, \text{при } x \le -5, \\ x^2 - x, \text{при } -5 < x \le 5, \\ x^2 - \sin x^2, \text{в противном случае}. \end{cases}$$

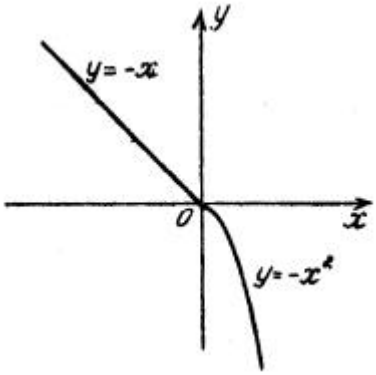
12. Дано действительное число a. Для функций f(x), графики которых представлены на рис. 32, вычислить f(a).

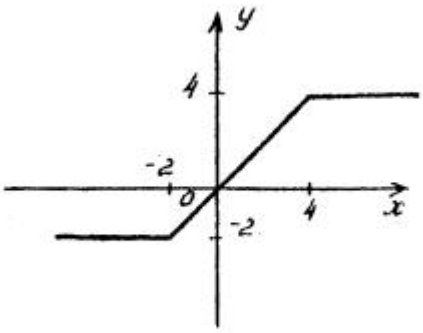
13. Даны действительные числа x, y. Определить, принадлежит ли точка с координатами x, y заштрихованной части плоскости (рис. 32).

14. Дано натуральное число n (n < 100), определяющее возраст человека (в годах). Дать для этого числа наименования "год", "года" или "лет": например, год, 23 года,5 лет и т.д.

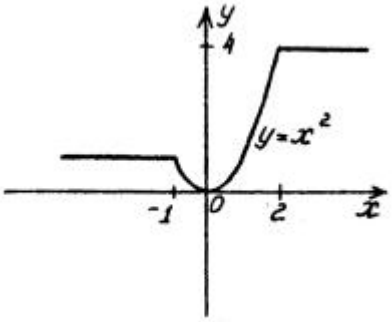
15. Дата вводится в формате "дд.мм.гг". Записать словесно название месяца и полностью указать год. Например, для 10.02.96 вывести 10 февраля 1998 года.

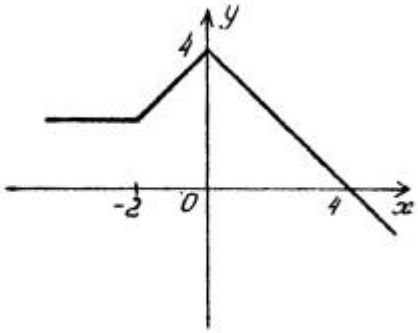
16. Дано натуральное число n. Определить является ли число четным.


y
y= -x
0
x
y= -x²

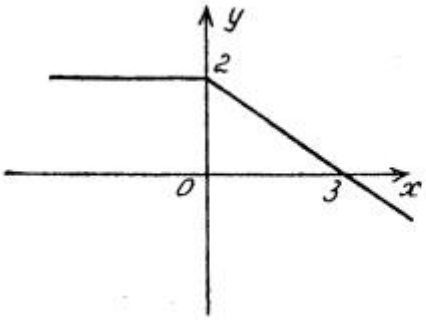

y
4
-2
0
4
x
-2

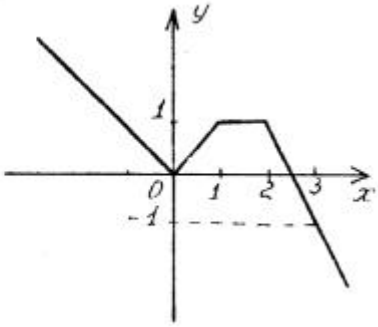
а б


y
4
y=x²
-1
0
2
x

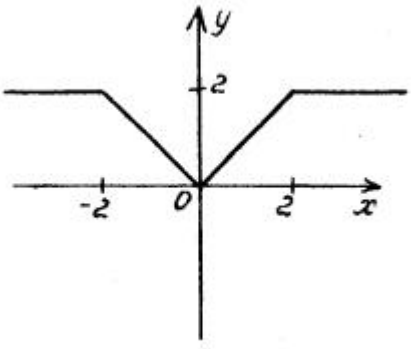

y
4
-2
0
4
x

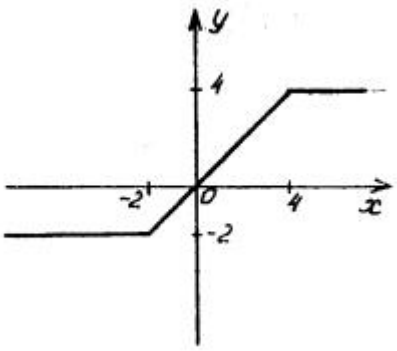
в г


y
2
0
3
x

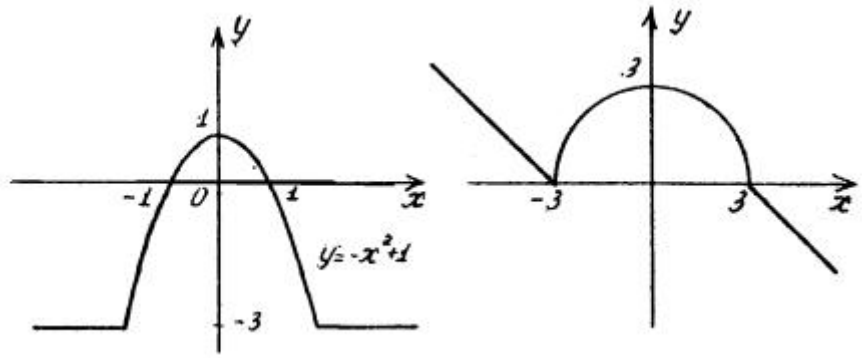

y
1
0
1
2
3
x
-1

д е

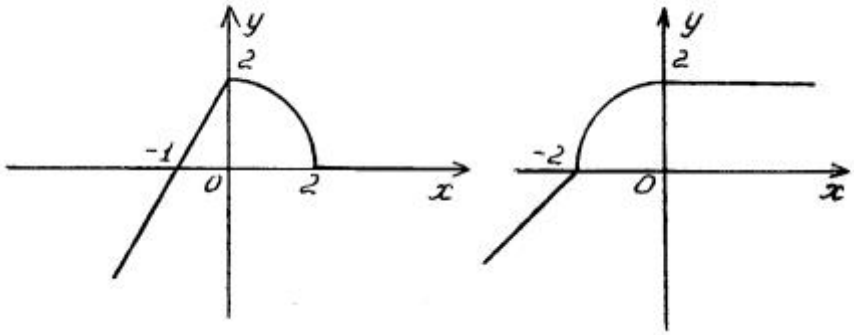

y
2
-2
0
2
x


y
4
-2
0
4
x
-2

ж з

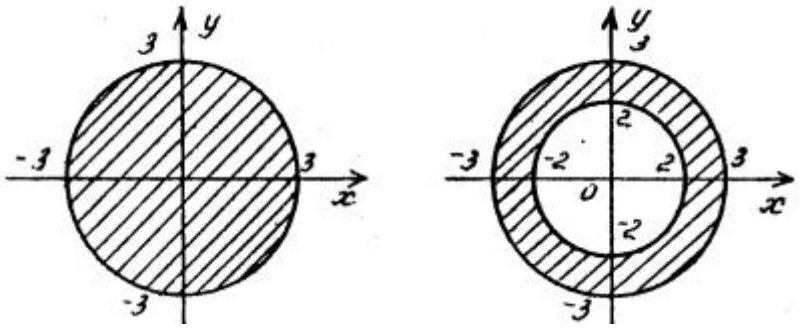


и к

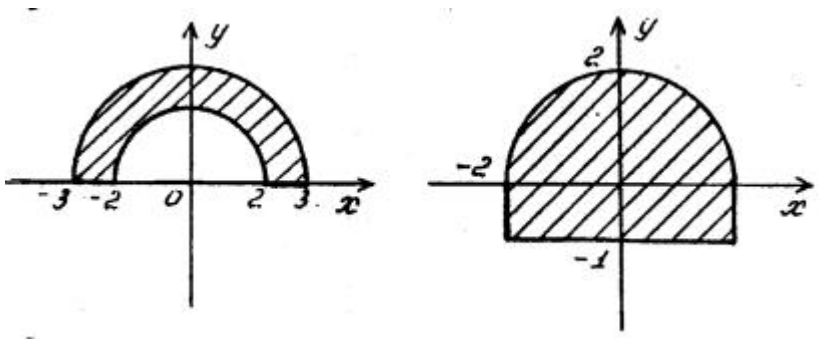


л м

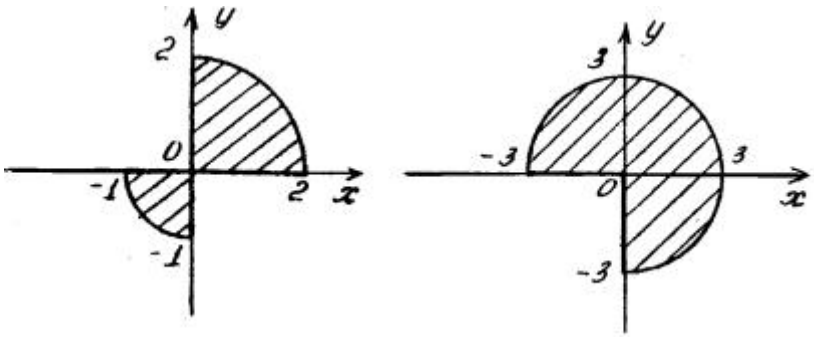
Рис. 32



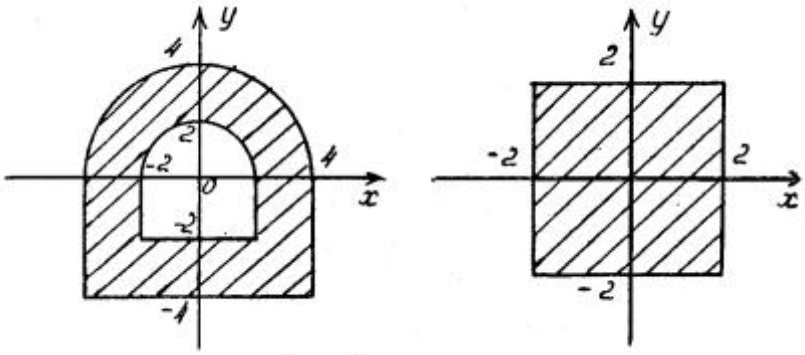
а б



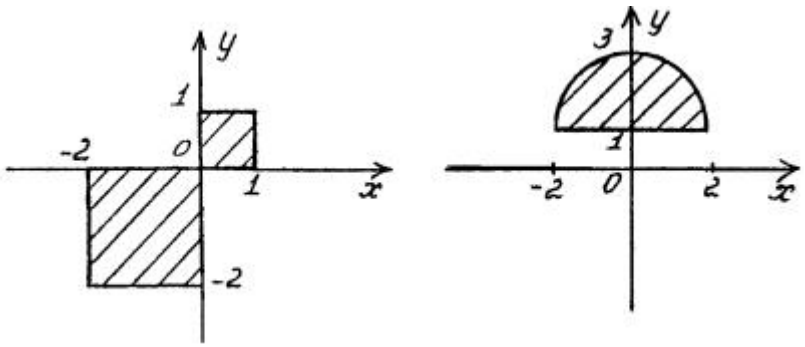
в г



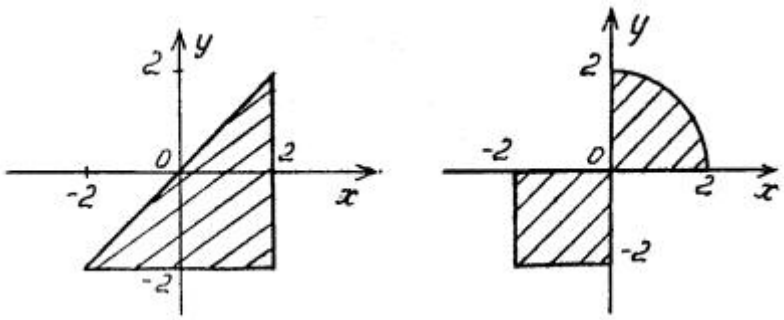
д е



ж з



и к



л м

Рис. 33

17. Сравнить между собой значение величины x и y. Вывести на печать результат сравнения в виде "x > y", "x = y" или "x < y".

18. Определить, какая из двух точек T1(x1,y1) или T2(x2,y2) расположена ближе к началу координат. Вывести на печать координаты этой точки.

19. Определить, какая из двух фигур - круг или квадрат имеет большую площадь. Известно, что сторона квадрата равна a, радиус круга r. Вывести на печать название и значение площади большей фигуры.

Задачи на работу с файлами

1. Дан файл f, компоненты которого являются действительными числами. Найти:

а) сумму компонент файла f;
б) произведение компонент файла f;
в) сумму квадратов компонент файла f;
г) модуль суммы и квадрат произведения компонент файла f;
д) последнюю компоненту файла.

2. Дан файл f, компоненты которого являются действительными числами. Найти:

а) наибольшее из значений компонент;
б) наименьшее из значений компонент с четными номерами;
в) сумму наибольшего и наименьшего из значений компонент;
г) разность первой и последней компонент файла.

3. Дано натуральное n. Записать в файл g целые числа b_1, …, b_n, где при i = 1, 2, …, n значение b_i равно:

а) i ; б) i^2 ; в) i! ; г) 2^{i+1} ; д) $2^i + 3^{i+1}$; е) $(-1)^i \times 3^i$;
ж) $1+\frac{1}{2}+\ldots+\frac{1}{i}$; з) $\frac{2^i}{i!}$;

4. Даны символьные файлы f_1и f_2. Переписать с сохранением порядка следования компоненты файла f1 в файл f2, а компоненты файла f_2 - в файл f_1. Использовать вспомогательный файл h.

5. Дан файл f, компоненты которого являются целыми числами. Получить в файле g все компоненты файла f:

а) являющиеся четными числами;
б) делящиеся на 3 и не делящиеся на 7.

6. Дан файл f, компоненты которого являются целыми числами. Записать в файл g все четные числа файла f, а в файл h - все нечетные. Порядок следования чисел сохраняется.

7. Дан символьный файл f. Записать в файл g компоненты файла f в обратном порядке.

8. Даны символьные файлы f и g. Записать в файл h сначала компоненты файла f, а затем - компоненты файла g с сохранением порядка.

Задачи на вычисление площади криволинейной трапеции

Вычислить площадь криволинейной трапеции 1) f(x)=x*sin(x) [0 ; 5]
Вычислить площадь криволинейной трапеции 2) f(x)=sin(x)+cos(x) [0 ; 2.25]
Вычислить площадь криволинейной трапеции 3) f(x)=1/x+x^2 [0.1 ; 5.17]
Вычислить площадь криволинейной трапеции 4) f(x)=sin(x)cos(x)+1 [0 ; 5]
Вычислить площадь криволинейной трапеции 5) $f(x)=(e^{x}+e^{-x})/2$ [-1 ; 4]
Вычислить площадь криволинейной трапеции 6) $f(x)=x^{5}$ [-1 ; 2]
Вычислить площадь криволинейной трапеции 7) f(x)=tan(x) [-1 ; 1.5]
Вычислить площадь криволинейной трапеции 8) $f(x)=x^{2}+2x+2$ [-5 ; 5]
Вычислить площадь криволинейной трапеции

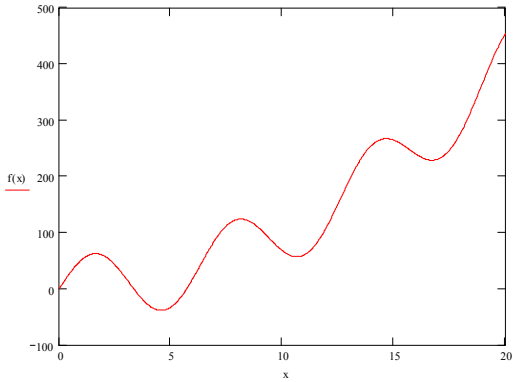
10) f(x)=sin(cos(x)) [-1.55 ; 1.55]
Вычислить площадь криволинейной трапеции 11) $f(x)=e^{1/x}$ [-2.54 ; 0]
Вычислить площадь криволинейной трапеции 9) $f(x)=\sin(x^2)$ [0 ; 1.75]
Вычислить площадь криволинейной трапеции 13) $f(x)=\sqrt[4]{x}$ [1 ; 8]
Вычислить площадь криволинейной трапеции 14) $f(x)=\sqrt[4]{x}+x^2$
Вычислить площадь криволинейной трапеции 12) $f(x)=\frac{\sqrt[4]{x}}{x}$ [1 ; 10.165]
Вычислить площадь криволинейной трапеции 1) f(x)=x*sin(x) [0 ; 5]
Вычислить площадь криволинейной трапеции 2) f(x)=sin(x)+cos(x) [0 ; 2.25]
Вычислить площадь криволинейной трапеции 3) f(x)=1/x+x^2 [0.1 ; 5.17]
Вычислить площадь криволинейной трапеции 4) f(x)=sin(x)cos(x)+1 [0 ; 5]
Вычислить площадь криволинейной трапеции 5) $f(x)=(e^x+e^{-x})/2$ [-1 ; 4]
Вычислить площадь криволинейной трапеции 6) $f(x)=x^5$ [-1 ; 2]

Вычислить площадь криволинейной трапеции 8) $f(x)=x^2+2x+2$ [-5 ; 5]
Вычислить площадь криволинейной трапеции 10) $f(x)=\sin(\cos(x))$ [-1.55 ; 1.55]
Вычислить площадь криволинейной трапеции 11) $f(x)=e^{1/x}$ [-2.54 ; 0]
Вычислить площадь криволинейной трапеции 9) $f(x)=\sin(x^2)$ [0 ; 1.75]

Примеры вариантов самостоятельной работы

Вариант 1

Найти все минимальные и максимальные значения функции (пики). Найти максимальное и минимальное значение по модулю, на отрезке [-100, 100]



Вариант 2

Найти максимальное и минимальное значение функции с помощью производной

$F(x) = x^2-2x+1$ [-10, 10]

Вариант 3

Найти максимальное и минимальное значение функции с помощью производной

$F(x) = x^3 + 10x^2 + x + 2$ [-10, 10]

Вариант 4

Найти область значений функции на отрезке [a, b]

Вариант 5

Перемножить матрицу на матрицу размерности 3×3 или n×n

Вариант 6

Перемножить матрицу на вектор столбец

Вариант 7

Решить придел вида $\lim_{x\to 0}\frac{P_n(x)}{Q_n(x)}$, и $\lim_{x\to \infty}\frac{P_n(x)}{Q_n(x)}$. $P_n(x)$, $Q_n(x)$ многочлены.

Вариант 8

В двумерном массиве вывести каждую диагональ.

Вариант 9

Найти площадь фигуры ограниченной двумя линиями
$f1(x) = x^2$, $f2(x) = 2x$, на отрезке [0, 2]

Вариант 10

Найти в матрице максимальное значение элемента, поменять строчку и столбец в которых найден максимальный элемент с первой строкой и столбцом соответственно т.е. элемент переместится в ячейку a_{11}.

ТЕМЫ РЕФЕРАТОВ ПО ДИСЦИПЛИНЕ

1) История Windows?
2) Что такое Internet? Классификация доменов. Почта, поисковые системы.
3) Сетевые технологии, TCP/IP, IP, MAC – адрес?
4) Языки программирования их классификация и назначение?
5) Развитие компьютерного оборудования с 80гг? Предполагаемые направления развития оборудования.
6) Базы данных. Сетевые, иерархические, реляционные.
7) История информатики. История Windows?
8) Программы для ОС Windows, их назначение, классификация.
9) Ассемблер основы программирования.
10) Нейронные сети, экспертные системы, системы реального времени.
11) Структура Windows? Компоненты, реестр.
12) BIOS, начальная загрузка Windows 98, Me, XP.
13) Организация хранение данных в ПК, серверах. Особенности RAID, SCSI, IDE и др.
14) Компьютерные вирусы, механизмы заражения.
15) Антивирусы, сравнение, принципы защиты.
16) Сетевое оборудование

ПРИМЕР ТЕСТОВЫХ ЗАДАНИЙ

Тест по основным понятиям информации и информатике

---

1. Отметьте те понятия, которые связаны с понятием "информатика".

- ☐ **Сигнал**
- ☐ **Вещество**
- ☐ **Сообщение**
- ☐ **Данные**
- ☐ **Энергия**

---

2. Что из ниже перечисленного является информационным процессом?

- ☐ **Сбор информации**
- ☐ **Обработка информации**
- ☐ **Получение информации**
- ☐ **Хранение информации**
- ☐ **Обмен информацией**

---

3. Архитектура ЭВМ – это:

- ○ **совокупность общих принципов организации аппаратно-программных средств и их характеристик**
- ○ **конкретный состав вычислительного средства на некотором уровне детализации**
- ○ **описание связей внутри вычислительного средства во всей их полноте**

---

4. Какие основные устройства содержит ЭВМ неймановской структуры?

- ☐ **арифметико-логическое устройство**
- ☐ **устройство управления**
- ☐ **устройства ввода-вывода**
- ☐ **запоминающее устройство**
- ☐ **устройство контроля**

5. Операционная система – это:

- комплекс программ, управляющих всеми процессами внутри компьютера
- программа для обработки текста
- программа-оболочка
- сервисная программа

6. Какая программа не относится к типовому прикладному программному обеспечению?

- текстовый процессор
- экспертная система
- система управления базами данных
- программа архивации данных
- графический процессор
- программа математического расчета

7. Что такое интерфейс?

- программа для распознавания текста
- совокупность средств и правил для взаимодействия устройств ПК, программ и пользователя
- программа-переводчик
- рабочий стол операционной системы Windows

8. Чему равен 1 Гбайт?

- 1024 байта
- 562 байта
- 1024 Кбайт
- 1024 Мбайт

9. Что такое файл?

- именованное последовательность байтов произвольной длины
- магнитный носитель

- **название программы**
- **название ОС**

---

10. Какая допустимая максимальная длина полного имени файла в операционной системе MS-DOS?

- **256 символов**
- **512 символов**
- **128 символов**
- **64 символа**

---

11. Какой из знаков недопустим в имени файла?

- **?**
- **%**
- **(**
- **№**

---

12. Сколько знаков может иметь расширение файла в операционной системе MS-DOS?

- **3**
- **4**
- **5**
- **6**

---

13. Какое расширение имеют исполняемые файлы программ?

- **.BAT**
- **.SYS**
- **.EXE**
- **.DOC**

---

14. Отметьте элементы файловой структуры.

- **стартовый сектор**
- **конечный сектор**

- ☐ **таблица размещения файлов**
- ☐ **корневой каталог**
- ☐ **область данных**

---

15. Что такое BIOS?

- ○ **операционная система**
- ○ **встроенная программа для загрузки операционной системы и автотестирования**
- ○ **интерпретатор команд**
- ○ **сервисная программа**

ВОПРОСЫ ДЛЯ ПОДГОТОВКИ К ЭКЗАМЕНУ

1. Информатика. Предмет, цели и задачи информатики. Информатика как наука. Основные направления в информатике.
2. Понятие информации, виды и свойства информации. Единицы измерения информации.
3. Этапы обработки информации в ЭВМ. Общая характеристика сбора, хранения, обработки, накопления и передачи информации.
4. Способы представления и преобразования информации. Системы счисления.
5. ЭВМ как универсальное устройство обработки информации, классификации ЭВМ. Тенденции развития ЭВМ.
6. Структура и функции аппаратной части ПК. Микропроцессоры, запоминающие устройства, основные внешние устройства ПК.
7. Программное обеспечение ЭВМ, его состав и назначение.
8. Операционные системы, их назначение. Понятие каталога, файла, директории.
9. Операционная система MS-DOS.
10. Оболочка операционной системы. Norton-подобные оболочки.
11. Программы-драйверы. Утилиты, NU.
12. Основные понятия и основы работы с WINDOWS.
13. Интернет. Службы Интернет. Адресация в Интернет.
14. Текстовые редакторы, назначение и классификация, общие сведения о создании и редактировании текстов. WORD, функциональные возможности.
15. Табличные процессоры (EXCEL), принципы их построения и обработки.
16. Адресация и форматирование ячеек. Работа с формулами. Построение диаграмм.
17. Компьютерная графика. Графические редакторы. Понятие графической информации, способы ее представления и обработки в ЭВМ.
18. Понятие базы данных, их назначение, классификации и использование.
19. Модели баз данных.
20. Системы управления базами данных. Принципы работы с базами данных (на примере Access).
21. Языки программирования, назначение и классификация.
22. Алгоритмы, способы задания и описания. Основные конструкции алгоритмов.
23. Константы: целые, вещественные, логические, символьные. Понятие идентификатор.
24. Переменные, их описание в программе. Скалярные типы. Тип integer. Операции и функции, применимые для данного типа.
25. Переменные, их описание в программе. Скалярные типы. Тип real. Операции и функции, применимые для данного типа.

26. Переменные, их описание в программе. Скалярные типы. Тип boolean. Логические операции AND,OR,NOT.
27. Переменные, их описание в программе. Скалярные типы. Тип char. Операции и функции, применимые для данного типа.
28. Переменные, их описание в программе. Скалярные типы. Перечисляемые типы. Операции и функции, применимые для данного типа.
29. Структура программы на языке Паскаль. Комментарии.
30. Процедура ввода READ, READLN, вывода WRITE, WRITELN.
31. Оператор присваивания. Приоритет операций. Составной оператор.
32. Оператор условного перехода IF. Оператор безусловного перехода GOTO.
33. Оператор CASE. Операции сравнения.
34. Операторы цикла WHILE, REPEAT, FOR.
35. Функции SUCC(x), PRED(x), ORD(x), ODD(x), ROUND(x), TRUNC(x).
36. Массивы.
37. Множества.
38. Подпрограммы общего вида (PROCEDURE). Подпрограммы-функции (FUNCTION).
39. Рекурсия. Рекурсивные программы.
40. Записи.
41. Записи с вариантами.
42. Понятие о файлах. Процедуры открытия и закрытия файлов.
43. Типизированные файлы. Процедуры и функции работы с файлами: GET, PUT, EOF, READ, WRITE.
44. Текстовые файлы.
45. Нетипизированные файлы.
46. Работа с графикой в среде Turbo Pascal. Видеорежимы. Процедуры и функции, предназначенные для управления графическими режимами.
47. Загрузка драйверов и шрифтов в программу.
48. Процедуры и функции, предназначенные для построения графических образов.
49. Процедуры и функции, предназначенные для работы с цветовой палитрой.
50. Процедуры и функции, предназначенные для вывода текстовой информации.

ПРИМЕР ЭКЗАМЕНАЦИОННОГО БИЛЕТА

АМУРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Утверждено на заседании кафедры
«_____»_____200__г.
Заведующий кафедрой
Утверждаю:_____

Кафедра МАиМ
Факультет МиИ
Курс 1
Дисциплина:
Информатика

ЭКЗАМЕНАЦИОННЫЙ БИЛЕТ № 4

1. Переменные, их описание в программе. Скалярные типы. Тип real. Операции и функции, применимые для данного типа.
2. Оператор CASE. Операции сравнения.
3. Составить программу поиска максимального из четырех чисел с использованием функции или процедуры поиска большего из двух.

УЧЕБНО-МЕТОДИЧЕСКАЯ ЛИТЕРАТУРА ПО ДИСЦИПЛИНЕ

Перечень обязательной (основной) литературы

1. Гусева А.И. Учимся информатике. Задачи и методы их решения. - М: Диалог-Мифи, 2001.

2. Зуев Е.А. Программирование на языке Turbo Pascal 6.0, 7.0.-М.: Радио и связь, Веста, 2003.-380с.

3. Емелина Е.И. Основы программирования на языке Паскаль.-М.: Финан¬сы и статистика, 1999.- 208с.

4. Епанешников А.М. Программирование в среде Turbo Pascal 7.О.- М.: Диалог-Мифи, 2000.-288с.

5. Королев Л. Н., Миков А. И. Информатика. Введение в компьютерные науки.-М.:Высшая школа, 2003.

6. Могилев А. В., Пак Н. И., Хеннер Е. К. Информатика.-М.: Академия, Серия: Высшее образование ("Академия"), 2004. Е. К.

7. Сафронов И. К. Задачник-практикум по информатике.-СПб.:ВНУ-СПб, 2002.

8. Семакин И.Г., Хеннер Е.К. Информатика. Задачник-практикум.-М.: Ла¬боратория Базовых Знаний, Том 2, 2002.

9. Стариченко Б.Е. Теоретические основы информатики. Учебное пособие для вузов.-М: Горячая Линия - Телеком, Серия: Специальность для высших учебных заведений, 2003.

Перечень дополнительной литературы

1. Абрамов С.А., Зима В.С. Начало программирования на языке Паскаль.-

М.: Наука, 1987.- 112с.

2. Вирт Н. Алгоритмы + структура данных = программы : перевод с англ.-М.Мир,1985.-406с.

3. Левин А. Самоучитель работы на компьютере. - М. Международное агенство A.D.&T.,1997 - 480с.

4. Перминов О.Н. Язык программирования Паскаль. - М.: Радио и связь, 1983.

5. Петров А.В. и др. Вычислительная техника в инженерных и экономических расчетах.-М.: Высшая школа, 1990. - 478с.

6. Фигурнов В. IBM PC для пользователя.-С.Петербург, 1997г.

7. Франклен Г. MS DOS 5.0 для пользователя.- Киев: Торгово-издательское бюро BHV, 1992.

Перечень методических пособий

1. Гордиенко А.М., Рогов Д.К. Численные методы проектирования на ЭВМ./ Методические указания, М., 1990. - 31с.

2. Ракитин В.И., Первушин В.Е. Практическое руководство по методам

вычислений. - М.: Выс.шк., 1998. - 383с.

НЕОБХОДИМОЕ ТЕХНИЧЕСКОЕ И ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

Лекции и практические занятия проводятся в стандартной аудитории, оснащенной в соответствии с требованиями преподавания теоретических дисциплин.

Для проведения лабораторных работ необходимо:

1. Компьютеры класса Pentium (10 посадочных мест), сетевое окружение.
2. Интернет-центр.
3. Системное программное обеспечение.
4. Среда программирования Турбо Паскаль.
5. Прикладное программное обеспечение.

УЧЕБНО-МЕТОДИЧЕКАЯ (ТЕХНОЛОГИЧЕСКАЯ) КАРТА ДИСЦИПЛИНЫ

Номер недели	Номер темы	Вопросы, изучаемые на лекциях	Занятия (номера)		Используемые наглядные и методические пособия	Самостоятельная работа студентов		Форма контроля
			Практ - тич. (семин.)	Лабораторные		Содержание	часы	
1	]		1	1	лекция, доп. литература		4	отчет по лабор. работе № 1
2	2			2	лекция, доп. литература	индивид, задание № 1	4	отчет по лабор. работе №2, отчет по индвид. зад. №l
	2		1	л	лекция, доп. литература		4	отчет по лабор. работе №3
4	2-3			<*)	лекция		4	отчет по лабор. работе №3, контр. | работа
5	3		2	4-5	лекция	индивид, задание №2	4	отчет по лабор. работе №4-5, отчет по индвид. зад. №2
6	4			6	лекция		4	отчет по лабор. работе №6
7	5		3	7	лекция		4	отчет по лабор. работе №7, контрольная работа
8	6			8	лекция		4	отчет по лабор. работе №8
9	6		4	9	лекция	индивид, задание №3	4	отчет по лабор. работе №9, отчет по индвид. зад. №3
10	6			10	лекция		4	отчет по лабор. работе №10
М	6		5	11	лекция		4	отчет по лабор. работе №11
12	6			12	лекция		4	отчет по лабор. работе № 12
13	6		6-7	13	лекция		4	отчет по лабор. работе № 13
14	6			14	лекция		4	отчет по лабор. работе №14, контрольная работа
15	6		8	15	лекция		4	отчет по лабор. работе № 15
16	6-7			16	лекция		4	отчет по лабор. работе №16
17	7		9	17	лекция	индивид, задание №4	4	отчет по лабор. работе №17, отчет по индвид. зад. №4
18	7			18	лекция		4	отчет по лабор. работе №18, устный и письменный опрос, экзамен

В настоящее время курс лекций, практических занятий и лабораторных работ, а также контроль самостоятельной работы студентов проводит доктор технических наук, профессор, Д.Л. Харичева.