Министерство образования Республики Беларусь
Учреждение образования
«Витебский государственный технологический университет»

Е.Ю. Вардомацкая

ОСНОВЫ ИНФОРМАТИКИ И ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

Курс лекций

Витебск
Издательство УО «ВГТУ»
2005

УДК 004 (07)
ББК 32.81
В -18

Печатается по решению научно-методического совета учреждения образования «Витебский государственный технологический университет»

Автор: Вардомацкая Е.Ю.

Рецензенты: кафедра информатики УО «ВГТУ»;
кандидат технических наук доцент В.Л. Шарстнев

Вардомацкая Е.Ю.

В-18 ОСНОВЫ ИНФОРМАТИКИ И ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ: курс лекций / Е. Ю. Вардомацкая. – Витебск : Издательство УО «ВГТУ», 2005. – 131 с.

ISBN 985-481-024-0

Настоящее издание представляет собой конспективный курс лекций по языку программирования Turbo Pascal. Оно предназначено для студентов, изучающих дисциплину «Основы информатики и вычислительной техники», и, в первую очередь, для тех, кому по каким-либо причинам предстоит самостоятельно усвоить те или иные разделы курса или весь курс целиком. Предлагаемые лекции содержат основной материал по всем разделам дисциплины в соответствии с действующей учебной программой и с учетом особенностей изучения предмета на экономических специальностях.

УДК 004 (07)
ББК 32.81

ISBN 985-481-024-0

ЛЕКЦИЯ 1. ВВЕДЕНИЕ В ПРЕДМЕТНУЮ ОБЛАСТЬ

Информатика – область человеческой деятельности, связанная с процессами преобразования информации с помощью компьютеров и их взаимодействием со средой применения.

Термин *«информатика»* возник в 60-х годах во Франции для названия области, занимающейся автоматизированной обработкой информации с помощью ЭВМ. Французский термин «информатика» образован путём слияния слов "information" (информация) и "automatigus" (автоматика) и означает «информационная автоматика или автоматизированная обработка информации». Выделение информатики как самостоятельной области человеческой деятельности в первую очередь связано с развитием компьютерной техники. Причем основная заслуга в этом процессе принадлежит микропроцессорной технике, появление которой в 70-х годах послужило началом второй электронной революции. Термин «информатика» теперь связывается не только с отображением достижений компьютерной техники, но и с процессами передачи и обработки информации. Информатика занимается изучением процессов преобразования и создания новой информации более широко, практически не решая задачу управления различными объектами, причем акцент ставится на свойствах информации и аппаратно-программных средствах её обработки.

Информация – это сведения об объектах и явлениях окружающей среды, их параметрах, свойствах и состоянии, которые уменьшают имеющуюся о них степень неопределённости, неполноты знания. Информатика рассматривает информацию как концептуально связанные между собой сведения, данные, понятия, изменяющие наши понятия о явлении или объекте окружающего мира.

Наряду с информацией, в информатике часто упоминается понятие «данные». Данные могут рассматриваться как признаки или записанные наблюдения, которые по каким-то причинам не используются, а только хранятся. В том случае, если появляется возможность использовать эти данные для уменьшения неопределённости о чём-либо, данные превращаются в информацию. Поэтому можно утверждать, что информацией являются используемые данные.

Одной из важнейших разновидностей информации является *экономическая информация*. Её отличительная черта – связь с различными процессами управления. Экономическая информация сопровождает процессы производства, распределения, обмена и потребления материальных благ и услуг.

Экономическая информация – совокупность сведений, отражающих социально-экономические процессы и служащих для управления этими процессами и коллективами людей в производственной и непроизводственной сфере. Информатика играет важную роль в

современной экономической науке, что привело к выделению отдельного направления развития науки – экономической информатики. Это новое направление объединяет в себе экономику, математику и информатику и помогает экономистам решать задачи оптимизации деятельности предприятий, принимать стратегически важные решения о развитии промышленности и управлять производственным процессом. Разработанная программная база основывается на математических моделях экономических процессов и предоставляет гибкий и надежный механизм предсказания экономического эффекта управленческих решений. С помочью ЭВМ быстро решаются аналитические задачи, решение которых не под силу человеку.

В последнее время компьютер стал неотъемлемой частью рабочего места управленца и экономиста.

Классификация информации по разным признакам

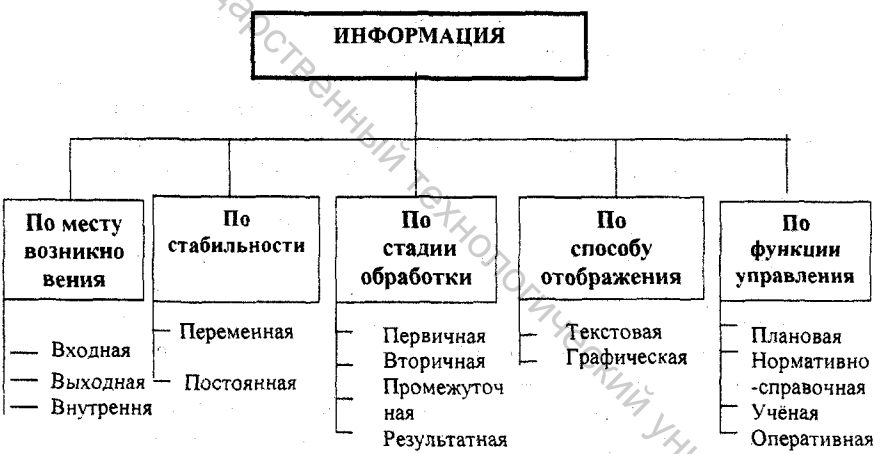


Рис. 1. Классификация информации

Место возникновения

Входная информация – информация, поступающая в фирму или её подразделения.

Выходная информация – информация, поступающая из фирмы в другую фирму, организацию, подорганизацию.

Внутренняя информация возникает внутри объекта, внешняя – за пределами объекта.

Стабильность

Переменная (текущая) информация отражает фактические количественные и качественные характеристики производственно-хозяйственной деятельности.

Постоянная (условно-постоянная) информация – неизменная и многократно используемая в течение длительного периода времени информация. Она может быть:

- справочная (описание постоянных свойств объекта в виде устойчивых длительное время признаков);
- нормативная (местные, отраслевые и общегосударственные нормативы);
- плановая (многократно используемые в фирме плановые показатели).

Стадия обработки

Первичная информация – информация, которая возникает непосредственно в процессе деятельности объекта и регистрируется на начальной стадии.

Вторичная информация – информация, получаемая в результате обработки первичной. Она может быть промежуточной и результатной.

Промежуточная информация используется как исходные данные для последующих расчётов.

Результатная информация получается в процессе обработки первичной и промежуточной информации и используется для выработки управленческих решений.

Способ отображения

Текстовая информация – совокупность алфавитных, цифровых и специальных символов, с помощью которых представляется информация на физических носителях (бумага, дисплей…).

Графическая информация – различного рода графики, диаграммы, схемы, рисунки и т. д.

Функция управления

По функциям обычно классифицируют экономическую информацию.

Плановая информация – информация о параметрах объекта управления на будущий период.

Нормативно-справочная информация содержит различные нормативные и справочные данные. Её обновление происходит достаточно редко.

Учётная информация – информация, характеризующая деятельность фирмы за определённый прошлый период времени.

Оперативная (текущая) информация – информация, характеризующая производственные процессы в текущий (данный) период времени и используемая в оперативном управлении.

Создание и использование информационной системы для любой организации нацелены на решение следующих задач.

- Структура информационной системы, её функциональное назначение должны соответствовать целям, поставленным перед организацией.

- Информационная система должна контролироваться людьми, ими пониматься и использоваться в соответствии с основными социальными и этическими принципами.
- Производство достоверной, надёжной, своевременной и систематизированной информации.

Единицы измерения информации

Какую бы информацию компьютер не воспринимал, ни обрабатывал, ни выводил, он всегда превращает ее в так называемую бинарную или двоичную информацию. Она представляется всего двумя уровнями некоего сигнала – условным наличием его или отсутствием. Их называют логическая единица и нуль (1 и 0), ДА и НЕТ, True и False и т. д. Единица двоичной информации получила название бит. 8 бит = 1 байту – единице информации, имеющей 2^8=256 значений от 0 до 255.

256 значений байта хватает на кодирование приличного числа знаков – например всех знаков русского или английского языков. Каждая последующая единица измерения объема информации, например килобайт, мегабайт и т. д., получается умножением предшевствующей единицы не в 1000 раз, а в 1024 раза.

ЛЕКЦИЯ 2. АППАРАТНЫЕ СРЕДСТВА И ОСНОВЫ УПРАВЛЕНИЯ ПЕРСОНАЛЬНЫМИ КОМПЬЮТЕРАМИ

Состав и архитектура ПЭВМ

Персональная ЭВМ (ПЭВМ) – это микроЭВМ, предназначенная для индивидуального пользования. В минимальной конфигурации ПЭВМ состоит из следующих основных компонент: системного блока, клавиатуры и дисплея. Системный блок содержит следующие устройства (см. рис.2):

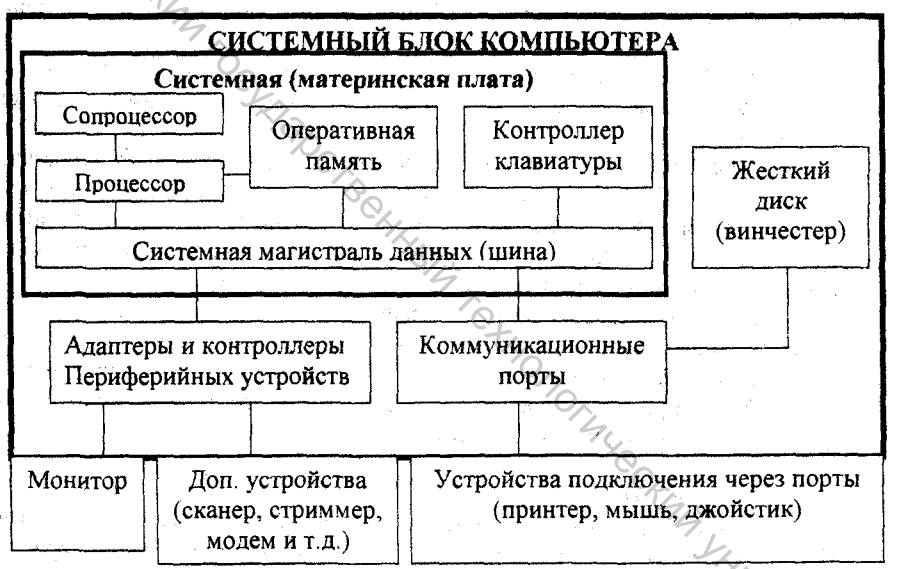


Рис.2. Архитектура ПК

Центральный процессор (системный блок), выполняющий вычисления в соответствии с заданной программой и управляющий работой компьютера. ЦП содержит:

основной процессор – для выполнения логических и математических операций;

математический сопроцессор – для ускорения вычислений с плавающей точкой;

оперативную память (ОЗУ), в которой располагаются программы, выполняемые компьютером, и используемые программой данные. Емкость ОЗУ может составлять десятки мегабайтов (16,32,64 Мб);

постоянную память (ПЗУ), которая содержит базовую систему ввода-вывода (BIOS), управляющие программы и команды;

электронные схемы (контроллеры и адаптеры), управляющие работой различных устройств, входящих в компьютер;

порты ввода-вывода, через которые производится обмен данными с внешними устройствами (PORT – вход-выход). Имеются специализированные порты, через которые происходит обмен данными с внутренними устройствами компьютера, и порты общего назначения, к которым могут подсоединяться различные дополнительные внешние устройства (принтер, мышь, сетевой адаптер и т.п.). Порты общего назначения бывают двух видов: *параллельные* (обозначаемые LPT1-LPT4) и *асинхронные последовательные* (обозначаемые COM1-COM3). *Параллельные* порты выполняют ввод и вывод с большей скоростью, чем последовательные, но требуют и большего числа проводов для обмена данными;

блок дисков, состоящий из накопителей (или дисководов) для гибких магнитных дисков (дискет) и из накопителя на жестком магнитном диске (винчестера);

дисплей (монитор), предназначенный для отображения информации, и в зависимости от адаптера, которым он комплектуется, может быть монохромным цифровым, монохромным графическим или цветным графическим. Наибольшее распространение получили следующие типы цветных графических мониторов: GGA, EGA, VGA, SVGA.

Передача информации между устройствами компьютера, находящимися в системном блоке, осуществляется по *системной шине*.

Таким образом, архитектура ПК, которая изображена на рисунке 2, дает общее представление о входящих в его состав устройствах и функциональных взаимосвязях между ними.

Классификация ПК

Главным фактором, определяющим вычислительные возможности ПЭВМ, является тип установленного в компьютер микропроцессора. В зависимости от использованного микропроцессора различают следующие классы ПК:

- компьютеры класса XT, созданные на базе микропроцессора Intel 8080/86 (1978г.);
- компьютеры класса AT, созданные на базе микропроцессора Intel 80286 (1982г.);
- компьютеры класса 386, созданные на базе микропроцессора Intel 80386 (1985г.);
- компьютеры класса 486, созданные на базе микропроцессора Intel 80486 (1989г.);
- компьютеры класса Pentium, снабженные микропроцессором Pentium фирмы Intel (1993г.);
- компьютеры класса PentiumII, снабженные микропроцессором Pentium фирмы Intel (1995г.);

- компьютеры класса PentiumIII, снабженные микропроцессором Pentium фирмы Intel (1999г.);
- компьютеры класса PentiumIV, снабженные микропроцессором Pentium фирмы Intel (2002г.).

Несмотря на значительные различия, ПК старшего семейства способны выполнять все программы, разработанные для младшего семейства.

В качестве перспективных можно рассматривать ПК класса Pentium, представители других классов морально устарели, и их выпуск прекращен.

В соответствии с целевым назначением компьютеры подразделяют на бытовые, учебные и профессиональные.

Бытовые – это наиболее простые, маломощные компьютеры, служащие для индивидуального применения в бытовых условиях (Celeron).

К учебным относятся компьютеры среднего класса, предназначенные для обучения различным предметам в учебных заведениях (ПК класса 486, Pentium, Celeron на базе процессоров Intel, Cyrix).

Профессиональными являются мощные ПК, обеспечивающие удовлетворение интересов профессиональных пользователей (ПК класса PentiumII и выше).

Внешние запоминающие устройства (ВЗУ)

ВЗУ обеспечивают долговременное хранение программ и данных на различных носителях информации. Наибольшее распространение получили накопители на магнитных и оптических дисках. Любая ПЭВМ содержит как минимум один накопитель на гибких магнитных дисках и накопитель на жестких магнитных дисках.

Накопители на гибких магнитных дисках (НГМД) являются устройством со сменным носителем информации. Они обеспечивают резервирование информации, секретность данных и распространение программного обеспечения.

НГМД состоит из следующих узлов:

- *механического привода*, обеспечивающего вращение диска;
- *блока магнитных головок* чтения/записи;
- *системы позиционирования магнитных головок*, которая служит для их перемещения относительно диска в радиальном направлении;
- *электронного блока*, обеспечивающего управление накопителем.

Информация на магнитном диске размещается вдоль концентрических окружностей, называемых дорожками. Каждая дорожка содержит определенное число секторов. По номеру сектора и дорожки определяется местонахождение информации на диске. Наибольшее распространение получили двухсторонние, высокой плотности дискеты, диаметром 3,5", емкостью 1,44 Мбайт.

Накопители на жестких магнитных дисках (НЖМД) с несъёмным носителем информации – магнитным диском (винчестером), системой позиционирования и блоком магнитных головок помещены в герметично закрытый корпус. Сам диск имеет металлическую основу. В НЖМД устанавливается несколько дисков суммарной емкостью в среднем до 512 Гбайт и даже выше. В отличие от гибкого, жесткий диск вращается непрерывно при включенном питании компьютера, имеет повышенную долговечность в связи с тем, что в нем отсутствует непосредственный контакт магнитных головок с поверхностью диска.

Накопители на оптических дисках (НОД) применяются все чаще в связи с широким распространением мультимедиа-приложений, интегрирующих тексты, звук и видео. Они способны считывать записанную на компакт-диск информацию (CD-ROM). Принцип работы оптического диска основан на использовании луча лазера для записи/считывания информации в цифровом виде. В отличие от магнитного диска, оптический диск имеет одну спиральную дорожку и вращается в накопителе так, чтобы обеспечить постоянную линейную скорость, т.е. угловая скорость вращения диска увеличивается в процессе перемещения головки чтения к его центру. Емкость компакт-диска составляет от 680 Мбайт и выше.

Устройства ввода информации

Клавиатура – является основным устройством ввода информации в ПЭВМ. Она представляет собой матрицу клавиш, объединенных в единое целое, и электронный блок для преобразования нажатия клавиш в двоичный код. По своему назначению клавиши разбиваются на группы: алфавитно-цифровые, служебные, функциональные, клавиши малой цифровой клавиатуры и клавиши управления курсором.

Манипуляторы «мышь» – являются дополнительным периферийным устройством ввода. Действия «мышью» выполняются с использованием левой кнопки.

Устройства вывода информации

Дисплей (монитор) – это основное устройство, служащее для отображения выводимой на экран информации. Независимо от физических принципов формирования изображения дисплей состоит из двух основных частей – экрана и электронного блока, размещенных в одном корпусе. По функциональным возможностям дисплеи подразделяются на *алфавитно-цифровые* и *графические*. *Алфавитно-цифровые* способны воспроизводить только ограниченный набор символов. *Графические* в состоянии отображать как графическую, так и символьную информацию. По количеству воспроизводимых цветов различают *монохромные* и *цветные* дисплеи. По физическим принципам формирования изображения распространение получили дисплеи на *базе электронно-лучевой* трубки и

жидкокристаллические дисплеи. Возможности компьютера по отображению информации определяются совокупностью и совместимостью технических характеристик дисплея и его адаптера. Всякий адаптер содержит видеопамять, хранящую воспроизводимую на экране информацию. Каждой точке экрана соответствует поле видеопамяти, в котором хранится элемент изображения, определяющий режим высвечивания и цвет точки или символа. В настоящее время повсеместно используются видеоадаптеры типа Super VGA 32-х и 64-х битные с разрешением 1024x768 точек и выше: 1152x864, 1280x1024.

Печатающие устройства, или принтеры, предназначены для вывода алфавитно-цифровой или графической информации на бумагу. Наибольшее распространение получили матричные, лазерные и струйные принтеры. Основным узлом *матричного принтера* является печатающая головка, содержащая тонкие металлические иглы. Наиболее распространенные устройства имеют головку с 9 и 24 иглами и способны работать в двух режимах – текстовом и графическом. Скорость работы матричных принтеров достаточно низкая. В основе работы *лазерных принтеров* лежит электрографический принцип печати, заимствованный из ксерографии. С помощью лазерного луча изображение, подлежащее печати проецируется на вращающийся барабан, покрытый светочувствительным материалом. После этого на барабан наносится порошок-тонер, частицы которого прилипают к барабану только в тех местах, которые были обработаны лазерным лучом. Путем прижатия бумаги к барабану на нее переносится и фиксируется рисунок. Изображение формируется по точкам, но за счет высокого разрешения лазерными принтерами обеспечивается качество, близкое к типографскому. *В струйных принтерах* формируется непрерывный поток из маленьких капель чернил, которые заряжаются и, пролетая через электрическое поле, отклоняются в вертикальной плоскости пропорционально их заряду. В цветных струйных принтерах печатающая головка имеет три сопла, которые формируют потоки капель красных, зеленых и синих чернил. Смешивая капли, можно получить на бумаге точку любого цвета.

Дополнительные устройства ввода-вывода

Плоттер – позволяет выдать на бумагу проектно-конструкторскую документацию, технические чертежи или любую графическую информацию.

Дигитайзер – обеспечивает считывание (скалывание) информации с листа вручную.

Стример – используют для копирования информации с винчестера на ленточную кассету (типа магнитофонной).

Сканер – позволяет с помощью оптики считывать любую информацию с листа в компьютер.

ЛЕКЦИЯ 3. ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ПК. ОПЕРАЦИОННЫЕ СИСТЕМЫ

Программное обеспечение **ПЭВМ** можно разделить на три основные части:

- операционная система (ОС) и сервисные программы;
- инструментальные языки и системы программирования;
- прикладные системы.

Понятия, основные функции и составные части ОС

Операционная система (ОС) - это часть программного обеспечения ЭВМ, которая предназначена для управления процессами обработки пользовательских программ с момента их поступления в систему до выдачи результатов, а также распределения ресурсов вычислительной системы между отдельными программами и пользователями. Являясь неотъемлемой частью современной ЭВМ, ОС обеспечивают поддержку работы программы с аппаратурой и предоставление пользователю возможностей общего управления ПК. В настоящее время существует большое количество ОС для ПЭВМ, отражающих как этапы развития технических средств, так и стремление разработчиков улучшить функциональные и эксплуатационные характеристики, повысить степень комфортности ОС по отношению к пользователю. По мере развития ОС выделились их основные функции:

- организация передачи информации между различными внутренними устройствами (например, между оперативной памятью и дисководом);
- обеспечение выполнения прикладных программ пользователя и системных программ;
- поддержка работы периферийных устройств (ввода/вывода);
- распределение ресурсов между задачами и поддержка взаимораспределения затрат при их параллельной обработке.

Перечисленные функции реализуются ядром ОС. Окружением ядра являются системные утилиты, редакторы, компиляторы и другие программные средства, составляющие обслуживающую часть ОС. Ядро обеспечивает базовые функции для окружающего программного обеспечения и допускает расширение обслуживающей части ОС. Например поддержку СУБД, графический интерфейс, средства сетевой обработки и т.д. Ядро ОС постоянно находится в памяти. Остальное программное обеспечение располагается на внешних запоминающих устройствах.

Важнейшими характеристиками, определяющими выбор ОС, являются следующие:

- распространенность;
- наличие большого количества прикладных программных средств, работающих под ее управлением;
- простота освоения и взаимодействия с ней пользователя;

- легкость перехода с одной версии ОС к другой, более совершенной.

Основными препятствиями к использованию прикладного программного обеспечения в различных ОС являются:

- разные способы организации файлов,
- несовместимость форматов дисков (различное число секторов и различная емкость).

В различных системах ПЭВМ используются системы с разной архитектурой и возможностями. Для их работы необходимы различные ресурсы ОП. Они предоставляют разные степени сервиса как для программ, так и для работы пользователей.

Операционная система CPM

Является восьмиразрядной ОС и предоставляет пользователю только самый необходимый набор средств для управления ресурсами ЭВМ, доступа к файловой системе и организации диалога. В рамках этой ОС было создано программное обеспечение значительного объема, включающее трансляторы таких языков программирования, как BASIC, PASCAL, C, FORTRAN, КОБОЛ, ЛИСП, АДА, текстовые и табличные процессоры, СУБД, графические пакеты и др., как общего, так и проблемно-ориентированного назначения.

Операционная система MS DOS (Microsoft Disk Operation System)

Другой класс образуют ОС с более развитыми средствами доступа ко всем аппаратным компонентам, гибкой файловой системой, основанной на иерархической системе каталогов, удобным для пользователя командным языком. Средства, предоставляемые ОС этого класса, позволяют, с одной стороны, формировать удобную среду для разработки программного обеспечения (ПО), с другой стороны, на их основе можно разрабатывать автоматизированные рабочие места (АРМ) с простыми средствами доступа пользователей, научное и прикладное программное обеспечение.

MS DOS - это 16-разрядная однопользовательская однопроцессорная ОС, принятая в качестве базовой для работы на компьютерах IBM PC. Программы DOS работают и выполняются в пределах одного мегабайта адресного пространства компьютера, а основная память может использоваться для хранения данных.

К основным достоинствам **MS DOS** относятся:

- развитый командный язык;
- возможность организации многоуровневых каталогов;
- возможность работы со всеми пользовательскими устройствами, как с файлами;
- возможность подключения пользователем дополнительных внешних устройств.

Для MS DOS разработан большой арсенал программных средств. Имеются трансляторы практически для всех популярных алгоритмических

языков высокого уровня, таких, как BASIC, PASCAL, C, FORTRAN, КОБОЛ, ЛИСП, ПРОЛОГ и др. Причем для большинства языков существует несколько вариантов трансляторов. Имеются инструментальные средства разработки программ в машинных кодах: ассемблеры, отладчики и др. Они включают в себя текстовые редакторы, компоновщики, редакторы связей и другие сервисные средства, необходимые для создания сложных программ.

Первая версия MS DOS появилась в 1981 году, последняя - в 1994 году.

Структура MS DOS

Первая версия MS DOS разработана фирмой MS в 1981 г. для компьютеров IBM. Важнейшей особенностью MS DOS является ее модульность, что позволяет при необходимости расширения функций системы модифицировать ее модули. В состав ОС входят следующие модули:

1. *Базовая система ввода-вывода (BIOS)*, находящаяся в ПЗУ компьютера. BIOS содержит драйверы стандартных периферийных устройств, тестовые программы для контроля работоспособности оборудования и программу вызова системного загрузчика ОС.
2. *Системный загрузчик* размещается в первом секторе нулевой дорожки системного диска. Его основная функция заключается в считывании с диска в оперативную память модуля расширения BIOS и базового модуля.
3. *Модуль расширения BIOS* (файл IO.SYS) организует интерфейс с BIOS и подключает внешние драйверы периферийных устройств. Указания на подключение внешних драйверов должны содержаться в файле конфигурации CONFIG.SYS. Базовый модуль (файл MSDOS.SYS) реализует основные функции по управлению всеми ресурсами ПЭВМ и выполняемыми программами.
4. *Командный процессор* или интерпретатор команд (файл COMMAND.COM). Основные его функции заключаются в приеме, анализе, выполнении команд пользователя и обработке командных файлов (файлы типа BAT).
5. *Утилиты MS-DOS* - это программы, поставляемые вместе с ОС в виде отдельных файлов с расширением COM или EXE. Они выполняют действие обслуживающего характера, например форматирование дискет, проверку дискет и т.п.
6. *Внешние, устанавливаемые драйверы ПУ* - это специальные программы, дополняющие BIOS и обеспечивающие обслуживание новых ПУ или нестандартное использование имеющихся.

Файловая система MS DOS

Информация на дисках хранится в файлах. *Файл* - это поименованная область на диске для хранения информации любого вида. В MS-DOS файл характеризуется составным именем, атрибутами, датой и временем создания, длиной файла в байтах.

Составное (полное) имя файла складывается из имени файла и его расширения, характеризующего содержимое файла.

Атрибуты файла определяют способы его использования и права доступа к нему. ОС допускает задание следующих атрибутов:

- R - файл предназначен только для чтения, его нельзя удалить и изменить с помощью команд MSDOS;
- A - архивный файл, если его изменять, то будет создана архивная копия;
- H - скрытый файл, доступ к которому обычным способом невозможен;
- S - системный файл, Это скрытый, доступный только для чтения файл, используемый MSDOS. Он не поддается модификации или удалению.

Под файловой структурой понимают совокупность файлов и взаимосвязей между ними. Файловая система MSDOS позволяет объединить файлы в каталоги. *Каталогом* называется специальный файл, в котором регистрируются другие файлы и каталоги. Так образуется разветвленная (древовидная) файловая структура на диске.

На каждом диске всегда имеется единственный корневой каталог, создаваемый при его форматировании. В корневой каталог входят файлы и
подкаталоги 1-го уровня (NC, Э79, EXE), в подкаталоги 1-го уровня - файлы и подкаталоги 2-го уровня (Nc.exe, Nc.mnu, Be.exe, Primer.pas и т.д.).

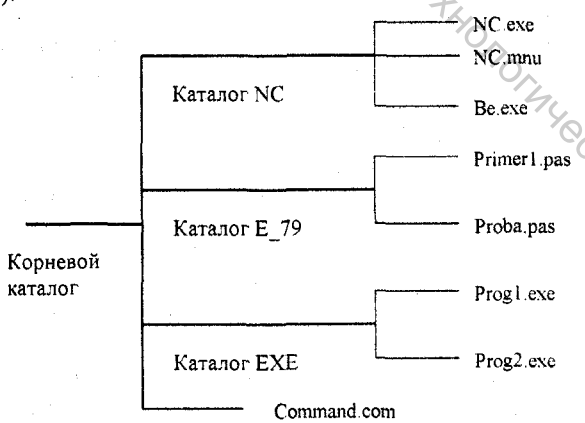


Рис.3. Дерево каталогов

Каталог, с которым в настоящий момент работает пользователь, называется *рабочим*. Его имя обычно указывается последним в строке приглашения MSDOS.

Если в команде ОС используется файл не из рабочего каталога, необходимо указывать *путь* к *файлу*. Путь состоит из

последовательности имен каталогов или символов «..», разделенных символом "\" (слэш). Для обеспечения доступа к файлу в общем случае необходимо задать его спецификацию:

[имя_диска:] [путь]имя_файла.

Например: d:\E_79\Proba.pas.

Кроме системного ПО, для MS DOS создано множество прикладных программ, которые могут работать на различных ПЭВМ, где установлена эта операционная система.

Одно из направлений развития MS DOS связано с созданием особых программных надстроек, называемых операционными оболочками, которые обеспечивают удобный интерфейс пользователя с ОС и некоторые другие дополнительные функции. К популярным программным надстройкам MS DOS относятся Norton Commander, PCToolls, Volkov Commander и некоторые другие. Они дают возможность компактно размещать на накопителе магнитных дисков необходимые для работы программы и данные, очищать диски от устаревших записей, быстро находить требуемую информацию, исправлять и дополнять ее, а также выполнять многие другие функции DOS без обращения к резидентным командам и утилитам ОС.

Другое направление развития ОС связано с поддержкой новых технических средств, в частности, с расширением до нескольких десятков и более мегабайт ОП, новых видов накопителей, видеомониторов и пр. Наиболее популярной операционной оболочкой, запускаемой на выполнение как обычная программа MS DOS, является WINDOWS 3.1, разработанная в 1991 г., и позже WINDOWS 3.11, как сетевой вариант.

Операционная система WINDOWS 3.1

Эта операционная система работает на базе MS DOS, используя на нижнем уровне внутренние функции и процедуры данной ОС. Принципиальным условием для программных приложений, предназначенных для работы в среде WINDOWS 3.1, является то, что они должны работать с внешними устройствами (монитор, принтер) не напрямую, а через универсальную систему команд. Управляющая система транслирует вызовы (обращение к тому или иному физическому устройству) и передает их соответствующему драйверу данного устройства, который непосредственно отвечает за работу с ними. Основу пользовательского интерфейса WINDOWS 3.1 составляют иерархически организованная система окон и другие графические объекты. Пользовательский интерфейс WINDOWS, в отличие от интерфейса командной строки MS DOS и оболочки NC, реализует оперативное управление на основе выбора того или иного графически визуализированного элемента (кнопки, пиктограммы и диалогового окна, мастера подсказок и др.) с помощью мыши, а команды клавиатуры и «горячие клавиши» имеют резервное значение.

Операционная система WINDOWS 95

WINDOWS 95 - это высокопроизводительная, многозадачная, многопотоковая 32-разрядная операционная система с графическим интерфейсом и расширенными сетевыми возможностями, работающая в защищенном режиме, поддерживающая 16-разрядные приложения без всякой их модификации. Это интегрированная среда, обеспечивающая эффективный обмен текстовой, графической, звуковой и видеоинформацией между различными программами. Функциональные возможности WINDOWS 95 качественно превосходят MS DOS, WINDOWS 3.1, WINDOWS 3.11. WINDOWS 95 - это полномасштабная информационная система семейства WINDOWS, не требующая MS DOS. Она полностью совместима с используемыми в настоящее время аппаратными и программными средствами. WINDOW95 - это первый представитель нового поколения 32-битовых ОС.

Преимущества WINDOWS 95

- *Интегрированная ОС*, ядро которой загружается в момент включения компьютера, активизируя графический интерфейс и обеспечивая полную совместимость с ОС MS DOS.
- *Вытесняющая многозадачность* - свойство ОС самостоятельно в зависимости от внутренней ситуации передавать или забирать управление того или иного приложения, не позволяющая одному приложению забирать все аппаратные ресурсы.
- *Многопоточность* - свойство ОС выполнять операции одновременно над потоками нескольких 32-битных приложений, называемых процессами.

WINDOWS 95 использует технологию Plug and Play, т.е. упрощает работу с компьютером за счет следующих функций:

- помощь при распознавании устройств для их установки и настройки;
- динамическое изменение состояния системы и автоматическое уведомление об этом программных приложений;
- интеграция драйверов устройств системных компонентов и пользовательского интерфейса.

WINDOWS 95 обеспечивает динамическое изменение конфигурации системы, упрощает настройку устройств и управление оборудованием. Специальная программа - диспетчер устройств - позволяет получать информацию обо всех найденных системой устройствах и изменить при необходимости их конфигурацию. WINDOWS 95 - это высокоэффективная платформа для мультимедиа, которая включает в себя CD или лазерный проигрыватель, обеспечивает поддержку VIDEO дисков и VIDEO магнитофонов, кроме того, имеет широкие возможности для пользователей с ограниченными возможностями:

- возможность масштабирования элементов интерфейса;
- запоминающие клавиши;

- режим Mouse Keys (все действия с мышью выполняются через клавиатуру);
- визуальное дублирование звуков системы.

Операционная система WINDOWS 98

WINDOWS 98 по сравнению с WINDOWS 95 включает средства, позволяющие компьютеру работать быстрее без добавления нового оборудования. В состав **WINDOWS 98** входит ряд программ, совместное применение которых повышает производительность компьютера.

В WINDOWS 98 надежность повышается за счет того, что применяются новые мастера. Например Мастер обслуживания позволяет быстрее выполнять программы, проверять жесткий диск на наличие ошибок и освобождать место на диске. Кроме того, служебные программы и ресурсы обеспечивают бесперебойную работу системы:

- Проверка системных файлов позволяет отслеживать состояние наиболее важных файлов, обеспечивающих работу компьютера. Если эти файлы повреждены или перемещены, программа проверки их восстанавливает.
- Проверка диска производится автоматически после неверного выключения ОС. Программа проверки дисков обнаруживает наиболее вероятные повреждения дисков и исправляет ошибки. Пользователь имеет возможность выполнить проверку диска в любой момент.
- Новый Web-узел ресурсов MS Windows Update автоматизирует процесс обновления драйверов и системных файлов и обеспечивает новейшие возможности технической поддержки.
- Проверка реестра. Эта системная программа позволяет обслуживать и проверять реестр на наличие ошибок и несогласованности структур данных при каждом запуске компьютера.
- Программы архивации, предоставляющие расширенные возможности архивации и восстановления данных.
- Использование мастером специальных возможностей, которые позволяют пользоваться ПК людям с физическими недостатками.

Обозреватель Интернета Internet Explorer позволяет сделать ряд функций доступными с рабочего стола WINDOWS:

- возможность поиска в сети из любого места на компьютере;
- возможность визуализации канала Web-узлов на рабочем столе;
- подписку на избранные узлы;
- настройку панели ссылок;
- настройку панели обозревателя;
- контроль содержимого из зоны безопасности при поиске Web.

Проводник WINDOWS и Internet Explorer позволяют определить ресурсы WINDOWS в едином представлении. WINDOWS 98 делает наиболее продуктивным использование Web за счет применения всех возможностей ПК к интерактивному содержимому Internet:

- автоматического добавления ранее вызывавшихся адресов web по мере их ввода;
- улучшенных списков часто посещаемых web-узлов;
- улучшенного журнала и возможности обследования часто посещаемых web-узлов;
- поддержки всех основных стандартов Интернета, в т.ч. и Java;
- высокой производительности динамических документов HTML, что позволяет сделать web-страницы разнообразными и интересными;
- нового мастера подключений к Internet, позволяющего зарегистрироваться для доступа к Интернет и автоматически выполнить шаги по настройке ПО для доступа к сети. Приложение Inernet Explorer объединяет рабочий стол с web, благодаря чему рабочий стол и его папки будут выглядеть и действовать так же, как и при работе с сетью. Аналогично просмотру содержимого web можно просматривать и содержимое компьютера, и при этом просмотр web возможен из любого места компьютера. Кроме того, на рабочий стол и на панель задач можно добавить активное содержимое, такое, как, например, бегущая строка новостей.

Операционная система WINDOWS 2000

Основные новые технические решения, реализованные в Windows2000:

1. Защита данных средствами протокола аутентификации, обеспечивающего передачу данных незащищенным путем. Именно он, наряду со службой каталогов Active Directory, позволяет обеспечивать важнейшую особенность Windows 2000 по сравнению с Windows NT и предоставляет ряд преимуществ по сравнению с HTML-протоколами аутентификаций.
2. Инфраструктура открытых ключей. ОС Windows 2000 оснащается средствами защиты информации на базе структуры открытых ключей. Эта структура представляет собой систему цифровых сертификатов и организаций, удостоверяющих сертификаты CAS. Так же, как и система Kerberous, она дает возможность обоим участникам транзакции(участникам обмена сообщениями) проверить, действительно ли партнер является тем, за кого себя выдает, а также шифровать транзакции. Система защиты данных на базе структуры открытых ключей лучше всего подходит для использования в сети Интернет и следовательно, является полезным средством для организации электронных конференций между предпринимателями. Почтовое ведомство США использует структуру открытых ключей для предоставления клиентам круглосуточного доступа по каналам Интернет. Включение инфраструктуры открытых ключей системы Windows 2000 дает основание полагать, что уровни безопасности и шифрования данных

станут неотъемлемой частью в повседневной коммерческой практике.

3. Windows 2000 оснащена рядом средств, предназначенных для пользователей малых офисов. В состав системы входит качественный маршрутизатор, обладающий интерфейсом подключения по запросу, что облегчает процесс установки соединения с интернет-провайдером. При помощи таких средств, как реализованный в Windows 2000 сервер-доступ, к одному предоставленному выходу в Интернет направляется сразу несколько пользователей сети. В данном модуле реализована технология трансляции сетевых адресов NAT, которая дает возможность компьютерам, объединенным в единую локальную Windows- сеть выходить через маршрутизатор в Интернет, используя единственный IP-адрес, предоставленный Internet-провайдером. Множество пользователей локальной сети являются, с точки зрения провайдера, единым клиентом.
4. Windows 2000 имеет встроенные средства удаленного доступа.
5. Новые функции управления хранением данных: управление квотами дискового пространства, иерархическая система хранения данных, динамическое управление ими, а также путем внесения изменений в файловую систему NTFS.
6. Дополнительные возможности для мобильных пользователей, доступ к сетевым файлам и папкам для работы с ними без подключения к сети. ОС создает локальный КЭШ, сохраняет в нем файлы и папки, и они синхронизируются с оригиналами в фоновом режиме. В Windows 2000 предусмотрена функция перехода в режим «спячки». Перед отключением питания все содержимое физической памяти записывается в специальный файл.
7. Расширенное административное управление ОС, что позволяет снизить общую стоимость эксплуатации ОС.

Операционная система WINDOWS XP

Операционная система Windows XP официально была представлена 25 октября 2001 года. Она должна была заменить на прилавках Windows 2000, которая так и не обрела большой популярности.

Windows XP - система, позволяющая работать в терминальном режиме, имеющая возможность одновременной работы неограниченного количества пользователей при добавлении утилиты Desktop manager и позволяющая работать одновременно на четырех рабочих столах. Windows XP выпущена в трех версиях: XP Home Edition, XP Professional и XP 64-Bit Edition.

Операционная система XP home, предназначена для домашнего использования и малого бизнеса, по сути является модификацией Windows 9x.

Операционная система XP Professional предназначена для пользователей Windows 2000 или Windows NT. Хотя XP Home Edition и XP Professional базируются на одном и том же ядре, ориентированная на применение в сфере бизнеса XP Pro обладает более расширенной функциональностью по сравнению с домашней системой. Но требования и для XP Pro, и для XP Home примерно похожи.

Основные технические решения, реализованные в WindowsXP:

1. Windows XP построена на усовершенствованном ядре Windows 2000 и имеет новый, простой интерфейс, который упрощает работу с компьютером.
2. Windows XP имеет службу терминалов Remote Assistance, которая позволяет дистанционно подключаться к компьютеру. Таким образом можно получить доступ к любым документам и файлам на удаленной машине.
3. Создан откат драйвера (Rollback Driver), который препятствует сбою системы.
4. Добавлена функция "System Restore" или восстановление системы. С ее помощью пользователь создает системную метку, ОС запоминает конфигурацию системы, а затем, в случае неполадок, можно вернуть систему на некоторое время назад, когда всё было в порядке.
5. Windows XP позволяет записывать компакт диски без привлечения посторонних программ. Нажатием одной кнопки файл может легко быть записан на диск.
6. Имеется встроенный архиватор, типа Zip.
7. В систему WindowsXP входит новый обозреватель интернета Internet Explorer версии 6.0, а также почтовая программа Outlook Express 6 . Версия Direct-8.
8. Брандмауэр интернет-подключений — автоматически защищает подключенный к интернету ПК от несанкционированного доступа.
9. Проигрыватель Windows Media для Windows XP — полнофункциональное средство, обеспечивающее поиск, воспроизведение, упорядочивание и хранение цифрового мультимедиа-материала.
10. Мастер установки сети — помогает легко подключать и совместно использовать компьютеры и устройства, применяемые в домашних условиях.
11. Служба сообщений Windows Messenger — эффективное средство связи и совместной работы, поддерживающее передачу немедленных сообщений, проведение голосовых и видеоконференций, а также совместное использование приложений.
12. Центр справки и поддержки — упрощает решение текущих проблем и помогает своевременно получать необходимую техническую поддержку.
13. Беспроводное подключение — автоматическая беспроводная конфигурация сети с использованием стандарта 802.1x..

14. Автономные файлы и папки — доступ к файлам и папкам, хранящимся на общем сетевом диске даже во время отключения компьютера от сервера.
15. Шифрованная файловая система — защита важных данных, содержащихся в файлах, хранящихся на диске, на котором используется файловая система NTFS.
16. Управление доступом — запрещение доступа к избранным файлам, приложениям или другим ресурсам.
17. Отображение текста на разных языках (технология Single Worldwide Binary) — можно вводить текст на любом языке и запускать версию приложений Win32 для любого языка, используя соответствующую версию операционной системы Windows XP.
18. Многоязычный пользовательский интерфейс — можно менять язык пользовательского интерфейса, чтобы работать с локализованными диалоговыми окнами, меню, файлами справки, словарями, средствами проверки правописания и т.д.

Операционная система UNIX

Семейство **ОС UNIX** является альтернативой семейству WINDOWS. Основное отличие и преимущество этой операционной системы заключается в том, что она может использоваться для широкого круга аппаратных платформ. Начиная с момента своего появления в 1969 году, ОС UNIX получила широкое распространение на машинах различной мощности и архитектуры. Эта ОС является не только многозадачной, но и многопользовательской системой, которая позволяет нескольким пользователям разделить вычислительные ресурсы одного компьютера. В первых версиях UNIX взаимодействие с пользователем осуществлялось с помощью командной строки, затем появились варианты графического интерфейса, позволяющие существенно облегчить работу и сделать систему легкодоступной. В настоящее время существует много версий UNIX от различных производителей. Наиболее известные версии UNIX: SUN OS, Solaris для компьютеров SUN, ATX, IRIX – для IBM, SCON IX – для SCON. Эти ОС созданы для платформы Intel.

В настоящее время все большую популярность приобретают версии UNIX для персональных компьютеров. Одной из них является ОС LINUX, поддерживающая большинство свойств, присущих ОС UNIX. К особенностям LINUX можно отнести следующие:

- поддержка национальной и стандартной клавиатур,
- динамически загружаемые драйверы,
- поддержка различных версий файловых систем,
- реализация файловой системы MS DOS, позволяющей прямо обращаться к файлам MS DOS на жестком диске,
- поддержка полного набора протоколов TCP/IP для работы в сети.

Операционная система WINDOWS NT

Сетевая операционная система, предоставляющая возможность работы каждому пользователю в индивидуальной области, защищенной паролем, с распределенным доступом к сетевым ресурсам. Эта операционная система существует только для аппаратных платформ Alpha и Intel. Широкого распространения она не получила и в настоящее время используется ограниченно.

ЛЕКЦИЯ 4. СЕРВИСНЫЕ ПРОГРАММЫ И СИСТЕМЫ ОБСЛУЖИВАНИЯ

Сервисные программы – это вспомогательные инструменты, расширяющие и дополняющие функциональные возможности ОС. К сервисным программным средствам относятся:

- программы обслуживания диска;
- программы архивации;
- антивирусные средства.

Программы обслуживания диска

Форматирование диска

Накопители на магнитных носителях, перед тем как их можно будет использовать в качестве носителей информации должны пройти специальное форматирование. Для этого обычно используется стандартная программа Windows, которая доступна при вызове контекстного меню соответствующего устройства. Можно форматировать диски Floppy устройств, жесткие диски и диски других устройств, допускающих эту процедуру. Исключение составляют системные устройства, с которых была проведена загрузка системы. Форматировать можно диски, учитывая, что при этом теряется информация, записанная на диске. Полное форматирование целесообразно применять для дисков, на которых появились сбойные участки.

Дефрагментация диска

Эта процедура связана с особенностью использования системы FAT. Доступ к файлу, расположенному в одном месте диска (файл размещается последовательно в смежных кластерах), занимает меньше времени, чем доступ к файлу, фрагменты которого разбросаны по всему диску. Чем больше фрагментов, тем меньше доступ к ним. Для увеличения фрагментов системы диск необходимо периодически дефрагментировать.

Проверка диска на наличие ошибок

В процессе эксплуатации диска возможно появление ошибок, связанное со сбоями в процессе записи на него. Целостность файловой системы обычно определяется:

- правильностью имен файлов (файлы с неправильными именами нельзя открыть);
- правильностью даты и времени создания файла (неправильная дата и время могут влиять на работу многих программ, обрабатывающих такие файлы);
- уникальностью имен файлов;
- отсутствием мастеров, не принадлежащих ни одному файлу;
- отсутствием файлов с общими кластерами.

Проверка поверхности диска на наличие повреждений и аппаратных ошибок занимает больше времени, чем проверка целостности файловой системы. Для сокращения времени работы программы можно выбрать проверку не всего диска, а только системной области или области данных. При нахождении поврежденных секторов их данные обычно переносятся в другие сектора. Некоторые системные файлы нельзя переносить, поэтому рекомендуется исправление ошибок в секторах, в которых располагаются такие файлы, отменить. Для большей надежности проверки поверхности диска необходимо использовать режим записи на диск, при котором данные на диске не теряются. Для исправления других ошибок необходимо использовать другие программы, например NORTON Utilities.

Очистка диска

В процессе эксплуатации дисковых носителей свободное пространство заполняется файлами. Для создания новых файлов необходимо освободить место, занимаемое уже ненужными файлами, – они удаляются. Удаленные файлы помещаются в корзину на рабочем столе. Они меняют прописку, но не удаляются с диска. Для освобождения места, занятого файлами, необходимо очистить корзину на рабочем столе.

Программы архивации

Наиболее простым и эффективным способом восстановления испорченных данных на компьютере является создание копий. Для уменьшения места, необходимого для содержания копий и удобства эксплуатации, создано большое количество программ, позволяющих работать с архивными файлами. Они могут содержать один или более файлов в сжатом виде. Можно как извлекать из этого файла исходные данные, так и добавлять новые. Наиболее популярные программы архивации под Windows: WINZIP, WINRAR, WINAZJ, NETZIP.

Антивирусные средства

Вирус – это, как правило, небольшая по объему последовательность программных кодов, обладающих следующими свойствами:

- возможностью создавать свои копии и внедрять их в другие программные объекты;
- обеспечением скрытности до определенного момента;
- несанкционированностью (со стороны пользователя) производимых ею действий;
- наличием отрицательных последствий от ее функционирования.

Не все вирусы обладают всеми из свойств, но программы, которые обладают одним или некоторыми свойствами, могут и не являться вирусами. К причинам создания и развития индустрии вирусов можно отнести причины технического (ранние версии операционных систем), экономического (борьба с конкурентами) и субъективного (компьютерное хулиганство) характера.

Классификация вирусов:

- загрузочные вирусы;
- файловые вирусы;
- макровирусы;
- сетевые вирусы.

Загрузочные вирусы внедряются в так называемые загрузочные области носителя. Эти области – это сектора накопительных устройств, предназначенные для хранения специальных программ, загрузчика ОС (если заражен системный диск, с которого происходит загрузка системы, то управление получит не обычная программа загрузки, а защищающая ее код от вируса). При заражении вирус считывает необходимую информацию с первоначального загрузчика и сохраняет ее в своих кодах. Как правило, загрузочные вирусы являются резидентными, т.е. при загрузке системы вирус устанавливается в память компьютера и находится там постоянно, перехватывая необходимое прерывание и производя все действия, на которые он запрограммирован.

Файловый вирус использует особенности организации системы. Такие вирусы внедряются в выполняемые файлы, библиотеки и объектные модули, загружаемые драйвера, файлы исходных текстов программы. Файловые вирусы внедряются в начало, конец или середину файла. Такие файлы остаются работоспособными, так как код вируса дописывается к исходному коду, т.е. некоторые вирусы, вместо исходного содержимого файла, записывают свой код. Файловые вирусы могут и не изменять содержимое зараженных файлов. Иногда для своей активизации они используют свойство ОС, определяющее порядок запуска программы. Так, если в одном каталоге есть файлы с одинаковыми именами, то сначала запускается файл с расширением *.BAT, потом *.COM и *.EXE. Следовательно, если зараженный файл имеет расширение EXE, то вирус файлы с тем же именем, но с расширением COM будет запускать раньше.

Макровирусы – это файловые вирусы, использующие особенности файлов, документов, популярных редакторов и элементов таблиц. В этих файлах размещаются программы на макроядрах, возможности которых позволяют создавать программу-вирус.

Сетевые вирусы используют для своего распространения возможности компьютерных сетей. Первоначально использовали ошибки в сетевых протоколах и программном обеспечении глобальных сетей. В настоящее время эти ошибки исправлены, и такие вирусы не имеют распространения. Сетевые вирусы распространяются, используя возможности электронной почты, в основном это макровирусы.

Резидентные вирусы могут оставлять свои копии в оперативной памяти. Они остаются активными и после завершения работы программы (и даже после удаления файла с вирусом, вплоть до перезагрузки системы). Заражение компьютера происходит через устройства, позволяющие

вводить информацию в компьютер: модем, сетевую карту, устройства хранения информации со сменными носителями.

Способы защиты от вирусов.

Одним из основных последствий деятельности вирусов является потеря или порча информации, поэтому для обеспечения устойчивой и надежной работы желательно всегда иметь эталонные (незараженные, чистые) копии используемой информации и программного обеспечения. При заражении вирусом нарушается целостность информации. Использование специальных утилит, которые позволяют отслеживать информацию о системных отличиях и файлах (контролировать размеры, дату создания и др.), относительно быстро позволяет обнаружить проникновение вирусов. Использование специальных антивирусных средств в большинстве случаев позволяет не только определить вирусное вторжение, но и оперативно обезвредить обнаруженные вирусы и восстановить испорченную информацию. Популярные антивирусные эффективные программные системы: антивирус Касперского (AVP), Norton Antivirus, Doctor Weber (DRW).

К сожалению, не существует антивирусов, которые бы обеспечили полную защиту от вирусов. При выборе антивирусного обеспечения следует принимать во внимание перечень и качество предоставляемых услуг:

- возможность сканирования «на лету», когда любой объект, к которому осуществляется доступ, проверяется на наличие вируса;
- возможность сканирования по запросу, когда время и область действия антивируса определяются пользователем;
- возможность настройки антивируса в соответствии с потребностями пользователя.

Качество работы антивируса определяется его надежностью (отсутствие ошибок, зависания) и эффективностью (обезвреживание всех вирусов), отсутствием ложных срабатываний.

Системы программирования

Это языки программирования высокого уровня, машинно-ориентированные языки и инструментальные средства.

Машинно-ориентированные языки - это языки, более понятные ЭВМ. Среди наиболее распространенных машинно-ориентированных языков прежде всего можно выделить Ассемблер. Между тем, если посмотреть, что происходит в мире современных компьютеров, нетрудно заметить, что почти все программное обеспечение создается на языках высокого уровня, более понятных для человека. Программу, написанную на этих языках, может разобрать не только ее автор, а в ряде случаев - даже непрограммист. Несмотря на то, что языков программирования сегодня гораздо более пятисот, их нетрудно разделить на несколько основных групп или семейств.

В первую группу можно объединить все алгоритмические языки, т.е. те, что позволяют заложить в программу способ решения - алгоритм. Наиболее известные представители этой самой большой семьи - Фортран, Алгол, Ада, Бейсик, Паскаль, Модула-2, Си.

Фортран - формульный транслятор. Предназначен для решения научно-технических задач. Разработан фирмой IBM в середине 50-х годов. Недостатком Фортрана является то, что он плохо поддается структурированию, т.е. программу нельзя представить в виде отдельных модулей.

Бейсик - универсальный символический код инструкций для начинающих. Прямой потомок Фортрана. Появился в 1964 году. До сих пор самый популярный среди языков программирования, поскольку наиболее подходит именно для использования на ПЭВМ, т.к. занимает в ее памяти совсем немного места. В отличие от Фортрана, Бейсик умеет обрабатывать не только числовые, но и символьные переменные, поэтому язык оказался удобным для решения большинства примитивных повседневных задач. Разновидности: TurboBasic, GWBasic, Visual Basic for Applications.

Паскаль (назван так в честь знаменитого математика Блеза Паскаля)проигрывает в популярности только Бейсику. Язык истинно структурного программирования. Это значит, что программу на Паскале можно формировать из отдельных, совершенно независимых процедур и функций, каждая из которых призвана выполнять определенную ограниченную задачу. При таком подходе становится возможным быстрое создание больших программных комплексов.

Прямые наследники Паскаля - *Модула-2* и *ПЛ/1*, в которых учтены все слабые места Паскаля.

С, С++ задуманы как инструмент для реализации и развития известной ОС Unix. Возникли в начале 70-х годов. Являются языками системного программирования и позволяют решать такие задачи, которые в ином случае потребовали бы машинно-ориентированного языка Ассемблера. Привлекательная черта С и С++ - наличие всех особенностей, присущих современному универсальному языку: структурности, модульности, определения типов данных, рекурсивности. Для начинающих программистов эти языки являются достаточно сложными.

Вторая группа - *это логические функциональные языки*, которые привлекают сейчас всеобщее внимание в связи с активными исследованиями в области искусственного интеллекта. Эта семья невелика, в качестве представителей можно назвать Лисп, Пролог.

Лисп (от англ. lisp - шорох, лепет), в отличие от процедурных языков типа Бейсика или Паскаля, является языком функционального программирования, ориентированным на динамическую обработку данных. Эффективен для работы со списками, может быть использован для задач, где отсутствует четкий алгоритм решения.

Пролог - программирование в логике. Европейский противовес американскому Лиспу. Относится к языкам для создания систем искусственного интеллекта, т.е. компьютер должен самостоятельно выводить желаемый результат из фактов и правил, не получая от программиста путь решения. Программист лишь определяет объекты и логические связи между ними.

Прикладные программные средства (ППС)

ППС составляют категорию программных средств, обращенных непосредственно к пользователю. Их назначение состоит в том, чтобы пользователь с помощью компьютера мог решать свои повседневные задачи, приобретал определенные навыки, проводил свой досуг.

К основным прикладным системам можно отнести:

- *системы управления базами данных* (СУБД), которые обеспечивают эффективные действия по обработке больших объемов информации, их поиску, корректировке и хранению. СУБД могут иметь свой язык программирования. (Dbase-подобные СУБД, FoxPro, Access);
- *текстовые и графические редакторы*, позволяющие наиболее удобным образом создавать, корректировать текстовую и графическую информацию. (Word, Page Maker, Corel Draw);
- *электронные таблицы*, обеспечивающие удобные средства работы с двухмерными таблицами. (Lotus, SuperCalc, Excel);
- *пакеты прикладных программ*, содержащие стандартные подпрограммы, для производства расчетов по известным алгоритмам;
- *прикладное программное обеспечение*, используемое для автоматизации работ в конкретной области (бухучет -1С, Бемби, система управления предприятием – автоматизированный комплекс «Галактика», САПР - AutoCAD);
- *обучающие системы* в различных областях знаний (энциклопедии).
- *игры* развлекательного и тренировочного характера;
- *вирусы* - программы, мешающие работе ПЭВМ, или разрушающие информацию и оборудование компьютера; они используются для защиты запатентованных программ или систем, нанесения информационного ущерба;
- *антивирусы* - программы для обнаружения и обезвреживания вирусов.

Могут использоваться также интегрированные системы, объединяющие наиболее часто используемые прикладные программы. Но любая прикладная программа может работать только под управлением определенной операционной системы.

Таким образом, структура программного обеспечения ПЭВМ может быть представлена в виде схемы (см. рис.4):

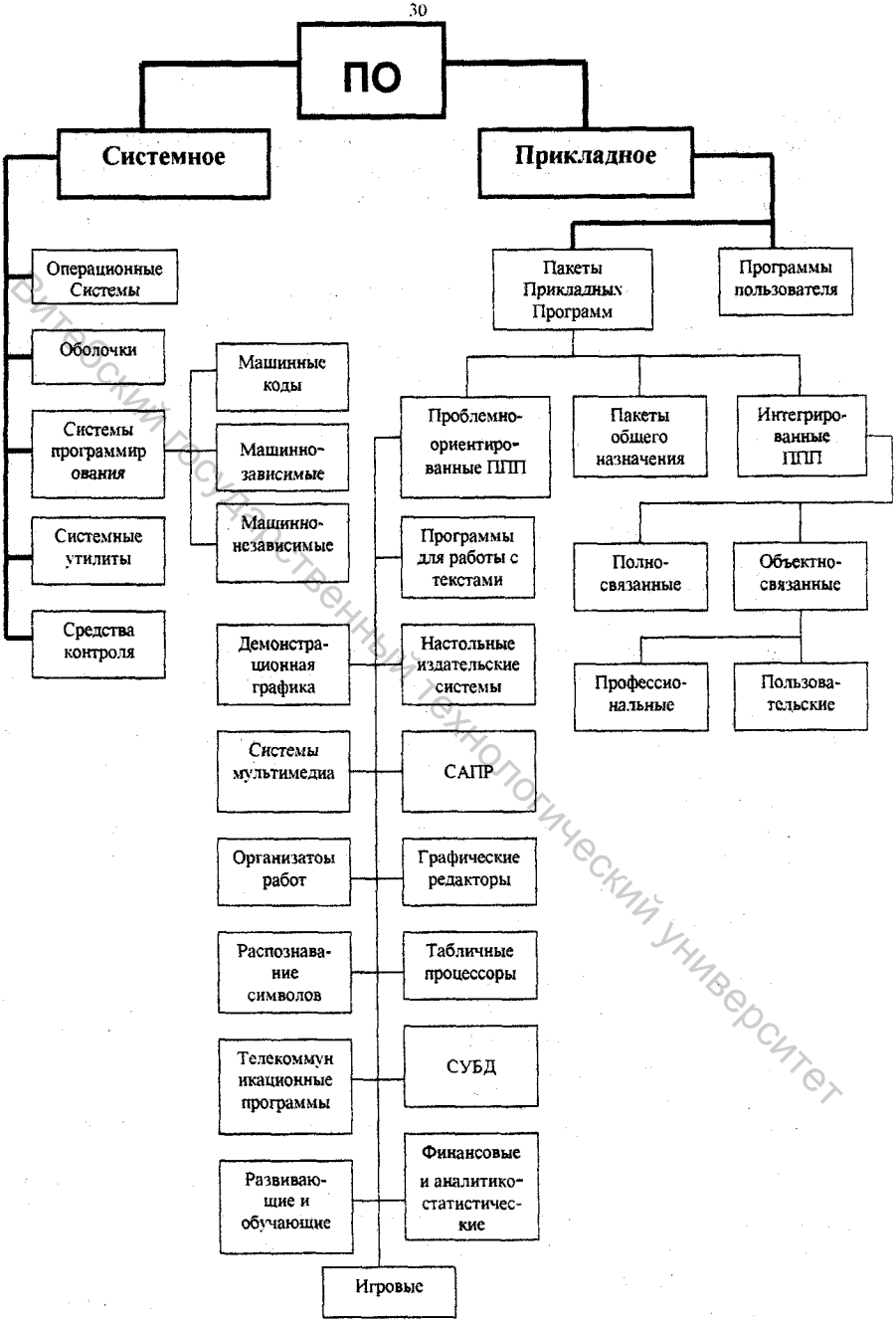


Рис. 4. Структура ПО

ЛЕКЦИЯ 5. АЛГОРИТМЫ И ИХ СВОЙСТВА

Этапы подготовки и прохождения задачи в ЭВМ

Решение задачи на ЭВМ представляет собой сложный и трудный процесс, в котором на различных этапах принимают участие различные специалисты: экономисты, инженеры, математики, программисты, статистики и т.д. Этот процесс можно условно разбить на следующие этапы:

1. *Постановка задачи.* На этом этапе специалист формирует цели решения задачи. Подробно раскрывает ее содержание, указывает количество и характер всех величин, используемых в задаче в качестве исходных данных.
2. *Математическая формулировка задачи.* Здесь устанавливаются в окончательном виде те формулы и математические зависимости, которые подлежат решению.
3. *Выбор численного метода.* Среди всех численных методов решения данной задачи выбирается тот из них, который наилучшим образом обеспечивает выполнение всех требований этой задачи.
4. *Разработка логической схемы* алгоритма и ее минимизация. Разрабатывается алгоритм решения, т.е. устанавливается необходимая последовательность арифметических и логических действий, с помощью которых может быть реализован выбранный численный метод. Если возможно, то алгоритм упрощается (минимизируется) с помощью операций и формул математической логики. Полученный алгоритм изображается наглядно в виде блок-схемы.
5. *Программирование.* Алгоритм решения задачи по блок-схеме записывается на языке конкретной ЭВМ или на одном из алгоритмических языков.
6. *Отладка программы.* Задача данного этапа состоит в том, чтобы путем опробования на машине вновь разработанной программы выявить ошибки, допущенные на всех предыдущих этапах. Для проверки правильности вычислений по составленной программе вручную решается один из вариантов задачи, называемый отладочным или контрольным вариантом. Затем этот же вариант по составленной программе просчитывается на ЭВМ. При совпадении результатов счета на машине и контрольного варианта считают, что программа составлена правильно.
7. *Решение задач на машине.* После отладки программы производится непосредственное решение задачи на машине с целью получения результатов для всех вариантов исходных данных.
8. *Анализ полученных результатов.* Производится специалистом, поставившим задачу. Если результаты по каким-либо причинам не устраивают, принимаются решения либо об изменении алгоритма, либо об уточнении исходных данных.

Понятие алгоритма

Это понятие является одним из основных а информатике. Слово «алгоритм» происходит от имени узбекского математика аль-Хорезми, что означает «из Хорезма». В IX веке он разработал правила арифметических действий над десятичными числами. Длительное время алгоритмами пользовались только математики, понимая под алгоритмом любое описание процесса решения задачи. Описание процессов решения математических задач предназначалось для человека, поэтому не требовалось большой строгости в описании действий.

С развитием техники появились автоматические устройства, способные воспринимать команды и исполнять соответствующие действия. Для таких устройств правила решения должны быть четко сформулированы, то есть исполнителю необходимо указать последовательность действий, которые он должен выполнить для достижения цели – получения результата. Иначе говоря, исполнителю должен быть указан алгоритм решения задачи, представленный на понятном ему языке. Под исполнителем подразумевается как человек, так и вычислительная машина.

Алгоритм решения задачи – это конечная последовательность чётко сформулированных правил решения некоторого класса задач, приводящая к получению результата.

В обыденной жизни мы часто сталкиваемся с алгоритмами как последовательностью действий, приводящих к достижению поставленной цели, – это правила перехода улицы, рецепты приготовления различных блюд, поиск нужного слова в словаре и др. Алгоритмы в математике – это правила нахождения корней квадратных и алгебраических уравнений, правила выполнения арифметических действий, правила нахождения наибольшего общего делителя и наименьшего общего кратного, разложения числа ни простые множители и др.

Свойства алгоритма

Для адекватности восприятия и понимания исполнителем алгоритмы должны обладать целым рядом свойств: дискретностью, точностью, понятностью, результативностью, массовостью. Таким образом, в качестве основных свойств алгоритма можно выделить следующие:

дискретность – это разбиение алгоритма на ряд отдельных законченных действий – шагов;

точность – это четкое указание последовательности шагов;

понятность – это однозначное понимание и исполнение каждого шага алгоритма его исполнителем;

результативность – обязательное получение результата за конечное число шагов;

массовость – применимость алгоритма к решению целого класса однотипных задач.

Если бы человек знал алгоритм решения всех задач, то их исполнение можно было бы поручить машине. Но оказалось, что не все задачи, которые нам хотелось бы решить, имеют алгоритмы решения. Задачи, в принципе не имеющие общего решения, называют алгоритмически неразрешимыми. Это, например, задача о трисекции угла, о квадратуре круга и др.

Способы описания алгоритмов

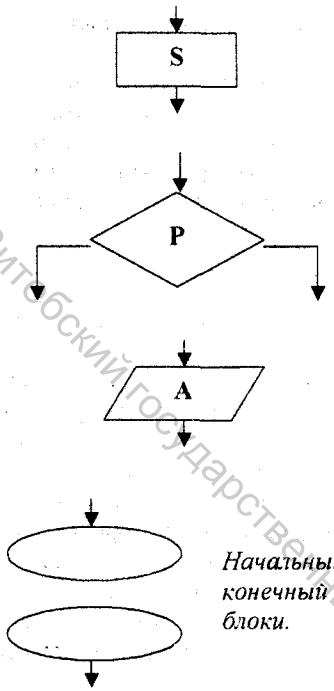
Существуют различные способы представления алгоритмов. Рассмотрим некоторые из них.

Текстовый – алгоритм решения задачи задается в виде текстового описания шагов решения. Таким способом составляются различные инструкции (например пользования бытовыми приборами, управление автоматическими игрушками), рецепты приготовления блюд и др. Такие алгоритмы предназначены для человека, поэтому в качестве команд могут использоваться привычные для человека предложения, фразы. Форма записи должна быть такой, чтобы был ясен порядок исполнения команд.

Математический – такой способ описания алгоритма также предназначен для человека и представляет собой логически связанную последовательность математических формул, реализующих решение задачи.

В виде блок-схем – схема алгоритма представляет собой графическое изображение алгоритма с помощью определенным образом связанных между собой блоков. Под блоком подразумевается любой конечный этап вычислительного процесса, принимаемый в данной схеме как целое. Внутри каждого блока дается описание содержащихся там операций. ГОСТом приняты определенные обозначения, используемые для составления блок-схем (см. рис. 5).

На языке программирования – на основании алгоритма, сформулированного одним из перечисленных способов, программистом составляется программа решения поставленной задачи на специальном языке для записи алгоритмов. Такие языки получили название *алгоритмические языки*, или *языки программирования высокого уровня*.



Функциональный блок. Используется для представления функций. В этом блоке осуществляется преобразование информации по заданному действию S.

Блок проверки условия используется для управления преобразованием информации. В результате анализа выполнения условия P выбирается одно из двух возможных направлений «да» или «нет», и управление передается блоку, записанному на выбранном направлении.

Информационный блок используется для обозначения операций обмена информации между устройствами.

Рис.5. Условные обозначения для составления блок-схем

Базовые структуры.

При изображении алгоритмов при помощи схем используются базовые управляющие структуры: *следование, развилка, повторение (см. Рис.6, 7, 8).*

Следование: структура, означающая, что действия S1 и S2 должны быть исполнены одно за другим.

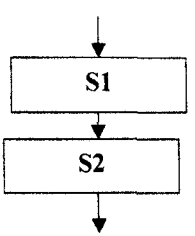


Рис.6. Структура следование

Развилка – это действие, определяемое структурой и осуществляющее анализ условия P (истинно или ложно) и альтернативный выбор дальнейшего направления в последовательности выполнения действий в зависимости от значения P.

Различают

полную развилку *и* *неполную развилку*

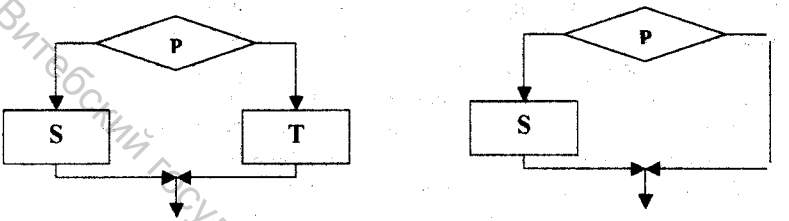


Рис.7. Структура развилка

Словесно *полная развилка* описывается так: если P истинно, то исполнить S, иначе T. *Неполную развилку* словесно можно описать так: если P истинно, то исполнять S.

Повторение – это структура, описывающая циклические вычислительные процессы.

Различают:

цикл-пока и *цикл-до*.

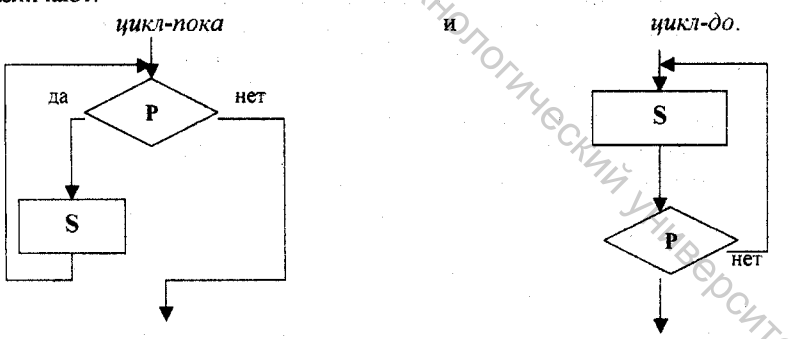


Рис. 8. Структура повторение

Цикл-пока: пока условие **P** истинно, выполнять тело цикла **S**.

Цикл-до: выполнять тело цикла **S**, до тех пор, пока условие **P** не станет истинным.

Все пять алгоритмических структур могут комбинироваться одна с другой, как того требует алгоритм.

ЛЕКЦИЯ 6. АЛФАВИТ ЯЗЫКА TURBO PASCAL. ИДЕНТИФИКАТОРЫ. КОНСТАНТЫ, ПЕРЕМЕННЫЕ, ТИПЫ ДАННЫХ

На протяжении уже многих лет среди программистов достаточно популярен язык программирования **Паскаль (Pascal)** Этот язык был разработан Никласом Виртом первоначально для целей обучения программированию вообще, но благодаря ряду особенностей получил достаточно широкое распространение и в практической работе.

Во-первых, по своей идеологии Pascal близок к современной методике и технологии программирования. В частности, этот язык весьма полно отражает идеи структурного программирования.

Во-вторых, Pascal хорошо приспособлен для применения технологии разработки программ методом нисходящего проектирования (пошаговой детализации).

В-третьих, Pascal предоставляет весьма гибкие возможности в отношении используемых структур данных.

Хотя Pascal создавался для целей обучения, он хорошо продуман с точки зрения эффективности реализации самого языка и получаемых в результате трансляции машинных программ. Большое внимание в языке уделено вопросу повышения надежности программ: средства языка позволяют осуществлять достаточно полный контроль правильности использования данных различных типов и программных объектов.

Как любой алгоритмический язык, Паскаль образуют три его составляющие: алфавит, синтаксис и семантика [1].

Алфавит – это фиксированный для данного языка набор основных символов, т.е. «букв алфавита», из которых должен состоять любой текст на этом языке – никакие другие символы в тексте не допускаются.

Синтаксис – это система правил, определяющих допустимые конструкции из букв алфавита. С помощью этих конструкций представляются отдельные компоненты алгоритма и алгоритм в целом, записанные на данном языке.

Семантика – это система правил истолкования отдельных языковых конструкций, позволяющих однозначно воспроизвести процесс обработки данных по заданной программе.

Лексемы языка

Лексема – это минимальная конструкция языка, состоящая из символов алфавита, которая имеет смысловое значение. Рассмотрим следующие виды лексем.

Слова. Слова подразделяются на зарезервированные слова, стандартные идентификаторы и идентификаторы пользователя (идентифицировать - отождествлять).

Зарезервированные слова являются составной частью языка, имеют фиксированное начертание и раз и навсегда определенный смысл. Всего зарезервированных слов около 60-ти (например, program, begin, end).

Стандартные идентификаторы служат для обозначения заранее определенных разработчиками языка типов данных, констант, процедур и функций. (Sin, Ln).

Идентификаторы пользователя применяются для обозначения меток, констант, переменных, процедур и функций, определенных самим программистом.

При записи алгоритма решения задачи на языке программирования необходимо знать правила написания и использования элементарных информационных и языковых единиц.

Программа на языке Turbo Pascal формируется с помощью конечного набора знаков, образующих алфавит языка, и состоит из букв, цифр и специальных символов. В качестве *букв* используются прописные и строчные буквы латинского алфавита **A.....Z, a.....z** и знак подчеркивания, в качестве *цифр* **0.....9.**

Существуют общие правила написания идентификаторов:

- идентификатор начинается только с буквы или знака подчеркивания;
- идентификатор может состоять из букв, цифр и знака подчеркивания (пробелы, точки и другие специальные символы недопустимы);
- между двумя идентификаторами должен быть хотя бы один пробел;
- максимальная длина идентификатора 127 символов, все они значимы.

При написании идентификаторов можно применять как прописные, так и строчные буквы.

П р и м е р

SUM2

1KL – ошибка, идентификатор начинается с цифры.

BL_6

NOMER.DOMA – ошибка, идентификатор содержит точку.

Знаки операций. К ним относятся

Знак операции	Его смысл
*	умножение
/	деление
+	сложение
-	вычитание
=	равно
< >	не равно

<	меньше
<=	меньше или равно
>	больше
>=	больше или равно

Разделители. Неделимые последовательности знаков алфавита образуют слова, отделенные друг от друга разделителями и несущие определенный смысл в программе. Они используются для того, чтобы придать тексту программы наглядность. Группу разделителей образуют следующие символы:

< разделитель> – « ;» « ,» « :» « .» .

Комбинации специальных символов могут образовывать *составные символы*:

:= присваивание

.. диапазон значени.

Кроме того, используются круглые **( )** и квадратные **[]** скобки.

Для комментариев, то есть пояснений к программе, используются символы **{ }** и **(* *)** – альтернатива фигурных скобок.

Константы

Это элементы данных, значения которых известны заранее и в процессе выполнения программы не изменяются. Различают следующие виды констант:

1. *Литералы*, к которым относятся:
   - десятичные числа (например, 69, -619, 21.14 и т.п);
   - целые шестнадцатиричные числа, которые начинаются со знака $: ($B, $3A и т.п.);
   - символьные константы ‘A’, ‘45’, ‘?’ и т.п.;
   - строковые константы ‘FFF’, ‘DOM’, ‘BOX’ и т.п.
2. *Именованная константа* – идентификатор, которому должно быть присвоено значение. Все именованные константы должны быть описаны в разделе описания констант программы. В языке Turbo Pascal для определения констант служит зарезервированное слово Const.

   Формат: Const

   <идентификатор> = <значение константы>;

   Например:

   Const
   Max = 1000;
   Arg = - 54.69;
   C = ‘GRUPPA’;

Имеется ряд констант, к значениям которых можно обращаться без предварительного определения.

Таблица 1. Зарезервированные константы

Идентификатор	Тип	Значение	Описание
Pi	Real (вещественный)	3,141592..	Число "пи"
True	Boolean (булевский)	True	"Истина"
False	Boolean (булевский)	False	"Ложь"
Maxint	Integer (целочисленный)	32767	Максимальное целое число

3. *Типизированная константа*, для которой указывается не только значение, но и тип.

Например:

Const
GR: INTEGER = 2005;

где INTEGER – целочисленный тип. Типизированной константе можно присваивать только те значения, которые определяются данным типом.

Переменные

Переменная – это величина, которая, в отличие от константы, может менять свое значение в процессе выполнения программы.

Каждая переменная характеризуется:

- *именем переменной*, которое является идентификатором;
- *типом переменной*. *Тип* – это набор значений, которые могут принимать объекты программы, и совокупность операций, допустимых над этими значениями.

Тип констант автоматически распознается компилятором ТП без предварительного описания. Тип переменных должен быть описан перед тем, как с переменными будут выполняться какие-либо действия. Для описания переменных предназначено зарезервированное слово **Var**.

Формат: Var

<идентификатор> : <тип>;

Имя переменной является «оболочкой», которую можно заполнить различными значениями, в отличие от имени константы.

Типы данных

Понятие типа является одним из фундаментальных понятий любого языка программирования. Объекты (константы, переменные, функции, выражения), которыми оперирует программа, относятся к определенному типу. *Тип – это множество значений, которые могут принимать объекты программы, и совокупность операций, допустимых над этими*

значениями. Например, значения 1 и 2 относятся к целочисленному типу, их можно складывать, умножать и выполнять другие арифметические операции. Значения «монитор» и «Паскаль» носят лингвистический характер, они имеют, естественно, свой набор допустимых операций. В большинстве широкоупотребительных языков могут использоваться толь ко строго определенные, заранее известные типы. Pascal, наряду со стандартными типами, имеющимися в других языках высокого уровня, позволяет программисту образовывать собственные типы.

Все допустимые в языке Паскаль типы подразделяются на две большие группы: *скалярные* и *структурированные*.

Скалярные типы подразделяются на *стандартные* и *описанные пользователем*.

Стандартные скалярные типы

К *стандартным скалярным типам* относятся данные целочисленного, байтового, вещественного, литерного и булевского типов.

Целочисленный тип. Определяет все целые числа в диапазоне от -32768 до +32767. Для описания этого типа служит стандартный идентификатор INTEGER.

Формат: <идентификатор,...>: integer;

Пример

```
Var
  Sort, Nomer  : integer;
  A, B, C      : integer;
```

Переменная типа INTEGER занимает в памяти два байта. Пользователь должен сам следить, чтобы промежуточные результаты арифметических выражений не выходили из диапазона -32768 . . . +32767.

Байтовый тип. Аналогичен целочисленному, но охватывает более узкий диапазон значений от 0 до 255. Описывается стандартным идентификатором BYTE.

Формат: <идентификатор,...> : byte;

Пример

```
Var
  Gran       : byte;
  Min, Max   : byte;
```

Переменная типа Byte занимает в памяти 1 байт, т.е. по сравнению с целочисленным типом память экономится в два раза. Использование байтового типа целесообразно, если известно, что значение переменной не превысит 255. В арифметических и логических выражениях допустимо смешение типов byte и integer.

Вещественный тип данных. Включает все положительные значения с десятичной точкой от -1E-38 до +1E+38, соответствующие отрицательные значения и 0. Мантисса может содержать до 11-ти значащих цифр. Описывается стандартным идентификатором REAL.

Формат: <идентификатор,...> : real;

Пример

```
Var
  Res, Summa, Itog : real;
  X, Y             : real;
```

Переменная типа REAL занимает в памяти 6 байт.

Булевский тип данных описывается стандартным идентификатором BOOLEAN. Переменные и константы этого типа могут принимать только одно из двух значений: TRUE (истина) или FALSE (ложь). **Формат: <идентификатор,...> : boolean;**

Пример

```
Var
  Sel1, Sel2 : boolean;
  A,B,C,D    : boolean;
```

Выражения булевского типа занимают в памяти 1 байт и используются для управления порядком выполнения операторов программы.

Литерный (символьный) тип. Описывается стандартным идентификатором **CHAR**. Константы и переменные этого типа могут принимать одно из значений кодовой таблицы ASCII, символ берется в апострофы.

Формат: <идентификатор,...> : char;

Пример

```
Var
  Bukva, Znak, Simvol : char;
  Ch                  : char;
```

Переменные символьного типа занимают в памяти 1 байт. Использование данных типа char в арифметических выражениях запрещено. К литерным значениям могут применяться операции сравнения, результат при этом зависит от номера литерной переменной или константы в кодовой таблице.

Скалярные типы пользователя

Кроме стандартных типов данных, ПАСКАЛЬ поддерживает скалярные типы, определенные самим пользователем. К ним относятся *перечисляемый* и *интервальный* типы. Данные этих типов занимают в памяти 1 байт, поэтому любой пользовательский тип не может содержать

более 255 элементов. Их применение значительно улучшает наглядность программы, делает более легким поиск ошибок и экономит память..

Перечисляемый тип задается непосредственно перечислением всех значений, которые может принимать переменная данного типа. Отдельные значения указываются через запятую, а весь список заключается в круглые скобки.

Формат: Type

< имя типа > = (<значение1, значение2, ..., значение n>);

Var

<идентификатор, . . .> : < имя типа>;

Пример

```
Type
  Gaz = (C, O, N, F);
  Metall =(Fe, Co, Na, Cu, Zn);
Var
   G1, G2, G3 : Gaz;
   Met1, Met2 : Metall;
  Season : (Winter, Spring, Summer, Autumn);
```

В данном примере [1] приведены два явно описанных типа данных пользователя – Gaz и Metall. Определены их значения – обозначения некоторых газов и металлов периодической таблицы Менделеева. Переменные G1, G2, G3, Met1, Met2 могут принимать только одно из перечисленных значений. Попытка присвоить им любое другое значение вызовет программное прерывание. Третий тип перечисления анонимный (не имеет имени) и задается перечислением значений в разделе Var. Season является переменной этого типа и может принимать значения Winter, Spring, Summer, Autumn. Таким образом, может быть задан любой тип, но это не всегда приемлемо. Первый способ, безусловно, более понятен и больше соответствует характеру языка Pascal.

Интервальный тип позволяет задавать две константы, определяющие границы диапазона значений для данной переменной. Компилятор при каждой операции с переменной интервального типа генерирует подпрограммы проверки, определяющие, остается ли значение переменной внутри установленного для нее диапазона. Обе константы должны принадлежать одному из стандартных типов (причем тип REAL здесь недопустим). Значение первой константы должно быть обязательно меньше значения второй.

Формат: Type

<имя типа> = <константа1> . . <константа2>;

Var

<идентификатор> : < имя типа>;

Пример

```
Type
   Days = 1.. 31;
Var
   Work_d, Free_d : Days;
```

В этом примере переменные *Work_d, Free_d* имеют тип *Days* и могут принимать любые значения из диапазона 1 . . 31. Выход из диапазона вызывает программное прерывание.

Можно определить интервальный тип и более универсальным способом, задав границы диапазона не значениями констант, а их именами:

```
Const
  Min = 1; Max = 31;
  Type
    Days = Min . . Max;
  Var
      Work_d, Free_d : Days;
```

Структурированные типы данных

Структурированные типы базируются на скалярных типах и могут содержать их различные комбинации. Эти типы определяют упорядоченную совокупность скалярных элементов и характеризуются типом своих компонентов. В языке Turbo Pascal допускаются следующие структурированные типы данных:

строка – последовательность символов, заключенная в апострофы;

массив – структурированный тип данных, состоящий из фиксированного количества элементов одного и того же типа, доступ к которым осуществляется по индексу;

множество – набор выбранных по какому-либо признаку или группе признаков объектов, которые можно рассматривать как единое целое;

запись – совокупность фиксированного числа компонентов разного типа;

указатель – неограниченное множество, указывающее на однотипные элементы значений;

файл – последовательность компонентов одного типа и одной длины.

ЛЕКЦИЯ 7. ВЫРАЖЕНИЯ, ОПЕРАТОРЫ ЯЗЫКА TURBO PASCAL

Арифметические выражения и правила их записи

Выражение задает порядок выполнения действий над элементами данных и состоит из операндов, круглых скобок и знаков операций. В качестве операндов выступают константы, переменные и обращения к функциям. Арифметическое выражение порождает целое или действительное значение и представляет собой набор операндов, соединенных знаками арифметических операций.

Арифметические операции выполняют арифметические действия в выражениях над значениями операндов типа integer, byte, real. Основные арифметические операции языка Pascal представлены в таблице 1.

Таблица 2. Арифметические операции языка Turbo Pascal

*	умножение	
/	деление	
+	сложение	
-	вычитание	
DIV	операция целочисленного деления	Эти операции используются только с данными целочисленного типа
MOD	остаток от целочисленного деления	
AND	арифметическое И	
OR	арифметическое ИЛИ	

Операции **DIV** и **MOD** выполняются следующим образом:

36 **div** 5 = 7 - выделяет целую часть полученного частного;

36 **mod** 5 = 1 - выделяет остаток от деления нацело.

В арифметических выражениях могут использоваться стандартные функции, поддерживаемые языком Pascal (см. таблицу 2).

Таблица 3. Стандартные функции языка Turbo Pascal

Математическая запись	Запись на Турбо Паскале	Примечание
Sin X	Sin(X)	X
Cos X	Cos(X)	X
Ln X	Ln(X)	X>0
$\sqrt{X}$	Sqrt(X)	X>=0
e^X	Exp(X)	
X^2	Sqr(X)	
Arctan X	Arctan(X)	-Pi/2<=X<=+Pi/2
|X|	Abs(X)	$X=\begin{cases} X, если X \geq 0 \\ -X, если X < 0 \end{cases}$

Аргумент X может быть целого, байтового или вещественного типа. При решении некоторых задач необходимы следующие функции:

- получение целой части X:
  Trunc(X) – результат имеет целочисленный тип;
  Int(X) – результат имеет вещественный тип;
- получение дробной части X:
  Frac(X) – результат вещественного типа;
- округление X до целого:
  Round(X) – результата имеет вещественный тип;
- получение значения случайного числа из диапазона 0..1:
  Random(X) – результат имеет целочисленный тип;
- **Pred(X)** – возвращает элемент, предшествующий X в списке значений типа. Тип результата совпадает с типом параметра;
- **Succ (X)** – возвращает значение, следующее за X в списке значений типа. Тип результата совпадает с типом параметра.

Особый интерес представляет функция **ODD**, служащая для определения четности числа. Ее аргументом может быть переменная только целого типа.

ODD(X) – возвращает значение булевского типа , равное **True**, если X нечетное и **False**, если X – четное.

Пример:

Выражение	*Результат*
ODD(3)	*True*
ODD(4)	*False*
ODD(2+7-1)	*False.*

Если в выражениях имеются нестандартные функции, они должны быть выражены через стандартные.

$X^k \rightarrow e^{k\ln(x)} \rightarrow \exp(k*\ln(x))$;

$a^x \rightarrow e^{x\ln(a)} \rightarrow \exp(x*\ln(a))$;

$\sqrt[7]{2+X} \rightarrow (2+X)^{1/7} \rightarrow e^{1/7\ln(2+x)} \rightarrow \exp(1/7*\ln(2+x))$;

$\log_a(X) \rightarrow \ln(x)/\ln(a) \rightarrow \ln(x)/\ln(a)$.

Операторы языка Turbo Pascal

Основная часть программы на языке Паскаль представляет собой последовательность операторов.

Оператор *– это конструкция языка, которая определяет действия, выполняемые над данными, или порядок выполнения этих действий.* Операторы выполняются последовательно в том порядке, в котором они записаны в тексте программы. Разделителем операторов служит точка с запятой. Все операторы языка Турбо Паскаль разделяются на три группы:

1. *Простые операторы.*
2. *Операторы ввода-вывода*, которые на самом деле являются процедурами, так как в них можно указывать параметры.
3. *Структурированные* (сложные) операторы, в состав которых включаются другие операторы по строго определенным правилам.

Простые операторы

Операторы, не содержащие в себе никаких других операторов, называются простыми. К ним относятся операторы присваивания, безусловного перехода, вызова процедуры и пустой оператор.

Оператор присваивания (:=) предписывает выполнить выражение, заданное в его правой части, и присвоить результат переменной, идентификатор которой расположен в левой части. Переменная и выражение должны иметь один и тот же тип. Исключение представляет случай, когда переменная имеет вещественный тип, а выражение – целочисленный. Допустимо присваивание любых типов данных, кроме файловых.

Формат:

<идентификатор>:=<выражение>;

Пример

```
Sort:=1;
Cena:=15.23;
Nazv:='Model N986';
Res:=Sin(A)+Cos(B);
```

Оператор безусловного перехода GOTO. Означает «перейти к» и применяется в случаях, когда после выполнения некоторого оператора надо выполнить не следующий по порядку, а какой-нибудь другой, отмеченный меткой оператор. Метка может содержать как цифровые, так и буквенные символы [1].

Формат:

GOTO <метка>;

Пример

```
...
Label Metka1, Metka2;
...
Metka1:
GOTO Metka2;
Metka2:
GOTO Metka1;
...
```

При записи оператора **GOTO** необходимо помнить следующее.

Метка, на которую передается управление, должна быть описана в разделе описания меток того блока процедуры, функции, основной программы, в котором эта метка используется. Областью действия метки является тот блок, в котором она описана. Попытка выйти за пределы блока вызывает программное прерывание.

Обычно оператор **GOTO** применяется для преждевременного выхода из цикла или обработке ошибок. Но частое использование оператора **GOTO** усложняет понимание логики программы.

Оператор вызова процедуры. Служит для активизации предварительно определенной пользователем или стандартной процедуры.

Формат:

<имя процедуры> {(список параметров)};

Пример

```
Program Prim;
Procedure V1;
Begin
. . .                    {тело процедуры V1}
end;
Procedure V2;
Begin
. . .                    {тело процедуры V2}
end;
Begin
V1;                      {вызов для выполнения процедуры V1}
V2;                      {вызов для выполнения процедуры V1}
End.
```

Пустой оператор не содержит никаких символов и не выполняет никаких действий. Он может быть расположен в любом месте программы, где синтаксис языка допускает наличие оператора. Как и все другие операторы, пустой оператор может быть помечен меткой. Чаще всего пустой оператор используется для организации выхода из середины программы или составного оператора.

Пример

```
Begin
  Goto Metka;            {переход в конец блока}
. . .
  Metka;                 {пустой оператор помечен меткой}
End.
```

Операторы ввода-вывода

Решение самой простой задачи на ЭВМ не обходится без операций ввода-вывода информации. Ввод данных – это передача информации от внешнего носителя в оперативную память для обработки. Вывод – обратный процесс, когда данные передаются после обработки из оперативной памяти на внешний носитель.

Для выполнения операций ввода-вывода служат четыре оператора: **Read, Readln, Write, Writeln.**

Для ввода данных с клавиатуры используется оператор чтения **Read**.

Формат:

Read(список ввода);

где в списке вводы могут фигурировать только переменные допустимых типов данных.

Пример

```
Var
  I : integer;
  J : integer;
  K : char;
Begin
  read(I, J, K);
```

Оператор чтения **Readln** аналогичен оператору **Read**, единственное отличие состоит в том, что после считывания последнего в списке значения для одного оператора **Readln** данные для следующего оператора **Readln** будут считываться с начала новой строки.

Оператор записи **Write** производит вывод числовых данных, символов, строк, булевских значений.

Формат:

Write(список вывода);

Где список вывода представляет собой набор переменных и выражений.

Например *Write(y1, y2, sin(3), yn)* – выражения типа integer, byte, real, char, boolean и т.д.

Оператор вывода **Write** может использоваться в разных форматах.

Для переменных целого и символьного типов используется следующий формат: **Write(X:3,C:7),** где цифра после «:» определяет, сколько места на экране будет выделено для значения той или иной переменной.

Для переменных вещественного типа используется обычно следующий формат: **Write(X:3:1,C:7:4),** где первая цифра означает общее число позиций на экране, выделенное для данной переменной, а вторая цифра – число позиций (из общего числа) после десятичной точки. Для того, чтобы хорошо и понятно оформить вывод результирующей информации на экран, иногда требуется вывести текстовую информацию, поясняющую результат.

Пример

```
Write('Сумма ряда = ', S:7:3, 'при X=',X:4:2);
```

Оператор записи **Writeln** аналогичен оператору **Write**, но после вывода последнего в списке значения происходит перевод курсора к началу следующей строки. Оператор **Writeln** без параметров вызывает перевод строки (пропуск пустой строки).

Знание рассмотренных операторов дает возможность программирования алгоритмов линейной структуры. Но для того, чтобы правильно составить программу на языке Turbo Pascal, необходимо знать общие правила написания программ, соблюдение которых обязательно.

ЛЕКЦИЯ 8. СТРУКТУРА ПРОГРАММЫ НА ЯЗЫКЕ TURBO PASCAL

Программа реализует алгоритм решения задачи. Она объединяет последовательность действий, выполняемых над определенными данными с помощью определенных операций для реализации определенной цели. Основные характеристики программы: точность полученного результата, время выполнения и объем требуемой памяти. Хорошо написанная программа должна соответствовать принципам структурного программирования:

- не использовать сложных методов там, где можно обойтись простыми;
- минимально использовать оператор goto, так как от нарушает логику программы;
- для программирования разветвляющихся вычислительных процессов использовать оператор if – then – else или оператор case;
- для программирования циклических вычислительных процессов использовать операторы for, repeat или while. Организация цикла с помощью goto не отвечает принципам структурного программирования;
- большие программы следует разбивать на логически завершенные сегменты (процедуры, функции);
- имена констант, переменных, процедур, функций должны нести смысловую нагрузку, то есть по возможности отражать свое назначение.

Синтаксически программа на языке **Turbo Pascal** состоит из двух частей: *заголовка*, который не является обязательным, и *блока*, который может содержать в себе другие блоки. Блок состоит из двух частей: раздела описаний и раздела операторов. Первая часть может отсутствовать, без второй блок не имеет смысла. Блок, который не входит ни в какой другой блок, называется *глобальным*. Если в глобальном блоке находятся другие блоки, они называются *локальными*. Глобальный блок – это основная программа, он должен присутствовать в любом случае. Локальные блоки – это процедуры и функции, их присутствие необязательно.

В начале программы находится *заголовок*, который в общем случае состоит из зарезервированного слова *Program*, имени программы и параметров, с помощью которых программа взаимодействует с операционной системой. Заголовок программы несет чисто смысловую нагрузку и может отсутствовать.

Формат заголовка программы:

Program <имя программы>;

где <имя программы> является идентификатором.

Например, **Program Lin;**

После заголовка следует программный блок, состоящий в общем случае из 6-ти разделов: описания меток, описания констант, определения типов данных, описания переменных, описания процедур и функций, операторов.

Любой раздел, кроме раздела операторов, может отсутствовать. Разделы описаний могут встречаться в программе любое количество раз и следовать в любом порядке. Главное, чтобы все описания объектов программы были сделаны до того, как они будут использованы.

Раздел описания меток

Перед любым оператором языка Pascal можно поставить метку, что позволяет выполнить прямой переход на этот оператор с помощью оператора перехода goto из любого места программы. Метка состоит из имени и следующего за ним двоеточия. Именем может служить идентификатор или цифра.

Перед употреблением метка должна быть описана.

Формат : Label < имя, . . . >;

Пример:

Label M1, M2, 4, Blok2;

Далее в тексте программы выполняется обращение к меткам, обычно когда используется оператор **Goto**.

Например,

```
Goto 20
-------------
20:  Y:=2*SIN(X);
```

Здесь 20 – это метка оператора.

Использование оператора **Goto** является нежелательным, так как он смазывает логику и структуру программы.

Раздел описания констант

В этом разделе производится присваивание идентификаторам констант постоянных значений.

Формат: Const < идентификатор > = < значение>;

Пример

```
Const R1=22;
      St=3.25;
      R2:Integer=4;
```

После того как константа определена, ей нельзя присвоить какое-либо другое значение. Ряд констант (Pi, True, False, Maxint) определен стандартно, к ним можно обращаться без предварительного описания.

Раздел описания типов данных

Этот раздел включается в программу тогда, когда программист использует в программе свои собственные типы. Стандартные типы (real,

integer, boolean, char, byte) не требуют описания, в отличие от типов, образованных пользователем. Синтаксис языка Pascal не требует обязательного определения идентификатора типа, так как тип всегда можно задать перечислением в разделе описания переменных.

Формат: Type < имя типа > = < значения типа >;

Пример

Type

T1=1..25;

T2=Array[1..10] of real;

Раздел описания переменных

Каждая встречающаяся в программе переменная должна быть описана. Раздел описания переменных включает в себя описание всех переменных, встречаемых в программе.

Формат: Var <идентификатор, . . .> : < тип >;

Пример:

Var

A,B,X : real;

Z,Y : integer;

Слова **Label,Const, Type и Var** являются зарезервированными и обязательно должны присутствовать в начале соответствующих разделов описаний.

Раздел описания процедур и функций

В разделе описания процедур и функций должны быть приведены тексты тех процедур и функций, которые разработаны самим программистом и используются в данной программе.

Стандартные процедуры и функции в программе не описываются, но следует знать, в каких библиотечных модулях они находятся. Имя этого модуля должно указываться сразу за заголовком программы в разделе **Uses**.

Например:

Program Lin;

Uses Crt, Printer;

Здесь в модуле **Crt** содержатся стандартные процедуры управления экраном, а модуль **Printer** служит для управления работой принтера.

Характерной чертой языка Turbo Pascal является наличие большого количества стандартных процедур и функций (общее название – подпрограммы). Примером стандартных функций являются Sin(x), Cos(x), **Ln(x)** и т.д.; все они находятся в библиотечном модуле **System**. При обращении к функциям аргумент указывается в скобках.

Рассмотрим несколько процедур, которые затем будут использоваться при написании программ:

1. **CLRSCR** – очистка экрана. Курсор перемещается в левый верхний угол экрана.
2. **GOTOXY(N1, N2)** – перемещение курсора в заданную точку экрана. Здесь **N1, N2** – параметры:
   - N1 – номер столбца на экране дисплея, целое, лежит в диапазоне от 1 до 80;
   - N2 – номер строки на экране дисплея, целое, лежит в диапазоне от 1 до 25;

   Заметим, что левый верхний угол экрана имеет координаты (1,1), а правый нижний угол экрана – (80,25). Например, **GOTOXY(20,10)** – перемещение курсора в 10 строку и 20 столбец.
3. **DELAY (T)** – задержка выполнения программы на T микросекунд. Например, чтобы сделать задержку на 2 сек, следует записать **DELAY** (2000).

Все рассмотренные процедуры находятся в библиотечном модуле **CRT**. В общем случае, для того, чтобы использовать ту или иную процедуру или функцию, необходимо знать:

- имя процедуры (функции);
- количество, порядок следования и назначения параметров процедуры (функции). Значения параметров записываются через запятую;
- в каком библиотечном модуле находится используемая процедура (функция).

Раздел операторов

В программе на языке Pascal раздел операторов является основным, так как именно в нем с предварительно описанными переменными, константами, значениями функций выполняются действия, позволяющие получить результат, ради которого создавалась программа. Здесь описывается алгоритм решения задачи. Этот раздел начинается со слова *Begin* и заканчивается словом *End*, между которыми располагаются строки операторов, описывающих, в каком порядке выполняются действия над данными, чтобы получить требуемый результат. В конце каждого оператора ставится «;». Завершает раздел зарезервированное слово *End* и точка.

Пример

```
Begin
  < оператор; >
  . . .
  < оператор; >
End.
```

Операторы выполняются строго последовательно в том порядке, в котором они записаны в тексте программы в соответствии с синтаксисом и правилами пунктуации.

В качестве примера составим и разберем простейшую программу на примере линейного алгоритма.

Пример

Вычислить значение функции

$$Y=\sqrt[5]{SIN(X+2)}*LOG_3(2*X)+e^{3*x+1};$$

значение X задать самостоятельно.

Вычисление значения Y проведем в несколько этапов. С целью упрощения процесса отладки разобьем функцию на промежуточные величины

$$Y1=\sqrt[5]{|\ SIN(X+2)\ |};\quad Y2=LOG_3(2*X);\quad Y3=e^{3*x+1};\quad Y=Y1*Y2+Y3.$$

Текст программы представлен ниже.

```
Program Lin;
   Uses CRT;      (*Раздел подключения стандартных модулей*)
   Var x,y,y1,y2,y3:real;        (*Раздел описания переменных*)
     Begin
     (*Ввод исходных данных*)
      Writeln('Введите x');
      Readln(x);
     (*Вычисление значения функции Y*)
       y1:= exp(1/5*ln(sin(x+2)));
       y2:=ln(2*x)/ln(3);
       y3:=exp(3*x+1);
       y:=y1*y2+y3;
        (*Вывод результатов*)
         Writeln('x=',x:4:2,'  y=',y:5:2);
         Delay(5000);  (*Задержка результата на экране*)
End.
```

Результат:

x= 0.55 y=14.23

ЛЕКЦИЯ 9. СТРУКТУРНЫЕ ОПЕРАТОРЫ ЯЗЫКА TURBO PASCAL. УСЛОВНЫЕ ОПЕРАТОРЫ IF И CASE. РАЗВЕТВЛЯЮЩИЕСЯ ПРОГРАММЫ С ПРОСТЫМ И СЛОЖНЫМ ЛОГИЧЕСКИМ УСЛОВИЕМ

Структурные операторы представляют собой структуры, построенные из других операторов по строго определенным правилам. Все структурные операторы подразделяются на три группы: составные, условные и повтора.

Составные операторы

Составной оператор представляет собой группу из произвольного числа операторов, отделенных друг от друга точкой с запятой и ограниченную операторными скобками **Begin**... **End**.

Формат: **begin**

<оператор;>

<оператор;>

. . .

<оператор>

end;

Пример

```
Begin
 A := Sin (Pi * x);
 B := exp (2 * x);
 Rez:= A + B;
 Writeln ( Rez: 6:3)
end;
```

Составной оператор может находиться в любой части программы и воспринимается как единое целое. Обычно используется при организации разветвлений или циклов.

Условные операторы

В разветвляющихся вычислительных процессах отдельные этапы вычислений (операторы) выполняются не всегда в одном и том же порядке. В зависимости от некоторых условий, проверяемых по ходу вычислений, выбираются для исполнения различные их последовательности. Поэтому в программе должно содержаться указание о том, в каком случае надо выбирать для исполнения тот или иной оператор. Это указание естественно формулировать в виде отношения, например условия «x > y». Если это условие справедливо (принимает значение TRUE), то выполняется соответствующий этому условию оператор, в противном случае, если условие принимает значение FALSE, выполняется другой оператор.

Для задания подобного рода разветвляющихся вычислительных процессов служат условные (выбирающие) операторы, которые относятся к числу производных операторов. Такой оператор обеспечивает выполнение или невыполнение некоторого оператора, группы операторов или блока в зависимости от заданных условий. Pascal допускает использование двух условных операторов IF и CASE.

Оператор условия IF является одним из самых популярных средств, изменяющих естественный порядок выполнения операторов программы. Он может принимать одну из следующих форм:

полная форма:

If <условие> Then <оператор1>
Else <оператор2>;

Если условие истинно, выполняется оператор 1, в противном случае оператор2;

неполная форма:

If <условие> Then <оператор>;

Если условие истинно, выполняется оператор, стоящий за **Then**, в противном случае – выполняется оператор, стоящий сразу за **If**.

Блок-схемы полной и неполной условной конструкции рассматривались в лекции 4, рис.7.

Пример

```
If X>0 Then Writeln('X больше 0')
       Else Writeln ('X меньше или равно 0');
```

Перед **Else** ";" не ставится. Если после **Then** или **Else** необходимо выполнить не один, а несколько операторов, они оформляются как составной оператор, т.е. должны быть взяты в операторные скобки Begin...End.

Например,

```
If X<=A Then
     Begin
       Y:= A*X;
       Z:= Ln(X)+Y;
     End;
```

Один оператор **IF** может входить в состав другого оператора **IF**. Такие конструкции называют вложенными.

Рассмотрим использование операторов условия для программирования разветвляющихся вычислительных процессов.

Пример 1

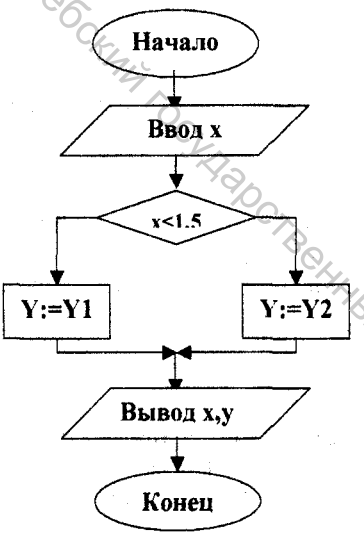
Составить блок-схему и программу на языке Turbo Pascal вычисления значения функции Y.

$$y=\begin{cases} LOG_2|X-4.2|, если X<1.5 \\ SIN(2*X) если X\geq 1.5 \end{cases}$$

Здесь имеет место простое логическое условие, так как разветвляющийся вычислительный процесс имеет только две ветви.

Первую ветвь условно обозначим Y1, вторую – Y2.

Блок-схема вычисления значения функции Y:



Программа вычисления значения функции Y:

```
Program razv1;
 uses crt;
 Var x,y:real;
 Begin
  Writeln('Введите x');
  Readln(x);
   If x < 1.5 Then Y:=Ln(Abs(x-4.2)) Ln(2)
              Else Y:=Sin(2*x);
  Writeln('x=',x:4:1, ' y=',y:5:2);
  Delay(5000);
 End.
```

Результат:

x= 1.2 y= 1.58

x= 3.4 y= 0.49

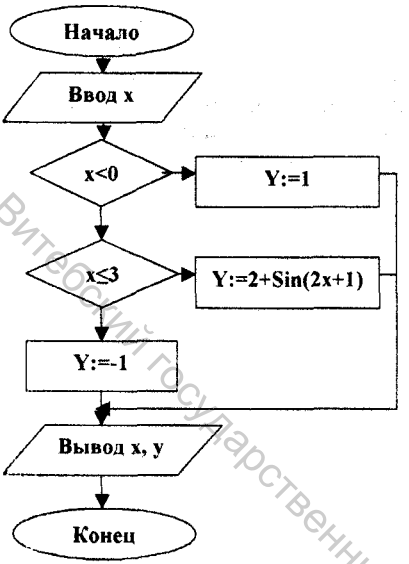
Рис. 9. Блок-схема схема вычисления значения функции Y (пример 1)

Пример 2

Составить блок-схему и программу на языке Turbo Pascal вычисления значения функции Y.

Здесь имеет место сложное логическое условие, так как разветвляющийся вычислительный процесс имеет более двух ветвей (в данном случае – три ветви).

$$y=\begin{cases} 1, если X<0 \\ 2+SIN(2*X+1), если 0\leq X<3 \\ -1, если X\geq 3 \end{cases}$$



x = 2.5 y =-1.92
x = 3.5 y= -1.00

Рис.10. Блок-схема вычисления значения функции Y (пример 2).

Программа вычисления значения функции Y:

```
Program razv1;
 uses crt;
 Var x,y:real;
 Begin
  Writeln('Введите x');
  Readln(x);
   If x < 0 Then Y:=1
            Else If x < 3
            Then Y:=2*Sin(2*x)
            Else Y:=-1;
  Writeln('x=',x:4:1, ' y=',y:5:2);
  Delay(5000);
End.
```

Результат:
x =-2.0 y = 1.00

Оператор выбора CASE является обобщением оператора **IF** и позволяет сделать выбор из произвольного числа имеющихся вариантов. Он состоит из выражения, называемого селектором, и списка параметров, каждому из которых предшествует список констант выбора. Как и в операторе **IF**, здесь может присутствовать слово **ELSE**, имеющее тот же смысл.

Формат:

```
Case <выражение-селектор> of
        <список1>: <оператор1;>
        <список2>: <оператор2;>
        . . .
        <списокN>: <операторN>
              else <оператор>
end;
```

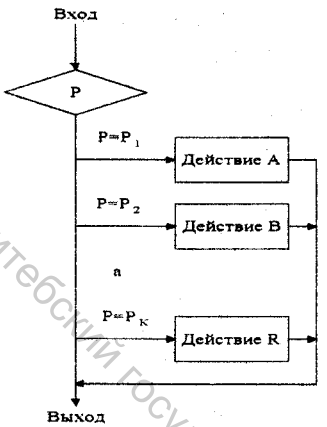


Рис.11. Оператор Case

Эта структура обеспечивает выбор одного из ряда возможных действий, в зависимости от значения, которое принимает выражение (условие) P. Количество значений, принимаемых P, должно быть конечным.

Пример

- Селектор целочисленного типа:

```
Case I of
      1:  Z:=A+B;
      2:  Z:=A+10;
      3:  Z:=A+100;
end;
```

- Селектор перечисляемого типа:

```
Case Season of
      Winter:   Writeln('WINTER');
      Spring:   Writeln('SPRING');
      Summer: Writeln('SUMMER');
      Autumn: Writeln('AUTUMN')
End;
```

ЛЕКЦИЯ 10. ОПЕРАТОР ПОВТОРА FOR. ЦИКЛИЧЕСКИЕ ПРОГРАММЫ

Операторы повтора используются при организации циклов.

Цикл *– это многократное использование одного и того же вычисления с изменяющимися исходными параметрами.*

Конкретные программные реализации циклов называются циклическими программами. Различают два типа циклов:

- циклы со счетчиком, когда число повторений цикла известно заранее;
- итерационные циклы, когда число повторений цикла не известно заранее. В этом случае задается условие, при котором цикл должен закончиться или, наоборот, продолжаться.

Можно выделить основные типы задач, которые сводятся к программированию циклических процессов:

1. Вычисление суммы, произведения элементов конечного ряда. При этом суть циклического процесса заключается в накоплении суммы S или произведения P по зависимости

   S:=S+U,

   где S, стоящее справа от «:=», есть сумма элементов конечного ряда,
   U – значение текущего элемента конкретного ряда,
   S, стоящее слева от «:=», последующее (определяемое) значение суммы конечного ряда.

   Или P:=P*U,

   где P, стоящее справа от «:=», есть произведение элементов конечного ряда,
   U – значение текущего элемента конкретного ряда,
   P, стоящее слева от «:=», последующее (определяемое) значение.

2. Определение суммы (произведения) элементов бесконечного ряда. Условием окончания цикла является дополнительное логическое условие $|S_n - S_{n-1}| < eps$, где eps – заданная точность вычислений. Сумма при этом вычисляется традиционно: S:=S+U.

3. Табулирование функций: получение значений функции при дискретном изменении значения аргумента.

X	Xmin	Xmin+ΔX	. . .	Xmax
Y	Y1	Y2	. . .	Ymax

Для реализации циклов в Турбо Паскале используются три оператора:

For – который используется только в тех циклах, число повторений которых известно заранее (циклах со счетчиком);

Repeat и **While** – которые используются как в итерационных циклах, так и в циклах со счетчиком.

Оператор повтора For

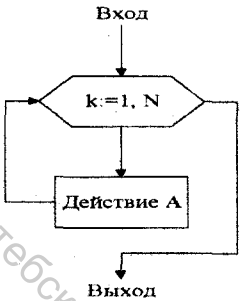


Рис.12. Оператор For

Эта конструкция обеспечивает повторение действия А заданное число раз. При каждом повторении цикла величина К, называемая параметром цикла, увеличивается (или уменьшается) на 1. Выход из цикла происходит тогда, когда величина К превзойдет значение N, которое определяет число повторений цикла.

Оператор **For** состоит из заголовка и тела цикла. Он может быть представлен в двух форматах:

For <параметр цикла>:=N1 to N2 do <оператор>;

For <параметр цикла>:=N1 downto N2 do <оператор>;

Здесь

For ... do – заголовок цикла;

<оператор> - тело цикла. Оператор может быть простым или составным. В последнем случае используются операторные скобки.

Параметр цикла, его начальное и конечное значения должны принадлежать к одному и тому же типу данных. При этом допустим любой скалярный тип, кроме вещественного.

N1 и **N2** – соответственно начальное и конечное значения параметра цикла.

Если используются типы integer, byte и интервальный, то параметр цикла в процессе выполнения циклической части программы принимает следующие значения:

в случае 1: N1, N1+1, N1+2,... N2;

в случае 2: N1, N1-1, N1-2,...N2.

Отсюда следует, что в первом случае (при For ... to) N1 должно быть меньше или равно N2, во втором (при For ... downto) N1 больше или равно N2. То есть шаг изменения параметра цикла в операторе **For** в любом случае равен единице. Однако, это не является большим недостатком, так как любой шаг можно задать при использовании операторов **While** и **Repeat**.

Пример

```
For i:=4 to 7 do
    Write(i:2);
```

Результат:

4 5 6 7.

```
For i:=7 downto 4 do
    Write(i:2);
```

Результат:

7 6 5 4.

После нормального завершения оператора For значение параметра цикла равно конечному значению. Если оператор For не выполнялся,

значение параметра цикла не определено. В теле оператора For могут содержаться другие операторы For. Это позволяет строить циклы, содержащие внутренние циклы. Такие внутренние циклы называются вложенными.

Пример

Вычислить сумму элементов конечного ряда (при X = 0;3):

$$S = \sum_{n=2}^{8} (-1)^{n+1} \cdot \frac{(x+1)^{2n}}{(n+1)!} \cdot Sin(n \cdot x)$$

Задачи такого типа сводятся к вычислению значения каждого элемента суммы при разных значениях параметра суммирования (в данном случае n). Каждое слагаемое суммы зависит от значения переменной X и параметра суммирования (параметра цикла) n, определяющего место этого слагаемого в сумме.

Обычно формула общего члена суммы принадлежит к одному из следующих трех типов:

а) $\frac{X^n}{n!}$; $(-1)^n \frac{X^{2n+1}}{(2n+1)!}$; $\frac{X^{2n}}{(2n)!}$;

б) $\frac{Cosnx}{n}$; $\frac{Sin(2n-1)x}{2n-1}$; $\frac{Cos2nx}{4n^2-1}$;

в) $\frac{X^{4n+1}}{4n+1}$; $(-1)^n \frac{Cosnx}{n^2}$; $\frac{n^2+1}{n!}\left(\frac{X}{2}\right)^n$

В случае а) для вычисления суммы целесообразно использовать рекуррентные соотношения, т.е. выражать последующий член суммы через предыдущий. Это позволяет существенно сократить объем вычислительной работы. Кроме этого, вычисления члена суммы по общей формуле в ряде случаев невозможно (например, из-за наличия n!).

В случае б) применение рекуррентных соотношений нецелесообразно. Вычисления будут наиболее эффективными, если каждый член суммы вычислять по общей формуле.

В случае в) член суммы целесообразно представить в виде двух сомножителей, один из которых вычисляется по рекуррентному соотношению, а другой непосредственно. Например, в нашем случае, если

$U_n = (-1)^{n+1} \cdot \frac{(x+1)^{2n}}{(n+1)!} \cdot Sin(nx)$, то полагаем

$C_n = (-1)^{n+1}, P_n = (x+1)^{2n}$, $F_n = (n+1)!$ и вычисляем рекуррентно:

$C_n = -C_{n-1}$, $P_n = P_{n-1} \cdot (x+1)^2$; $F_n = F_{n-1} \cdot (n+1)$,

а $Sin(n \cdot x)$ вычисляем непосредственно. Таким образом, для члена суммы имеем

$$U_n = C_n \cdot P_n / F_n \cdot Sin(nx)$$

Алгоритм решения задач суммирования при значениях параметра суммирования, изменяющегося в некотором диапазоне (n = 2,8) с заданным шагом (в нашем случае шаг равен 1), сводится к циклу, когда при каждом прохождении цикла номер члена суммы изменяется на 1, а сумма изменяется на ее n-ый член, т.е.

$$S_n = S_{n-1} + U_n, \qquad (1)$$

где *Sn* и *Sn-1* – сумма n и n-1 членов.

Формула (1) применяется многократно при n = 2, 3, 4, …, 8. При этом S1 = 0 (не просуммировано ни одного элемента), S – искомое значение суммы 8 элементов.

При составлении блок-схемы и программы нет необходимости использовать переменные с индексами Sn, Un и т.п., поскольку одновременно в вычислениях участвуют лишь 2 значения: Sn и Sn-1, Cn и Cn-1, Pn и Pn-1, Fn и Fn-1. Конечным результатом является сумма всех элементов, и значения суммы одного, двух, трех и т.д. элементов запоминать не требуется. В таком случае можно использовать простые переменные, значения которых будут изменяться каждый раз при вычислении очередного элемента суммы. При этом в формулах

c:=-c,

$p:=p(x+1)^2$,

F:=F(n+1) (блок 4) и

S:=S+U (блок 5)

переменные C, P, F и S справа и слева имеют разные значения: справа предыдущие (Cn-1, Pn-1, Fn-1, Sn-1), слева текущие (Cn, Pn, Fn, Sn).

Блок-схема алгоритма решения задачи представлена на рисунке 13.

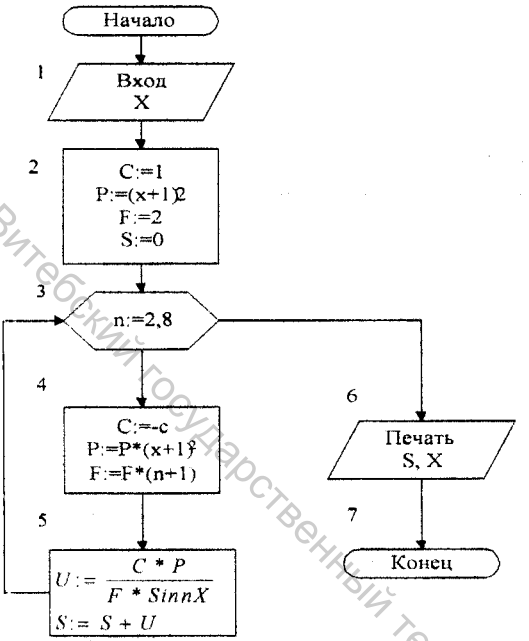


Рис.13. Блок-схема решения задачи 2.

Ниже приводится программа вычисления суммы элементов ряда.

```
Program Cikl1;
(*Вычисление суммы заданного числа элементов ряда*)
Uses(*указание библиотечных модулей*)
   Crt;
Var (*раздел описания переменных*)
   X,P,U,S:Real;
     C,F,N:Integer;
Begin (*начало раздела операторов*)
Clrscr;
Writeln('Введите x');
Readln(x);
C:=1;
P:=Sqr(X+1);
F:=2;
S:=0;
(*начало цикла по N*)
For N:=2 to 8 do
Begin
C:=-C;
```

```
P:=P*Sqr(X+1);
F:=F*(N+1);
U:=C*P/F*Sin(N*X);
S:=S+U;
End;
(*вывод результата на дисплей*)
Writeln('При X=',X:4:2,'  сумма', N:2,' элементов=',S:7:5);
Readln;
End.
(*конец программы*)
```

ЛЕКЦИЯ 11. ОПЕРАТОРЫ ПОВТОРА WHILE И REPEAT. ЦИКЛИЧЕСКИЕ ПРОГРАММЫ

Оператор повтора Repeat

Чаще всего используется для итерационных циклов.

Рис. 14. Оператор Repeat.

Первой особенностью итерационного цикла является то, что заранее невозможно определить число необходимых итераций или приближений. Поэтому при программировании циклов в таких задачах применяется не цикл со счетчиком (или с параметром), а цикл с постусловием, называемый иначе итеративным, или итерационным.

Второй особенностью итерационного цикла является то, что тело цикла должно располагаться до проверки условия окончания (или повторения) цикла. В языке Паскаль для организации цикла с постусловием имеется специальный оператор ***Repeat Until***, проверяющий условие окончания цикла.

Формат:

Repeat
<оператор;>
<оператор;>
. . .
<оператор >
Until <условие>;

Условие – это выражение булевского типа, при котором цикл заканчивается. Операторы, заключенные между словами **Repeat** и **Until**, являются телом цикла. Вначале выполняется тело цикла, затем проверяется условие выхода из цикла. Если результат выражения **FALSE**, тело цикла выполняется еще раз, если результат **TRUE** – происходит выход из цикла.

Оператор **Repeat** имеет три характерные особенности:

- выполняется по крайней мере хотя бы один раз;
- тело цикла выполняется пока условие ложно;
- в теле может находиться произвольное число операторов без операторных скобок begin ... end.

По крайней мере, один из операторов тела цикла должен влиять на значение условия, иначе цикл будет выполняться бесконечно.

Пример

```
I:=4;
Repeat
  Write(I:2);
```

```
i:=i+1
Until I>7;
```

Результат:

5, 6, 7, 8

Так как условие проверяется после выполнения тела цикла, оператор *Repeat* выполняется по крайней мере один раз. Поэтому важно четко проследить правильность вхождения в цикл и выхода из него. При неправильной организации выхода может также возникнуть ошибка переполнения, так как наращивание какой-либо величины в теле цикла может продолжаться до бесконечности.

Пример

Составить с объяснениями блок-схему и программу вычисления суммы элементов ряда с заданной точностью.

$$S = \sum_{n=1}^{\infty} \frac{Sin^{n}(x+1)}{n!} \cdot \left(\frac{x}{2}\right)^{3}$$

Условием окончания цикла считать $|Sn-Sn-1| < \varepsilon$,

где: Sn – значение суммы элементов ряда;

Sn-1 – значение суммы n-1 элементов ряда;

$\varepsilon=10^{5}$ – заданная точность вычислений.

Очевидно, что формула в условии данной задачи относится к типу в) (см. стр. 61).

Поэтому, полагая Pn=Pn-1*Sin(x+1), a Fn=Fn-1.n,

получим значение n-го элемента ряда

Un=Pn/Fn*(x/2)

и суммы

Sn=S(n-1)+Un.

Блок–схема этой задачи может быть представлена следующим образом:

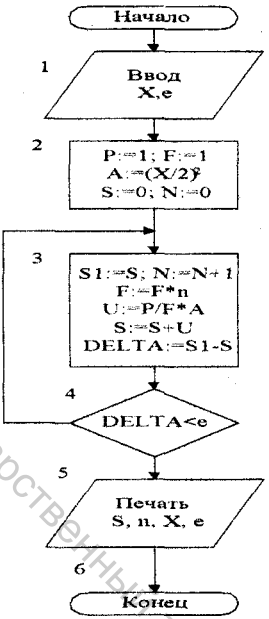


Рис.15. Блок-схема вычисления суммы S.

Пояснения к блок-схеме:

БЛОК 1 – ввод значений x, ε.

БЛОК 2 – задание начальных значений для организации цикла, (P=1, F:=1) таких, что при N = 1, Pn=Sin(x+1), а Fn=1!. Очевидно, что нет необходимости повторять вычисление $(x/2)^3$ на каждом шаге цикла. Поэтому можно вычислить его заранее и поместить в переменную A. Так как вычисление суммы идет по стандартному алгоритму Sn=S(n-1)+Un, то начальное значение суммы обнуляется. Чтобы число просчитанных элементов ряда соответствовало значению N, N также принимается равным 0.

БЛОК 3 – образует тело цикла, здесь вычисляются значения P и F по рекуррентным соотношениям, приведенным выше, значения n-го элемента ряда U и суммы S. Для сохранения предыдущего (n-1) значения суммы использована переменная S1. Переменная DELTA сохраняет значение разности сумм на n-ом и (n-1) шаге.

БЛОК 4 – проверка условия окончания цикла. Если оно истинно, т.е.

$$|S1-S| < \varepsilon,$$

то осуществляется переход на блок 5, если ложно – на блок 4 и повторение тела цикла.

БЛОК 5 – печать результатов – S, N, X.

БЛОК 6 – конец программы.

Программа вычисления суммы ряда с заданной точностью приведена ниже.

```
Program Cikl2;
(*вычисление суммы элементов ряда с заданной точностью*)
Uses(*указание библиотечных модулей*)
   Crt;
Var (*раздел описания переменных*)
  X,EPS,P,A,U,S,S1,Delta:Real;
                F,N:Integer;
Begin (*начало раздела операторов*)
Clrscr;
Writeln('Введите X');
Readln(X);
Writeln('Введите точность вычислений EPS');
Readln(EPS);
A:=exp(3*ln(x/2)); P:=1; F:=1; N:=0;S:=0;
(*начало итерационного цикла*)
Repeat
S1:=S; N:=N+1; P:=P*Sin(X+1); F:=F*N;
U:=P/F*A;  S:=S+U;Delta:=Abs(S-S1);
(*проверка условия окончения цикла*)
Until Delta<EPS;
(*вывод результата на дисплей*)
Writeln('При x=',x:4:2,'  сумма', n:2,' элементов=',S:7:5);
Readln;
End.
(*конец программы*)
```

Оператор повтора While

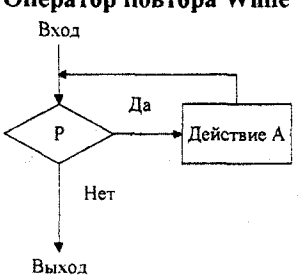


Рис.16. Оператор While

Эта структура обеспечивает многократное повторение некоторого действия A (называемого телом цикла). Действие повторяется до тех пор, пока выполняется условие P (условие повторения цикла). Если после очередного повторения действия A условие P не выполняется, происходит выход из цикла. Повтор реализуется оператором цикла ***WHILE*** (пока), имеющим следующий формат:

While <выражение> Do <оператор>;

Этот оператор аналогичен оператору Repeat, но проверка условия выполнения тела цикла производится в самом начале оператора. Любой цикл, построенный с помощью For или Repeat, может быть построен и с помощью **While.** Здесь выражение представляет собой условие, при котором цикл повторяется. Перед каждым выполнением тела цикла вычисляется значение выражения условия. Если результат равен True, тело цикла выполняется и снова вычисляется выражение условия. Если результат равен False, то есть условие не выполняется, происходит выход из цикла и переход к первому после While оператору. Если перед первым выполнением цикла значение выражения было False, тело цикла вообще не выполняется и происходит переход на следующий оператор.

Цикл, который был описан для оператора REPEAT, в случае с WHILE будет иметь вид

```
i:=4;
While i<=7 do
  Begin
  Write(i:2);
  i:=i+1
End;
```

Как и в опреаторе Repeat, программист сам должен позаботиться об изменении переменных, определяющих условие выхода, иначе цикл получится бесконечным. Выйти из цикла можно с помощью оператора goto, минуя вычисление выражения условия.

Для начинающих программистов использование оператора While вызывает гораздо меньшее число ошибок, чем применение оператора Repeat, поэтому при прочих равных условиях лучше пользоваться оператором While.

Таким образом, можно выделить следующие отличия **While** от **Repeat**:

1. Условие ставится в начале цикла.
2. Это условие, при котором цикл повторяется.
3. Необходимость использования операторных скобок *Begin... End*, если в цикле выполняется более одного оператора.
4. Выход из цикла будет произведен тогда, когда условие, стоящее после **While,** не будет выполняться.

Пример

Разработать схему и программу на языке Турбо-Паскаль задачи табулирования функции

$$Y = \begin{cases} \sqrt{T + \ln^2 x} & x < 1 \\ T \cdot Sin^2(x+1) & 2.3 \geq x \geq 1 \\ \sqrt[3]{e^{2x+T}} & x > 2.3 \end{cases}$$

заданной на отрезке [A, B], шаг приращения аргумента DX.

Для контрольного примера взять следующие значения исходных данных:

- границы интервала A = -0,6; B = 3,4;
- коэффициент T = 1,2 ;
- шаг изменения параметра цикла Dx = 0,25.

Алгоритм решения этой задачи включает в себя следующие этапы:

- ввод исходных данных, в качестве которых выступают A, B – границы интервала табулирования функции, DX – приращение аргумента и T;
- задание начального значения аргумента функции X = A;
- организация цикла с помощью структуры повтор (цикл-пока), которая реализуется в программе оператором ***While***.

Тело цикла состоит из следующих действий:

- вычисление значения функции, которая реализуется управляющей структурой развилка;
- печать значений аргумента и функции;
- приращение значения аргумента x на величину dx;
- после выполнения тела цикла управление передается на начало цикла, где проверяется условие повторения цикла. Если оно выполняется, тело цикла повторяется, в противном случае происходит выход из цикла.

Схема алгоритма решения задачи приведена на рис.17.

В схеме необходимо обратить внимание на следующее:

- блок 4 реализует проверку условия повторения цикла и будет реализовываться оператором ***WHILE***;
- блоки 5 и 7 проверяют, в каком диапазоне значений находится величина аргумента X (для выбора соответствующей формулы для вычисления Y) и реализуется в программе оператором ***IF***.

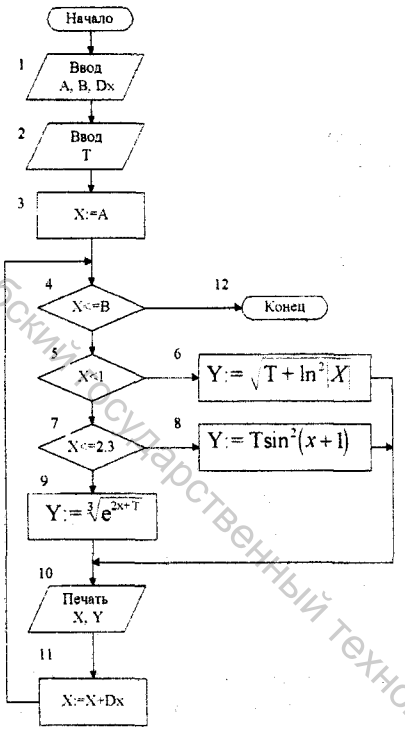


Рис.17. Блок-схема табулирования функции Y.

Текст программы решения задачи на языке Turbo Pascal приведен ниже.

```
Program Tab;
Uses crt;(*указание библиотечных модулей*)
 Var a,b,dx,t:real;
       x,y:real;
Begin
(*начало раздела операторов*)
Clrscr;
Writeln('введите границы интервала [a,b],шаг dx-');
Readln(a,b,dx);
Writeln('Введите Значение t');
Readln(t);
(*вывод заголовка таблицы значений функции на дисплей*)
Write('Таблица Значений Функции');
Writeln;
```

```
(*начальное значение a*)
X:=a;
While x<=b do
 Begin
(*вычисление и вывод y в зависимости от условия*)
 If x<1 Then y:=sqrt(t+sqr(ln(abs(x))))
      Else If x<=2.3 then y:=t*sqr(sin(x+1))
      Else y:= exp(1/3*(2*x+t));
(*печать таблицы значений функции на дисплей*)
 Writeln (X:5:2,' ':5,Y:5:2);
(*получение нового значения аргумента*)
 x:=x+dx;
 End;
(*конец цикла*)
 Readln;
 End.
```

Рассмотренные операторы дают возможность описывать алгоритмы решения различных задач, которые представляют собой комбинацию линейных, разветвляющихся и циклических участков.

Линейные участки, которые не содержат разветвлений и циклов, реализуются обычно с помощью оператора присваивания, операторов ввода-вывода.

Разветвления в программе реализуются с помощью оператора If, а циклические участки – с помощью одного из операторов цикла – For, While и Repeat.

Пример

Для каждого из десяти задаваемых вещественных чисел вычислить значение Y по правилу

$$Y = \begin{cases} a\cos 2x, & x < 1 \\ a\sqrt{4 - \lg x}, & x \geq 1 \end{cases}$$

Программу можно представить в довольно наглядной и легко понимаемой форме:

```
Program Primer;
Uses crt;
Const n=10;
      a=2.5;
Var i: integer;
  x,y:real;
  Begin
   ClrScr;
   Writeln('Введите 10 чисел - значений x');
   For i:=1 to n do
    Begin
```

```
  read(x);
  if x< 1 then y:=a*Cos(2*x)
          else y:=a*sqrt(4-ln(x) ln(10));
  Writeln('x=',x:4:1,'   y=',y:5:2);
  End;
 Delay(5000);
End.
```

Результат:

Введите 10 чисел - значений x

2

x= 2.0 y= 4.81

5

x= 5.0 y= 4.54

3

x= 3.0 y= 4.69

1

x= 1.0 y= 5.00

2.5

x= 2.5 y= 4.74

3.7

x= 3.7 y= 4.63

5.2

x= 5.2 y= 4.53

6.1

x= 6.1 y= 4.48

8.4

x= 8.4 y= 4.38

2.6

x= 2.6 y= 4.73

ЛЕКЦИЯ 12. ПОНЯТИЕ МОДУЛЬНОЙ ПРОГРАММЫ. ПРОЦЕДУРЫ В ЯЗЫКЕ TURBO PASCAL

В практике программирования довольно часто встречаются ситуации, когда одну и ту же группу операторов, реализующих определенную цель, требуется повторить без изменений в нескольких других местах программы. Чтобы избавить программиста от столь нерационального занятия, была предложена концепция подпрограмм, впервые описанная М.Уилксом в 1957 году. Она получила широкое распространение во всех языках программирования.

Подпрограммой называется именованная, логически законченная группа операторов языка, которую можно вызвать для выполнения по имени любое количество раз из различных мест программы. В языке Паскаль для организации подпрограмм используются *процедуры и функции*.

Все процедуры и функции языка ТП разделяются на две группы: *встроенные* и *описанные* пользователем.

Встроенные (стандартные) процедуры и функции являются частью языка и могут вызываться по имени без предварительного определения в разделе описаний.

Процедуры и функции пользователя организовываются самим программистом в соответствии с синтаксисом языка и представляют собой локальный блок. Предварительное описание процедур и функций пользователя обязательно.

Паскаль представляет программисту достаточно широкий набор встроенных процедур и функций, которые реализуют наиболее часто встречающиеся при написании программ действия. Различают *9 основных групп* встроенных процедур и функций. Для обозначения типов данных аргументов используем сокращения: IBR – целочисленный, байтовый, вещественный тип; S – любой из скалярных типов, кроме вещественного.

1. *Арифметические* (Abs(IBR), Arctan(IBR), Cos(IBR), Sin(IBR), Exp(IBR)…).
2. *Скалярные* (Odd(I), Pred(S), Succ(S)).
3. *Преобразования типов* (Round(R) – возвращает значение R, округленное до ближайшего целого числа. Результат имеет целочисленный тип; Trunc(R)- возвращает ближайшее целое число, меньшее или равное R, если R>=0, и большее или равное R, если R<0. Результат имеет целочисленный тип.
4. *Управление строками на экране* (ClrScr, DelLine, InsLine).
5. *Специальные функции* (Delay(I), Exit, Halt).
6. *Функции обработки строк.*
7. *Функции и процедуры обработки файлов.*
8. *Функции управления памятью.*
9. *Графические процедуры.*

Процедуры пользователя

Использование подпрограмм, организованных пользователем, обеспечивает возможность:

- организовать работу нескольких программистов над одной программой;
- проводить отладку отдельных блоков и только потом всей программы в целом;
- значительно сэкономить память, т.к. многократно используемый участок заносится в память только один раз;
- упростить внесение изменений в программу.

Язык Pascal позволяет использовать следующие виды размещения подпрограмм:

- головная программа и подпрограммы распложены в одном программном модуле;
- подпрограммы расположены в отдельных файлах и включаются в основной файл посредством директивы компилятора;
- подпрограммы организуются как оверлейные структуры и поочередно загружаются в одно и тоже место памяти;
- подпрограммы пишутся на машинном коде и затем включаются в программу;
- подпрограммы располагаются во внешней библиотеке и вызываются из основной программы.

Процедура пользователя – это именованная группа операторов, реализующая определенную часть общей задачи. Процедура вызывается по имени из любой части головной программы.

Описание процедуры включает заголовок и тело процедуры. Заголовок состоит из зарезервированного слова **Procedure**, идентификатора (имени) процедуры и необязательного списка формальных параметров, заключенного в круглые скобки. Для каждого параметра из списка указывается его тип.

Формат:

Procedure <имя>[(формальные параметры); - заголовок
<разделы описаний>
Begin
<раздел операторов> тело процедуры
End;

Пример заголовка:

Procedure Sort(A:integer; B:real);
Procedure Sum;

Имя процедуры – это идентификатор, записанный по правилам записи идентификаторов и уникальный в пределах программы. Тело процедуры – это локальный блок, по структуре аналогичный программе.

Синтаксическая диаграмма процедуры представлена на рис. 18:

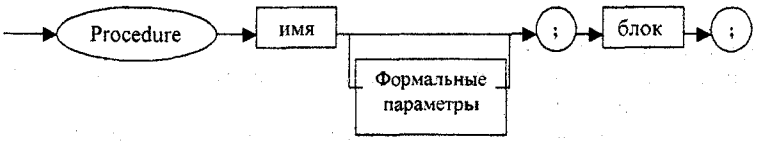


Рис.18. Синтаксическая диаграмма процедуры.

Для обращения к процедуре используется оператор вызова процедуры. Он состоит из идентификатора (имени) процедуры и списка фактических параметров, отделенных друг от друга запятыми и заключенных в круглые скобки.

Например:

SORT(A1,B1); - фактические параметры – значения переменных;

SORT(14,25); - фактические параметры – непосредственные значения;

SUM; - фактические параметры не указаны, т.к. в вызываемой процедуре нет формальных параметров.

Между фактическими параметрами в операторе вызова процедуры и формальными параметрами в заголовке описания процедуры устанавливается взаимнооднозначное соответствие в результате их перебора слева направо. Количество и тип формальных параметров равны количеству и типу фактических параметров.

Если процедура возвращает в программу какие-то значения, соответствующие переменные должны быть описаны как параметры-переменные с использованием слова VAR.

Например:

Procedure One(X:integer; var Y:real);

пар-знач парам-перем.

Формальные параметры делятся на два типа: параметры-значения, которые передаются в подпрограмму, и параметры-переменные, которые передаются из подпрограммы в вызывающую часть.

```
Procedure One(X:integer; var Y:real);
    Begin
        Y:=X*Sin(X);
    End;
```

Вызов: **One(2.5, Z);**

Если используется несколько процедур в программе и одна из них вызывает другую, то вызываемая процедура должна быть описана обязательно до вызывающей.

Различные особенности описания процедур и способы их использования рассмотрим на примере следующей задачи [3].

По задаваемым вещественным значениям x и y вычислить

U = max(x+y), V = max(0.5, U).

Составим программу, реализующую решение без использования подпрограммы-процедуры:

```
Program Max_a;
  var x,y,u,v:real;
 begin
  writeln('x,y');
  readln(x,y);
  If x+y>x*y  Then U:=x+y Else U:=x*y;
  If 0.5>U Then V:=0.5 Else V:=U;
  Writeln('U=',U:3:1,'  V=',V:3:1);
 end.
```

В программе фигурируют два условных оператора, выполняющих одну и ту же частичную задачу: поиск максимального из двух значений. Поэтому алгоритм решения этой задачи целесообразно объявить процедурой.

- Процедура без параметров.

Чтобы подчеркнуть сходство двух условных операторов, запишем программу иначе:

```
a:=x+y; b:=x*y;
If a>b Then S:=a Else S:=b;
U:=S;
a:=0.5; b:=U;
If a>b Then S:=a Else S:=b;
V:=S;
```

Таким образом, условные операторы полностью совпадают друг с другом, то есть один и тот же условный оператор присутствует в программе дважды. Объявим его процедурой.

```
Procedure Max_1A;
  Begin
    If a>b Then S:=a Else S:=b;
  End;}
```

Для вызова этой процедуры в нужном месте программы достаточно записать оператор процедуры MAX.

Программа с этой процедурой выглядит следующим образом:

```
Program Max_B;
 var x,y,u,v,a,b,s:real;
 Procedure Max_1A;
  Begin
```

```
 If a>b Then S:=a Else S:= b;
 End;
Begin
 Writeln('x,y');
 Readln(x,y);
 a:=x+y; b:=x*y;
 Max_1A; U:=S;
 a:=0.5; b:=u;
 Max_1A; V:=S;
 Writeln('U=',u:3:1,'  V=',v:3:1);
End.
```

- Процедура с параметрами.

Предыдущий вариант не очень удобен, так как назначение процедуры зафиксировано слишком жестко. Следовательно, исходными данными для нее могут быть только значения переменных a и b. Поэтому при каждом обращении к процедуре этим переменным нужно присваивать те значения, из которых должно быть выбрано наибольшее. Этого можно избежать, оформив процедуру с параметром, и передавать значения a и b в качестве параметров-значений, а S – в качестве параметра-переменной.

```
Program Max_C;
 var x,y,u,v:real;
 Procedure Max_2A(a,b:real;var s:real);
  Begin
   If a>b Then S:=a Else S:= b;
  End;
Begin
 Writeln('x,y');
 Readln(x,y);
 Max_2A(x+y,x*y,U);
 Max_2A(0.5,U,V);
 Writeln('U=',u:3:1,'  V=',v:3:1);
End.
```

ЛЕКЦИЯ 13. ФУНКЦИИ ПОЛЬЗОВАТЕЛЯ В ЯЗЫКЕ TURBO PASCAL. ЛОКАЛЬНЫЕ И ГЛОБАЛЬНЫЕ ПЕРЕМЕННЫЕ

Хотя набор встроенных функций языка Pascal достаточно широк, он вряд ли может удовлетворить требованиям каждого программиста. Поэтому пользователю предоставлена возможность самому реализовывать нужные ему алгоритмы в виде функций и обращаться к ним из программ по мере необходимости. Функция, определенная пользователем, состоит из заголовка и тела функции. Заголовок состоит из ключевого слова **Function,** имени функции, необязательного списка формальных параметров, заключенного в круглые скобки с указанием типа каждого параметра, и типа возвращаемого результата.

Формат:

Function <имя>{(формальные параметры)}:тип;

<разделы описаний>

Begin

<раздел операторов>

End;

Например:

Function One(X:integer):real;

Function Sum:real;

В разделе операторов должен находиться по крайней мере один оператор, присваивающий идентификатору функции значение. Если таких операторов несколько, то результатом работы функции будет значение последнего оператора присваивания.

Обращение к функции производится по имени, после которого в круглых скобках следует необязательный список фактических параметров.

Синтаксическая диаграмма функции представлена на рис. 19:

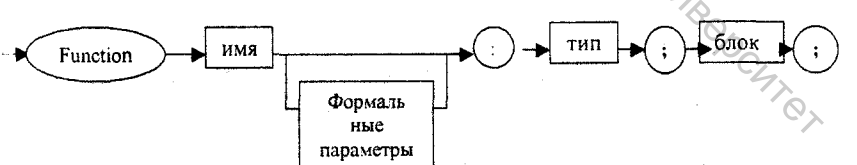


Рис. 19. Синтаксическая диаграмма функции

Имя функции – уникальный в пределах блока идентификатор. Раздел, относящийся к формальным параметрам для функции, аналогичен разделу описания формальных параметров для процедур. Возвращаемый результат может иметь любой скалярный тип.

Тело функции представляет собой локальный блок, по структуре аналогичный программе.

Особенности, присущие функциям

1. Обязательно задается тип возвращаемого результата.
2. В результате работы функции получается только один результат.
3. В разделе операторов блока должен присутствовать хотя бы один оператор, влияющий на значение функции.
4. Обращение к функции должно происходить только из какого-либо выражения. Y:=One(X); а не One(X).

Область действия переменных. Локальные и глобальные переменные

Программа на языке Pascal имеет модульную структуру и может состоять из ряда вложенных друг в друга блоков. Основная программа – это самый крупный блок, который не входит ни в какой другой. Те идентификаторы, которые описаны в головной программе, являются ***глобальными*** и могут использоваться во всех вложенных блоках (во всех процедурах и функциях). Идентификаторы, описанные в определенных модулях, называются ***локальными*** и могут быть использованы только в той части программы, где они описаны. Все параметры, используемые в подпрограммах пользователя, подчиняются определенным правилам.

1. Каждый идентификатор должен быть описан перед тем, как он будет использован.
2. Областью действия идентификатора является блок, в котором он описан.
3. Идентификаторы в блоке должны быть уникальными, т.е. не должны повторяться.
4. Один и тот же идентификатор может быть по разному определен в каждом отдельном блоке.
5. Если процедура или функция пользователя совпадает по имени со стандартной процедурой или функцией, то будет выполняться подпрограмма пользователя, а стандартная игнорироваться (Sin, Cos).

Параметры

Параметры, описанные в подпрограммах, могут иметь любой тип.

Следует прежде всего различать *формальные* и *фактические* параметры.

Параметры, описанные в заголовке подпрограммы при ее определении, называются *формальными*. Параметры, которые указываются в заголовке подпрограммы при обращении к ней, называются *фактическими*.

Когда параметры передаются по *значениям*, формальный параметр является переменной, локальной в блоке. Фактический параметр может быть любым выражением, переменной или идентификатором того же типа, что и соответствующий ему формальный параметр. Такие параметры называются *параметрами-значениями*. Их главная отличительная черта:

изменение формального параметра не влечет за собой изменения фактического. Если же любое изменение формального параметра влечет за собой изменение фактического, то такие параметры называются *параметрами-переменными*. Для их описания служит зарезервированное слово **Var**. Составим программу, реализующую решение рассмотренной выше задачи с использованием подпрограммы-функции.

Вычислить U = max(x+y), V = max(0.5, U).

- Функция без параметров:

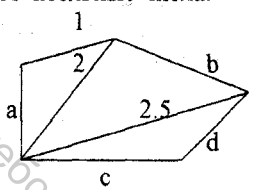
```
Program Max_D;
 Var x,y,u,v,a,b:real;
 Function Max:real;
  Var S:real;
  Begin
   If a>b Then S:=a Else S:= b;
   Max:=S;
  End;
 Begin
  Writeln('x,y');
  Readln(x,y);
  a:=x+y; b:=x*y;
  U:=Max;
  a:=0.5; b:=u;
  V:=Max;
  Writeln('U=',u:3:1,'  V=',v:3:1);
 End.
```

- Функция с параметрами:

```
Program max_e;
    Var x,y,u,v:real;
    Function Max(a,b:real):real;
     Var s:real;
     Begin
      If a>b Then s:=a Else s:= b;
      Max:=s;
     End;
    Begin
    Writeln('x,y');
    Readln(x,y);
    U:=Max(x+y,x*y);
    V:=Max(0.5,U);
    Writeln('U=',u:3:1,'  V=',v:3:1);
    End.
```

Примеры описания подпрограмм при решении задач

Вычислить площадь пятиугольника, изображенного на рисунке, где a,b,c,d – действительные числа:



$$p = \frac{x+y+z}{2}$$

$$S = \sqrt{p(p-x)(p-y)(p-z)}$$

а) с использованием подпрограммы-процедуры:

```
Program SP;
  Var a,b,c,d,S1,S2,S3,S:real;
(************************************)
  Procedure GERON(x,y,z:real;Var SG:real);
    Var P:real;
        Begin
          P:=(x+y+z)/2;
          SG:=Sqrt(p*(p-x)*(p-y)*(p-z));
        End;
(************************************)
Begin
  ClrScr;
  Writeln('Ввведите стороны a,b,c,d');
  Readln(a,b,c,d);
  GERON(1,a,2,s1);
  GERON(2,b,2.5,S2);
  GERON(c,2.5,d,S3);
  S:=S1+S2+S3;
  Writeln('Площадь S=',S:6:3);
  Readln;
  End.
```

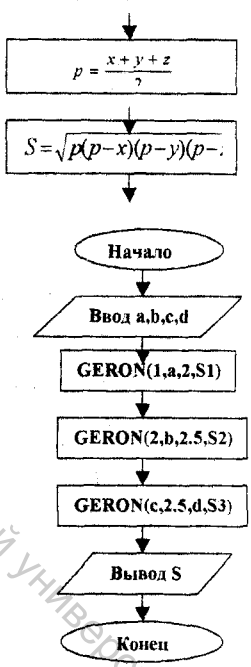


Рис. 20. Блок-схема с ППП

Здесь x,y,z,SG – формальные параметры. Причем x,y,z – параметры-значения, SG – параметр-переменная.

A,b,c,d, S - фактические параметры.

P – локальная переменная;

б) с использованием подпрограммы-функции:
Блок-схема решения задачи:

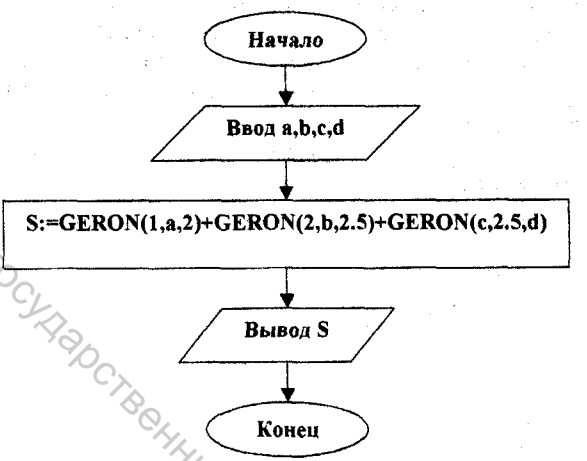


Рис. 21. Блок-схема с ППФ

Программа на языке Turbo Pascal:

```
Program SF;
 Var a,b,c,d,S:real;
(************************************************)
 Function GERON(x,y,z:real):real;
   Var P, SG:real;
       Begin
         P:=(x+y+z)/2;
         SG:=Sqrt(p*(p-x)*(p-y)*(p-z)c);
         GERON:=SG;
       End;
(************************************************)
    Begin
       ClrScr;
       Writeln('введите стороны a,b,c,d');
       Readln(a,b,c,d);
       S:=GERON(1,a,2)+GERON(2,b,2.5)+GERON(c,2.5,d);
       Writeln('Площадь  S=',S:6:3);
       Readln;
       End.
```

Рекурсивные подпрограммы

Во многих случаях оптимизация алгоритма решения задачи требует вызова для выполнения подпрограммы из раздела операторов той же самой подпрограммы, т.е. подпрограмма вызывает сама себя. Такой способ вызова называется *рекурсией*. Рекурсия полезна в тех случаях, когда основную задачу можно разделить на подзадачи, имеющие ту же структуру, что и первоначальная задача. Программы, реализующие рекурсию, называются *рекурсивными*.

Прямая рекурсия заключается в том, что подпрограмма в своем блоке использует саму себя.

Пример

```
Program A;
    Procedure B;
        Begin
        Writeln('Пример');
         B;
        End;
    Begin
    B;
    End.
```

Здесь программа А содержит процедуру В, которая будет вызывать сама себя и печатать слово «Пример» бесконечное число раз.

Классическим примером использования рекурсии является вычисление факториала целого положительного числа [3]. Проиллюстрируем на программе.

```
Program DemoRecurs;
 Var N:integer;
 Function Fact(A:integer):integer;
  Begin
  If A=0 Then Fact:=1 Else Fact:=A*Fact(A-1);
  End;
  Begin
   While True do
   Begin
   Writeln('Введите число N');
   readln(N);
   Case N of
   0   : Writeln('0! = 0');
   1..9: Writeln(N,'!=',Fact(N));
    10: Halt
    Else Writeln('Число должно быть в диапазоне 0..9');
   End;
   End;
   End.
```

Здесь рекурсия используется, чтобы решить подзадачи, затем сложить результаты их решений и получить, таким образом, решения первоначальной задачи.

Часто встречается и другой вариант рекурсии, когда первая подпрограмма вызывает вторую, которая в момент вызова еще не определена. Такая ситуация называется *косвенной рекурсией*. Для реализации косвенной рекурсии используется так называемое предварительное описание процедур и функций.

ЛЕКЦИЯ 14. СТРУКТУРИРОВАННЫЕ ТИПЫ ДАННЫХ. ОДНОМЕРНЫЕ МАССИВЫ

На языке Паскаль можно обрабатывать не только отдельные переменные, но и их совокупности. Одной из таких совокупностей является массив.

Массив – *это структурированный тип данных, состоящий из фиксированного числа элементов, имеющих один и тот же тип.*

Тип элементов массива называется *базовым*. Число элементов массива фиксируется при описании и в процессе выполнения программы не меняется. Доступ к каждому отдельному элементу осуществляется путем индексирования элементов массива. Индексы представляют собой выражения любого скалярного типа, кроме вещественного. Тип индекса определяет границы изменений значений индекса.

Например, массив А состоит из элементов a_1, a_2, a_3, . . . a_n.

Здесь а – имя элемента массива,

n – его индекс, т.е. порядковый номер в массиве, по которому можно определить (найти) каждый элемент.

Для описания массива в языке Turbo Pascal предназначено словосочетание *array of* (массив из...).

Описание массива возможно двумя способами:

1. Используя раздел описания типов:

Type

<*имя типа*> = **array** [тип индекса] **of** <*тип компонент*>;

Var

<идентификатор> : <*имя типа*>;

Пример

Type

*Mas = **array**[1..100] **of** real;*

Var

M1,M2:Mas;

2. Используя только раздел описания переменных (Var) и без представления типа в разделе описания типов данных:

Var

<идентификатор> : **array** [тип индекса] **of** <тип компонент>;

Пример

Var

M1, M2: array[1..100] of real;

V1:array[1..50] of integer;

Если в качестве базового типа взят другой массив, образуется структура, которую принято называть многомерным массивом.

Пример

```
Type
  Vector = array [1..4] of integer;
  Massiv = array [1..4] of Vector;
Var
  M1:Massiv;
```

Ту же структуру можно получить, используя другую форму записи:

```
Var
 M1: array[1..4,1..4] of integer;
```

Таким образом, многомерный массив задается при помощи индексированных выражений, следующих друг за другом. Если индексное выражение одно, то массив называется одномерным, если два – двухмерным (матрица), если n – n-мерным.

Одномерные массивы в Паскале обычно используются для представления векторов, двухмерные – для представления матриц.

Пример: *Var Vector = array [1..5] of real ;* (одномерный массив, представляющий собой вектор из 5-ти элементов типа real);

Matr = array [1..4,1..3] of integer; (матрица из 4-х строк и 3-х столбцов с элементами типа integer).

Можно использовать предварительно определенные константы:

```
Const G1=4;G2=5;
Var
  Matr = array [1..G1,1..G2] of integer;
```

Элементы массива располагаются в памяти последовательно, по строкам. Для массива Matr:

Matr[1,1]	Matr[1,2]	Matr[1,3]
Matr[2,1]	Matr[2,2]	Matr[2,3]
Matr[3,1]	Matr[3,2]	Matr[3,3]
Matr[4,1]	Matr[4,2]	Matr[4,3]

При обращении к элементам массива на Паскале индекс указывается в квадратных скобках: A[i], B[i,j]. Синтаксическая диаграмма массивового типа представлена на рис. 22:

Рис. 22. Синтаксическая диаграмма массивового типа

При работе с массивами необходимо помнить следующее:

- массив – это структура данных, представляющая собой набор, совокупность элементов одного типа;
- при объявлении массива указывается количество элементов массива;
- доступ к элементу массива осуществляется путем указания индекса (номера) элемента массива.

Тип индекса может быть только скалярным. Наиболее часто в качестве типа индекса используется ограниченный тип [3], причем в большинстве случаев это ограниченный целый тип.

Так, например, массив из 100 компонент вещественного типа может быть задан

array [1..100] of real;

Ограниченный целый тип 1..100 определяет количество компонент – 100 и их упорядоченность – от 1-ой до 100-ой.

В большинстве задач нумерация компонент начинается от 1 и ограничивается положительным целым числом. Но это необязательно. Если, например, представить в виде массива численность г. Москвы [3] в отдельные годы, то нумерацию удобно начинать с 1147 – года основания Москвы и до 2003 , т.е. такой массив можно задать как

array[1147..2003] of integer;

а для Рима, основанного в 754 г. до н.э.

array[-754..-1] of integer;

здесь –1 –последний год до н.э.

Каждый из рассмотренных массивов в программе может быть задан двояко:

а) используя описание типа:

```
Type Gran=1..100;
     vector = array[Gran] of real;
     ch_M = array[1147..2003] of integer;
     ch_R = array[-754..-1] of integer;
Var a,b:vector;
    c:ch_M;
    d:ch_R;
    n:Gran;
```

Обращение к элементам массивов c и d : c[1147], c[1999] и т.п., то есть c[i], d[i].

Для обращения к элементам массивов a и b нужно ввести переменную типа Gran, например n. Обращение a[Gran] неверно, т.к. Gran является типом индекса, а не его именем;

б) непосредственно при описании переменной:

```
Var a,b:araay[1..100] of real;
    c:array[1147..2003] of integer;
    d:array[-754..-1] of integer;
```

Т.е. использование *ограниченного целого типа* для индексов дает достаточно широкие возможности для описания массивов.

Множество значений *перечисляемого типа* также образует *ограниченное перенумерованное множество.* Следовательно, *перечисляемый тип* также может быть использован в качестве типа индексов [3]. Значения перечисляемого типа упорядочены (порядок

задается порядком перечисления имен, являющихся значениями этого типа), и число их конечно (определяется количеством имен в этом типе).

Например, необходимо проанализировать среднемесячную температуру воздуха за год [3]. Можно в качестве индекса использовать ограниченный целый тип 1..12, но можно для наглядности ввести перечисляемый тип, состоящий из имен месяцев года:

Type *Month=(Jan, Feb, Mar, Apr, ... , Dec);*

тогда переменные, являющиеся векторами среднемесячных температур, можно определить следующим образом:

Var *t,r:array[Month] of real;*

Тогда частичные компоненты массива, обозначающие температуру каждого месяца, запишутся след. образом:

t[Yan], t[Feb], t[apr] и т.п.

Если ввести переменную *m:Month*, то можно обращаться *t[m]*, где m меняется от *Jan до Dec.*

В качестве типа индексов могут использоваться также стандартные типы Boolean и Char.

Например:

```
Type priznak= array[Boolean] of integer;
     kod_s = array[Char] of integer;
var  k:priznak;
     s:kod_s;
```

Частичные компоненты таких массивов выглядят следующим образом:
k[False], k[True];
s['d'], s['h'].

Операции над массивами

Для работы с массивом как с единым целым используется идентификатор массива без указания индекса в квадратных скобках. Если массивы описаны одинаково, то с ними можно проводить следующие операции [1]:

- *Проверка на равенство:*

A = B. Результатом является значение "истина", если каждый элемент массива A равен соответствующему элементу массива B.

- *Проверка на неравенство:*

A <> B. Результатом является значение «истина», если значение хотя бы одного элемента массива A не равно значению соответствующего элемента массива B.

- *Присваивание.*

A : = B Все значения элементов массива B присваиваются элементам массива A. Значения элементов массива B остаются без изменения.

Действия над элементами одномерного массива

После объявления массива каждый его элемент можно обработать, указав идентификатор массива и индекс элемента в квадратных скобках. Рассмотрим типичные ситуации, возникающие при работе с массивами. Для этого опишем четыре массива: два (A и B) из элементов целочисленного типа, два (C и B) из элементов вещественного типа.

```
Var A,B:array[1..5] of integer;
    C,D:array [1..10] of real;
```

Инициализация массива

Инициализация – это присваивание каждому элементу массива какого-либо (или одного и того же) значения. Например: A[1] := 0; A[2] := 0; . . . A[5] := 0. В данном случае индекс изменяется от 1 до 5. Но при большом количестве элементов такой способ инициализации нерационален. Гораздо удобнее получить такой же результат, используя оператор FOR:

```
For I:=1 to 5 do
A[i] : = 0;
```

Ввод-вывод элементов массива

Pascal не имеет средств ввода-вывода элементов массива сразу, поэтому ввод и вывод значений производится поэлементно. Чаще всего они вводятся с экрана с помощью оператора *read* или *readln* с использованием оператора цикла *for*:

```
For i : = 1 to 5 do
  Begin
    Writeln('введите', i:2, ' элемент массива');
    Readln (B[i]);
  End;
```

Вывод элементов массива производится аналогично, но с использованием оператора write или writeln:

```
For i ; = 1 to 5 do
Writeln( i:2, ' –й элемент массива=', B[i]:4:1);
```

Копирование элементов из одного массива в другой

B : = A; (B и A имеют одинаковый тип компонент)

A : = C ; (Неверно, т.к. C имеет тип real, а A – тип integer и массивы A и C разной длины);

Или используя цикл:

```
For i: = 1 to 5 do
 Begin
C[i] : = A[i];
B[i] : = D[i]; неверно, т.к. B – integer; D – real;
```

Поиск в массиве элементов, удовлетворяющих некоторым условиям

Пусть надо выяснить, сколько элементов в массиве С имеют нулевое значение.

```
K : = 0;
For i : = 1 to 10 do
 If C[i] = 0 then  k: = k + 1;
```

После выполнения цикла переменная К будет содержать количество элементов с нулевым значением.

Сумма (произведение) элементов массива:

```
Sum : = 0;                    Pr: =1;
For i : = 1 to 10 do          For i : = 1 to 10 do
Sum : = Sum + C[i];           Pr: = Pr*C[i];
```

Перестановка элементов массива.

Осуществляется с использованием вспомогательной переменной того же типа, что и базовый тип массивов. Например, требуется поменять местами первый и четвертый элементы массива А:

```
Per : = A[1];
A[1] : = A[4];
A[4] : =  Per;
```

Поиск максимального (минимального) элемента в массиве.

```
Max : = A[1];
   K : = 1;
   For i : = 2 to 5 do
       If  A[i] > Max
       Then  Begin
                Max : = A[i];
                 K : = i;
                End;
```

Для получения более полного представления об основных действиях при обработке массивов разберем примеры.

Пример 1

Вычислить % дней в месяце, когда температура опускалась ниже 0.

Введем обозначения:
all_d – всего дней в месяце (примем значение all_d =31);
cold_d – количество холодных дней;
temp – массив температур за месяц;
pr – процент дней с отрицательной температурой.

Текст программы на языке Turbo Pascal приведен ниже:

```
program cold;
uses crt;
const all_d=31;                          {описание констант и переменных}
type days=array[1..all_d] of real;
var temp:days;
      pr:real;
  cold_d:integer;
       i:integer;
  begin
  writeln('vv temp');
    for i:=1 to 31 do                    {ввод значений температуры за
      read(temp[i]);                      каждый из 31   дня месяца}
  cold_d:=0;
  for i:=1 to all_d do
  if temp[i]< 0 then cold_d:=cold_d+1;       {подсчет в % количества дней
  pr:=cold_d/all_d*100;                       с температурой ниже 0^0}
  writeln('cold_d=',cold_d:2,' pr=',pr:5:2);  {вывод результата}
  delay(10000);
  end.
```

Достаточно часто возникает необходимость использования массивов и при решении экономических задач. Разберем типичную задачу

Пример 2

Определить средний тарифный разряд рабочих и работ. Сделать вывод, обеспечен ли участок квалифицированными кадрами.

Заданы:

N – число разрядов рабочих N=5;
M – число разрядов работ M=3;
R_i – разряды рабочих {2, 3, 4, 5, 6};
P_i – число рабочих по i-му разряду {2, 4, 5, 3, 3};
S_i – разряды работ {2, 4, 5};
T_i – трудоемкость работ по разрядам {120, 180, 210}.

Постановка задачи:

Средний тарифный разряд рабочих определится по формуле:

$$RSR = \frac{\sum_{I=1}^{N}(R_I * P_I)}{\sum_{I=1}^{N} P_I} > 1$$

Средний тарифный разряд работ определится по формуле:

$$SSR = \frac{\sum_{I=1}^{m}(S_I * T_I)}{\sum_{I=1}^{M} T_I} > 1$$

Если RSR < SSR, то квалификация рабочих не соответствует квалификации работ.

Текст программы на языке Turbo Pascal приведен ниже:

```
Program Uchastok;
 Uses CRT;
 Type mas=array[1..10] of real;
 Var P,R,S,T:mas;
    RSR,SSR:real;
    n,m:integer;
 (*****************************************)
 Procedure Vvod(dl:integer;var v:mas);
  Var i:integer;
  Begin
  For i:=1 to dl Do
   Begin
    Writeln('Введите',I:2,'элемети массива');
    Readln(v[i]);
   End;
  End;
 (*****************************************)
 Procedure Vyvod(dl:integer; v :mas);
  Var i:integer;
  Begin
  For i:=1 to dl Do
   Begin
    Write(v[i]:3:1,'':2);
   End;
  End;
(*****************************************)
 Function SUM(dl:integer;v:mas):real;
  Var i:integer;
    S:real;
```

```
Begin
S:=0;
For i:=1 to dl Do
 S:=S+v[i];
SUM:=S;
End;
(*****************************************)
Function SUM_PR(dl:integer;v,w:mas):real;
Var i:integer;
  S:real;
Begin
S:=0;
For i:=1 to dl Do
 S:=S+v[i]*w[i];
SUM_PR:=S;
End;
(*****************************************)
Begin
ClrScr;
Writeln('Введите количество разрядов работ - m');
Readln(m);
Writeln('Введите количество разрядов рабочих - n');
Readln(n);
Writeln('Введите массив разрядов рабочих R');
Vvod(n,R);
Writeln('Введите количество рабочих каждого разряда P');
Vvod(n,P);
Writeln('Введите массив разрядов работ S');
Vvod(m,S);
Writeln('Введите трудоемкость работ по разрядам T');
Vvod(m,T);
RSR:=SUM_PR(n,R,P)/SUM(n,P);
SSR:=SUM_PR(m,S,T)/SUM(m,T);
ClrScr;
If RSR>=SSR Then Writeln('Участок обеспечен квалифицированными
рабочими')
        Else Writeln('Участок не обеспечен квалифицированными
рабочими');
Writeln('Средний разряд рабочих RSR=',RSR:4:1);
Writeln('Средний разряд работ SSR=',SSR:4:1);
Writeln('Массив разрядов рабочих');
Vyvod(n,R);
Writeln;
Writeln('Количество рабочих каждого разряда');
```

```
Vyvod(n,P);
Writeln;
Writeln('Массив разрядов работ');
Vyvod(m,S);
Writeln;
Writeln('Трудоемкость работы по каждому разряду');
Vyvod(m,T);
Readln;
End.
```

Р е з у л ь т а т ы:

```
Участок обеспечен квалифицированными рабочими
Средний разряд рабочих RSR=4,1
Средний разряд работ SSR=3,9
Массив разрядов рабочих
2 3 4 5 6
Количество рабочих каждого разряда
2 4 5 3 3
Массив разрядов работ
2 4 5
Трудоемкость работы по каждому разряду
120 180 210
```

Пояснения к программе.

Для ввода и вывода массивов используются процедуры *Vvod* и *Vyvod*, написанные для формального массива *v* длиной *dl*. Используя эти процедуры вводятся и выводятся массивы исходных данных (R, P, S, T).

В подпрограмме-функции *SUM* описывается вычисление суммы элементов формального массива *v* длиной *dl*.

В подпрограмме-функции *SUM_PR* описывается вычисление суммы произведений формальных массивов *v* и *w* длиной *dl*.

Далее в программе эти функции используются для вычисления по заданным формулам значений среднего разряда рабочих *RSR* и среднего разряда работ *SSR* для фактических параметров – значений исходных данных.

ЛЕКЦИЯ 15. СОРТИРОВКА МАССИВОВ

Сортировка – это процесс упорядочивания набора данных одного типа по возрастанию или по убыванию значения какого-либо признака.

Массив (например массив A_i) является отсортированным (упорядоченным) по возрастанию, если для всех i из интервала [1.., n+1] выполняется условие $a_i \le a_{i-1}$.

Различают следующие виды сортировок:

1. *Обменные сортировки*

Суть обменных сортировок заключается в следующем. Сначала последовательно просматриваются числа $a_1, \ldots, a_n$ и находится наименьшее I, такое, что $a_i > a_{i+1}$. Затем эти a_i и a_{i+1} меняются местами и просмотр возобновляется с элемента a_{i-1} и так далее. Тем самым наибольшее число передвигается на последнее место. Следующие просмотры начинаются опять сначала, уменьшая на единицу количество просматриваемых элементов. Массив будет упорядочен после просмотра, в котором участвовали только первый и второй элементы. В пределах этого вида можно выделить следующие методы сортировок:

- Метод «пузырька». Массив последовательно просматривается несколько раз. Каждый просмотр состоит из сравнения каждого элемента массива a_i с элементом a_{i-1} и обмена этих двух элементов, если они расположены не в нужном порядке.

  Например, в качестве исходного используем массив из элементов

  25 57 48 37 12 92 86 33.

  На первом просмотре делаем следующие сравнения:

  a_1 с a_2 (25 и 57) – нет обмена;

  a_2 с a_3 (57 и 48) – обмен;

  a_3 с a_4 (57 с 37) – обмен;

  a_4 с a_5 (57 с 12) – обмен;

  a_5 с a_6 (57 с 92) – нет обмена;

  a_6 с a_7 (92 с 86) – обмен;

  a_7 с a_8 (92 с 33) – обмен.

  Таким образом после *первого* просмотра элементы массива будут располагаться в следующем порядке:

  25 48 37 12 57 86 33 92.

  После первого просмотра наибольший элемент 92 находится в нужной позиции. В общем случае элемент $a_{(n-i+1)}$ будет находиться в нужной позиции после i-той итерации. Этот метод называется *«методом пузырька»* потому, что каждое число медленно всплывает как «пузырек» вверх в свою нужную позицию.

  После *второго* просмотра получим последовательность

  25 37 12 48 57 33 86 92. Элемент *86* на нужном месте.

  Поскольку каждая итерация помещает новый элемент в нужную позицию, то для сортировки некоторого массива, состоящего их n

элементов, требуется не более *n-1* итерации. Полный набор итераций для массива А выглядит следующим образом:

1-ая итерация 25 48 37 12 57 86 33 92;
2-ая итерация 25 37 12 48 57 33 86 92;
3-ая итерация 25 12 37 48 33 57 86 92;
4-ая итерация 12 25 37 33 48 57 86 92;
5-ая итерация 12 25 33 37 48 57 86 92;
6-ая итерация 12 25 33 37 48 57 86 92;
7-ая итерация 12 25 33 37 48 57 86 92.

- Метод улучшенной «пузырьковой» сортировки. Этот метод основан на методе «пузырька» с той лишь разницей, что количество итераций меньше.

П р и м е р программы, иллюстрирующей сортировку массива А методом «пузырька» приведен ниже:

```
program sort1;
 uses crt;
  var a:array[1..50] of real;
     i,j,n:integer;
        w:real;
 Begin
    Clrscr;
    Writeln('Введите количество элементов массива A');
    Readln(n);
    Writeln('Введите массив A');
    For i:=1 to n do
    Read(a[i]);
    Writeln;
    Writeln('Исходный массив A');
    For i:=1 to n do
      Write(a[i]:4:1,'':2);
      Writeln;
{Сортировка массива по возрастанию .При использовании метода улучшенной «пузырьковой» сортировки  i изменяется от 1 до n-1, j от 1 до n-i}
    For i:=1 to n do
        For j:=1 to n-1 do
          If a[j] > a[j+1] then begin   w:=a[j];
                                        a[j]:=a[j+1];
                                      a[j+1]:=w;  end;
    Writeln('Отсортированный по возрастанию массив A');
    For i:=1 to n do
      Write(a[i]:4:1,'':2);
      Writeln;
    {Сортировка массива по убыванию}
```

```
  For i:=1 to n do
    For j:=1 to n-1 do
      If a[j] < a[j+1] then begin  w:=a[j];
                                   a[j]:=a[j+1];
                                   a[j+1]:=w;  end;
  Writeln('Отсортированный по убыванию массив A');
  For i:=1 to n do
    Write(a[i]:4:1,'':2);
    Writeln;
    readkey;
End.
```

Сеанс работы с программой:

Введите количество элементов массива A
8
Введите массив A
25 57 48 37 12 92 86 33
Результаты:
Исходный массив A
25.0 57.0 48.0 37.0 12.0 92.0 86.0 33.0
Отсортированный по возрастанию массив A
12.0 25.0 33.0 37.0 48.0 57.0 86.0 92.0
Отсортированный по убыванию массив A
92.0 86.0 57.0 48.0 37.0 33.0 25.0 12.0

2. *Сортировки выбором*

Наиболее распространены такие методы этого вида сортировки, как :

- с поиском минимального значения;
- с использованием бинарных деревьев;
- с использованием метода «турнира с выбыванием»;
- «пирамидальная» сортировка.

Рассмотрим подробнее первый метод этого вида сортировки – с поиском минимального значения. Суть его заключается в следующем. Сначала в массиве необходимо найти минимальный элемент, затем поменять его местом с первым, затем в усеченном (исключая первый элемент) массиве опять найти минимальный элемент и поставить его на второе место и так далее.

Пример

Пусть задан массив A из элементов 1 3 0 9 2.

Процесс сортировки выбором по методу поиска минимального значения пройдет через следующие шаги:

0:	1	3	0	9	2	min=a[3]=0	переставляем a[1] < -- > a[3]
1:	0 |	3	1	9	2	min=a[3]=1	переставляем a[2] < -- > a[3]
2:	0	1 |	3	9	2	min=a[5]=2	переставляем a[3] < -- > a[5]
3:	0	1	2 |	9	3	min=a[5]=3	переставляем a[4] < -- > a[4]
4:	0	1	2	3	9	Готово	

Здесь знак | отделяет уже отсортированную часть массива от еще не отсортированной.

Ниже приводится п р и м е р программы «Сортировка выбором»:

```
program sort2;
 uses crt;
  var a:array[1..50] of real;
     i,j,n,k:integer;
       min,w:real;
 Begin
    Clrscr;
    Writeln('Введите количество элементов массива A');
    Readln(n);
    Writeln('Введите массив A');
    For i:=1 to n do
    Rean(a[i]);
    Writeln;
    Writeln('Исходный массив A');
    For i:=1 to n do
      Write(a[i]:4:1,'':2);
      Writeln;
    {Сортировка массива  выбором}
    For i:=1 to n do
     Begin
      min:=a[i]; k:=i;
      For j:=i+1 to n do
        If a[j] < = min then Begin min:=a[j];
                                   k:=j;  end;
       {Перестановка i-го и min элементов}
       w:=a[i];
       a[i]:=a[k];
       a[k]:=w;
     End;
    Writeln('Отсортированный массив A');
    For i:=1 to n do
      Write(a[i]:4:1,'':2);
      Writeln;
      readkey;
  End.
```

Сеанс работы с программой:

Введите количество элементов массива A

5

Введите массив A

1 3 0 9 2

Исходный массив A

1.0 3.0 0.0 9.0 2.0

Отсортированный массив A

0.0 1.0 2.0 3.0 9.0

3. *Сортировка «вставками»*

Наиболее распространены такие методы этого вида сортировки как:

- сортировка «*простыми*» вставками;
- сортировка «*бинарными*» вставками;
- сортировка *Шелла*;
- сортировка с вычислением адреса.

Суть, например, сортировки «*простыми*» вставками заключается в последовательном просмотре массива a_2 , ... , a_n с тем, чтобы каждый новый элемент a_i вставлять на подходящее место в уже упорядоченной совокупности a_1, ... , a_{i-1}. Это место определяется последовательным сравнением a_i с упорядоченными элементами a_1, ... , a_{i-1}.

При сортировке «*бинарными*» вставками место, на которое надо вставить элемент a_i в уже упорядоченную совокупности a_1, ... , a_{i-1} определяется алгоритмом деления пополам. Таким образом, получается новый алгоритм сортировки «*бинарными*» вставками, который понимается как «вставка делением пополам.»

4. *Сортировка «слияниями»* (Алгоритм фон Неймана).

Суть заключается в упорядочении массива a_1, ..., a_n по неубыванию. Метод основан на многократных слияниях уже упорядоченных групп элементов массива. Вначале весь массив рассматривается как совокупность упорядоченных групп по одному элементу в каждом. Слиянием соседних групп получаются упорядоченные группы, каждая из которых содержит два элемента (кроме, может быть, последней группы, которой не нашлось парной). Далее полученные группы укрупняются тем же способом и т.д. Здесь приходится оперировать не только с массивом a_2, ... , a_n , но и с вспомогательным массивом b_1, ... , b_n.

Пример

Вставить в упорядоченный по возрастанию массив A = {3 4 7 9} новый элемент, равный 5, таким образом, чтобы сохранилась упорядоченность.

Алгоритм решения задачи следующий:

1. Найти в массиве тот элемент, который больше вставляемого. Это элемент a[3] = 7. Для этого последовательно просмотреть все элементы, начиная с первого.
2. Увеличить длину массива на 1. Получим массив A = {3 4 7 9 X}.

3. Все элементы, стоящие правее от найденного, включая его самого, то есть от 3-го, сдвинуть вправо. Получим массив A = {3 4 7 7 9}.
4. На освободившуюся позицию, то есть на место элемента a[3], вставить новый элемент, то есть 5.
   В итоге получаем массив A = {3 4 5 7 9}.

При решении задач такого рода следует учитывать, что, если все элементы массива меньше вставляемого, новый элемент надо вставить в конец массива, и если все элементы массива больше вставляемого, то новый элемент надо вставить в начало массива.

Ниже приведена программа, реализующая данный алгоритм:

```
program sort3;
 uses crt;
  var a:array[1..50] of real;
     i,j,n,k:integer;
          g:real;
 Begin
    Clrscr;
    Writeln('Введите количество элементов массива A');
    Readln(n);
    Writeln('Введите массив A');
    For i:=1 to n do
    Read(a[i]);
    Writeln;
    Writeln('Введите число, которое нужно вставить');
    Readln(g);
    Writeln('Исходный массив A');
    For i:=1 to n do
      Write(a[i]:4:1,'':2);
      Writeln;
    {1. Ищем элемент больше вставляемого}
    k:=1; {k - индекс сравниваемого элемента}
    While (k<=n) and (g>=a[k]) do {если k не вышла за границу n и
                                   вставляемый элемент меньше или равен a[k]}
      k:=k+1;                     {то переходим к следующему элементу}

    {2. Увеличиваем длину массива на 1}
    n:=n+1;
    {3.Сдвигаем элементы, начиная с k-го вправо}
    For i:=n downto k+1 do
      a[i]:=a[i-1];
    {4.В a[k] заносим g}
    a[k]:=g;
    Writeln('Новый массив A');
```

```
For i:=1 to n do

  Write(a[i]:4:1,'':2);
  Writeln;
  readkey;
End.
```

Сеанс работы с программой:

Введите количество элементов массива A
4
Введите массив A
3 4 7 9
Введите число, которое нужно вставить
5
Исходный массив A
3.0 4.0 7.0 9.0
Новый массив A
3.0 4.0 5.0 7.0 9.0

ЛЕКЦИЯ 16-17. ПРОЦЕДУРЫ И ФУНКЦИИ ОБРАБОТКИ МНОГОМЕРНЫХ МАССИВОВ

Основные операции по обработке данных, организованных в многомерные массивы, рассмотрим на уровне подпрограмм-процедур и подпрограмм-функций, описанных для формальных параметров.

В качестве примера использования в программах составленных ниже подпрограмм опишем следующие массивы:

Матрицы A, B, C размера M x N и одномерные массивы D и F длиной N (N <=10).

Опишем их.

```
Type Matr = Array[1..10,1..10] Of Real;
     Mas = Array[1..10] Of Real;
Var A, B, C : Matr;
    F,D : Mas;
```

1. Ввод матрицы по строкам.

```
Procedure Vvod(M,N:Integer; Var V:Matr);
  Var I,J:Integer;
    Begin
      For I:=1 To M Do
        Begin
          Writeln('Введите',I,'строку матрицы');
          For J:=1 To N Do
            Readln(V[I,J]);
        End;
    End;
```

Обращение из программы:

```
Vvod(4,5,A);
Vvod(3,5,B);
```

2.Вывод матрицы (в виде матрицы).

```
Procedure Vyvod(M, N:Integer;  V:Matr);
  Var I,J:Integer;
    Begin
      For I:=1 To M Do
        Begin
          For J:=1 To N Do
            Write(V[I,J]:4:1,'':2);
          Writeln;
        End;
    End;
```

Обращение из программы:

```
Vyvod(4,5,A);
Vyvod(3,5,B);
```

3.Вычисление суммы элементов матрицы.

```
Function Sum(M,N:Integer;V:Matr):Real;
    Var I,J:Integer;
        S:Real;
    Begin
      S:=0;
      For I:=1 To M Do
        For J:=1 To N Do
          S:=S+V[I,J];
    Sum:=S;
    End;
```

Обращение из программы:

```
Sa:=Sum(4,5,A);
Sb:=Sum(3,5,B);
```

4.Суммирование диагональных элементов матрицы. (Вычисление следа матрицы).

```
Function Sled(N:Integer;V:Matr):Real;
    Var I:Integer;
        S:Real;
    Begin
      S:=0;
      For I:=1 To N Do
        S:=S+V[I,I];
    Sled:=S;
    End;
```

Обращение из программы:

```
Sla:=Sled(4,A);
Slb:=Sled(3,B);
```

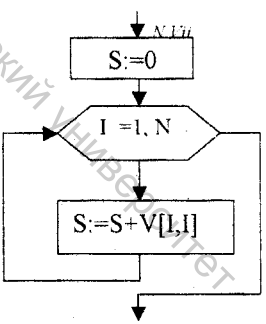


Рис.23. Блок-схема вычисления следа матрицы

5. Суммирование матриц (матрицы должны быть одного размера).

```
Procedure Sum_Matr(N,M:Integer;V,W:Matr;Var U:Matr);
Var I,J:Integer;
    Begin
     For I:=1 To N Do
       For J:=1 To M Do
         U[I,J]:=V[I,J]+W[I,J];
    End;
```

Обращение из программы:

```
Sum_Matr(3,5,A,B,D);
```

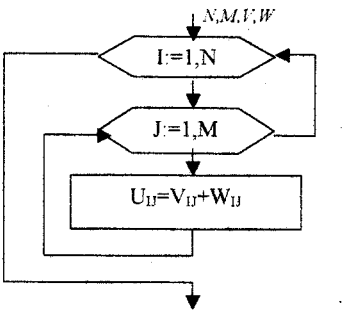


Рис.24. Блок-схема вычисления суммы матриц

6.Суммирование матрицы по строкам.

```
Procedure Sum_Str(N,M:Integer;V:Matr;Var S:Mas);
    Var I,J:Integer;
      Begin
       For I:=1 To N Do
           Begin
            S[I]:=0;
            For J:=1 To M Do
            S[I]:=S[I]+V[I,J];
           End;
      End;
```

Обращение из программы:

```
Sum_Str(4,5,A,S_A);
Sum_Str(3,5,B,S_B);
```

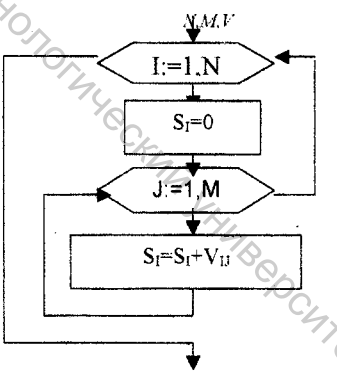


Рис.25. Блок-схема вычисления суммы сток матрицы

7. Умножение матрицы на вектор. $Fi = \sum_{J=1}^{M} A_{IJ} * D_J$. Длина исходного вектора равна количеству столбцов матрицы. Длина результирующего вектора равна количеству строк матрицы.

```
Procedure Vector(N,M:Integer;V:Matr;T:Mas;Var S:Mas);
    Var I,J:Integer;
      Begin
       For I:=1 To N Do
           Begin
            S[I]:=0;
            For J:=1 To M Do
            S[I]:=S[I]+V[I,J]*T[J];
           End;
    End;
```

Обращение из программы:
Vector (4,5,A,D,F);

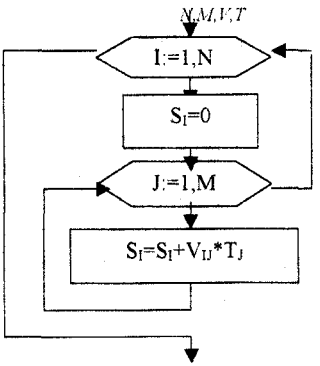


Рис.26. Блок-схема умножения матрицы на вектор

8. Удаление строки матрицы.

Требуется удалить строку с заданным номером К. Все строки, начиная с (К+1)-ой, нужно переместить вверх. Число строк уменьшится на 1.

```
Procedure Udstr(N,M,Kk:Integer;V:Matr;Var W:Matr);

    Var I,J:Integer;
      Begin
       W:=V;
        N := N-1;
       For I:=Kk To N Do
            For J:=1 To M Do
            W[I,J] := W[I+1,J];
           End;
```

Обращение из программы:
Udstr (4,5,K,A,D);

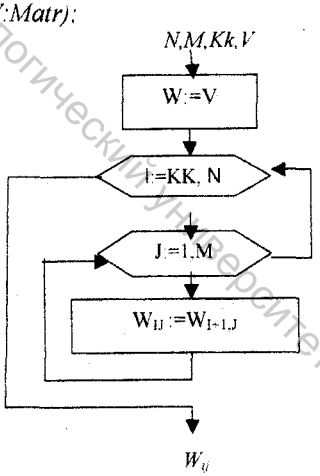


Рис.27. Блок-схема удаления строки матрицы

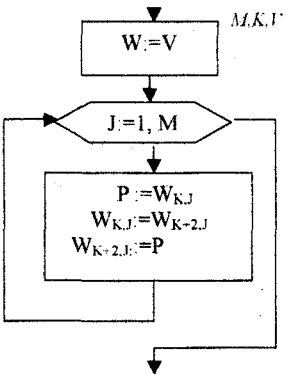
9. Перестановка строк матрицы.

Пусть требуется поменять местами К-ую и (К+2)-ую строки.

Рассмотрим два варианта алгоритма:

а) с использованием вспомогательной переменной P.

```
Procedure Perest(M,K:Integer;V:Matr;Var W:Matr);
    Var I,J:Integer;
        P:Real;
  Begin
    W:=V;
    For J:=1 To M Do
      Begin
        P:=W[K,J];
        W[K,J]:=W[1+2,J];
        W[K+2,J]:=P;
      End;
  End;
```



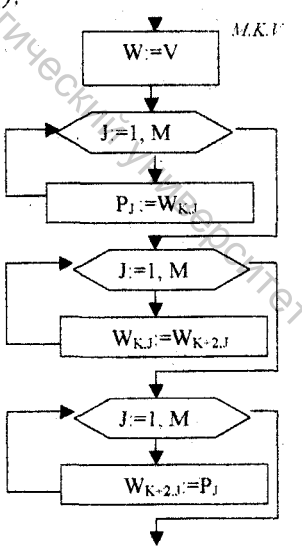
Обращение из программы:

```
Perest (5,K,A,D);
```

Рис.28. Блок-схема перестановки строк матрицы

б) с использованием вспомогательного массива P_j: одна из строк пересылается целиком в массив P_j для временного хранения:

```
Procedure Perest1(M,K:Integer;V:Matr;Var W:Matr);
    Var I,J:Integer;
        P:Mas;
  Begin
    W:=V;
    For J:=1 To M Do
        P[J]:=W[K,J];
    For J:=1 To M Do
        W[K,J]:=W[K+2,J];
    For J:=1 To M Do
        W[K+2,J]:=P[J];
      End;
```



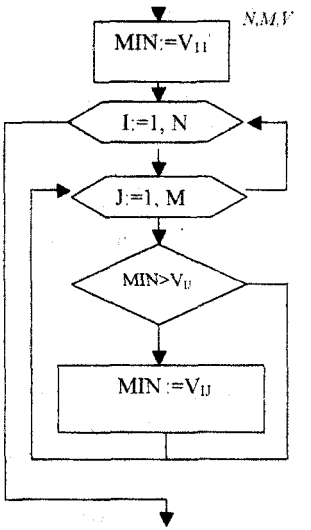
Обращение из программы:

```
Perest1 (5,2 ,A,D);
```

Рис.29. Блок-схема перестановки строк матрицы

10. Поиск минимального (максимального) элемента матрицы.

```
Procedure Min_El(N,M:Integer;V:Matr;Var Min:Real;Var  L,K:Integer);
Var I,J:Integer;
Begin
 Min:=V[1,1];
 K:=1; L:=1;
 For I:=1 To N Do
   For J:=1 To M Do
     If Min>V[I,J] Then
       Begin
         Min:=V[I,J];
           K:=I;
           L:=J;
       End;
 End;
```



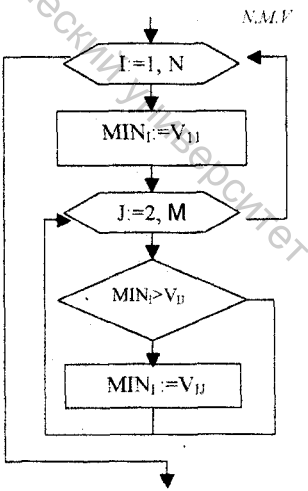
Обращение к процедуре:

```
Min_El(4,5,A,Min_A,K_A,L_A);
```

Рис.30. Блок-схема поиска минимального элемента матрицы

11. Поиск минимального элемента строки (столбца) матрицы.

```
Procedure Min_Str(N,M:Integer;V:Matr;Var Min:Mas);
    Var I,J:Integer;
    Begin
     For I:=1 To N Do
       Begin
         Min[I]:=V[I,1];
         For J:=2 To M Do
           If Min[I]>V[I,J] Then
             Min[I]:=V[I,J];
         End;
     End;
```



Обращение к процедуре:

```
Min_El(4,5,A,Min_A);
```

Рис.31. Блок-схема поиска максимального элемента строки матрицы

Пример:

Установить соответствие производственной программы выпуска продукции производственной мощности цеха.

В механосборочном цехе завода четыре производственных участка, на которых установлено следующее технологическое оборудование (N_i):

на участке токарных станков – 17 шт;
фрезерных - 11 шт;
протяжных - 14 шт;
шлифовальных - 26 шт.

Эффективный фонд времени – **F**=3931 час в год;

Среднее выполнение норм выработки =110% (**K**=1,1);

Заданная программа характеризуется следующими данными, приведенными в таблице 3:

Таблица 3. Исходные данные

	Обознач	**А**	**Б**	**В**
Количество комплектов на программу выпуска	**(Qj)**	300	700	500
Плановая трудоемкость токарных работ	$t_{шкij}$	*75*	*35*	*50*
Плановая трудоемкость фрезерных работ		*35*	*25*	*38*
Плановая трудоемкость протяжных работ		*40*	*32*	*34*
Плановая трудоемкость шлифовальных работ		*70*	*80*	*60*

Вначале определяется станкоемкость планируемой продукции по видам работ

$$ST_i = \frac{\sum_{J=1}^{m} Q_J * T_{шкiJ}}{K}, \qquad \text{ч/год}$$

где m – число комплектов оборудования (в данном случае – 3 – А, Б, В);

К – коэффициент выполнения норм выработки.

Далее определяется действительный фонд времени по группам оборудования

$$FD_i = F * N_i$$

Затем определяется коэффициент загрузки оборудования

$$KZ_i = \frac{ST_i}{FD_i} \le 1$$

Таким образом, исходными данными к задаче являются:

одномерные массивы Ni = {17, 11, 14, 26};

Qj = {300, 700, 500};

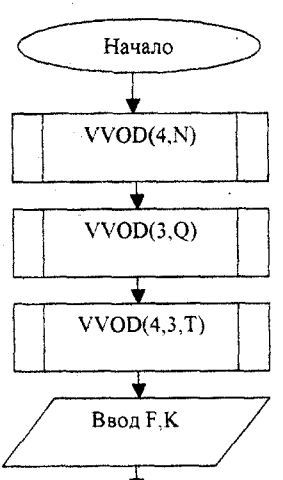
матрица трудоемкостей по видам работ Tij = $\begin{vmatrix} 75 & 35 & 50 \\ 35 & 25 & 38 \\ 40 & 32 & 34 \\ 70 & 80 & 60 \end{vmatrix}$.

Ниже приводится блок-схема (рис.32) и программа решения этой задачи на языке Turbo Pascal.

Текст программы:

```
Program Zagruzka;
 Uses Crt;
 Type matr=array[1..4,1..3] of integer;
     mas=array[1..4] of real;
    Var T:matr;
       N,Q,S,ST,FD,KZ:mas;
       F,i,j:integer;
       k:real;
 (************************************)
 Procedure Vvod(m:integer;var v:mas);
  Begin
   For i:=1 to m do Readln(v[i]);  End;
 (************************************)
  Procedure Vvod1(m,l:integer;var v:matr);
  Begin
   For i:=1 to m do
   Begin  Writeln('введите ',i:2,' строку матрицы');
    For j:=1 to l do readln(v[i,j]); End;
  End;
 (************************************)
 Procedure Vyvod(m:integer; v:mas);
 Begin
  For i:=1 to m do   Write(v[i]:5:2,'':2);    Writeln;   End;
 (************************************)
 Procedure Vyvod1(m,l:integer; v:matr);
 Begin
  For i:=1 to m do
  Begin
   For j:=1 to l do   Write(v[i,j]:3);    Writeln;   End;
 End;
 (************************************)
 Begin
 Clrscr;
 Writeln('Введите количество оборудования каждого вида N');
 Vvod(4,N);
 Writeln('Введите количество комплектов на программу выпуска Q');
 Vvod(3,Q);
```

```
Writeln('Введите трудоемк. по видам работ и видам оборудования T');
Vvod1(4,3,T);
Writeln('Введите эффективный фонд времени F'); Readln(F);
Writeln('Введите коэффициент k'); Readln(k);
For i:=1 to 4 do
 Begin
  S[i]:=0;
  For j:=1 to 3 do
   S[i]:=Q[j]*T[i,j];
  ST[i]:=S[i]/k;
 End;
For i:=1 to 4 do
  Fd[i]:=F*N[i];
For i:= 1 to 4 do
  KZ[i]:=ST[i]/FD[i];
Writeln('Количество комплектов на программу выпуска Q');
Vyvod(3,Q);
Writeln('Количество оборудования каждого вида N');
Vyvod(4,N);
Writeln('Трудоемкость по видам работ и видам оборудования T');
Vyvod1(4,3,T);
Writeln('Станкоемкость продукции ST');
Vyvod(4,ST);
Writeln('Действительный фонд времени FD');
Vyvod(4,FD);
Writeln('Коэффициенты загрузки оборудования KZ');
Vyvod(4,KZ);
Readln;
End.
```



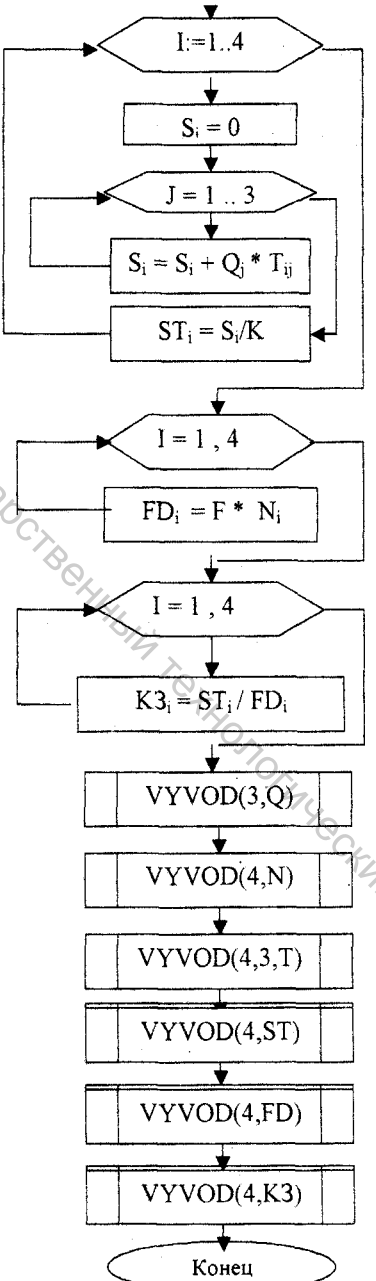


Рис.32. Блок-схема решения задачи Zagruzka

ЛЕКЦИЯ 18. ПРИБЛИЖЕННЫЕ МЕТОДЫ РЕШЕНИЯ АЛГЕБРАИЧЕСКИХ И ТРАНСЦЕНДЕНТНЫХ УРАВНЕНИЙ. МЕТОД ПРОСТОЙ ИТЕРАЦИИ

В практических вычислениях довольно часто приходится решать уравнения вида:

$$f(x)=0, \qquad (1)$$

Где функция $f(x)$ определена и непрерывна на некотором конечном или бесконечном интервале $a < x < b$.

Если функция представляет собой многочлен, то уравнение $f(x)=0$ называют алгебраическим, если же в функцию $f(x)$ входят элементарные (тригонометрические, логарифмические, показательные и т.п.) функции, то такое уравнение называется трансцендентным.

Всякое значение x^*, обращающее функцию $f(x)$ в ноль, т.е. такое, что

$$f(x^*)=0,$$

называют корнем уравнения , а способ нахождения этого значения x^* и есть решение уравнения.

Найти корни уравнения вида $f(x)=0$ точно удается лишь в частных случаях. Кроме того, часто уравнение содержит коэффициенты, известные лишь приблизительно, и, следовательно, сама задача о точном определении корней уравнения теряет смысл. Поэтому разработаны методы численного решения уравнений, которые позволяют отыскать приближенные значения корней этого уравнения.

При этом приходится решать две задачи:

1. Отделение корней, т.е. отыскание достаточно малых областей, в каждой из которых заключен только один корень уравнения;
2. Вычисление корней с заданной точностью.

При выделении областей, в которых находятся действительные корни уравнения, можно воспользоваться тем, что если на концах некоторого отрезка непрерывная функция $f(x)$ принимает значения разных знаков, то на этом отрезке уравнение $f(x)=0$ имеет хотя бы один корень.

Для выделения областей, содержащих один и только один корень, можно воспользоваться графическим способом.

Для решения второй задачи существуют многочисленные приближенные методы, из которых рассмотрим *метод простой итерации, метод Ньютона и метод половинного деления.*

Полагаем, что нам известен отрезок $\alpha <= x <= \beta$, внутри которого существует и располагается один и только один корень уравнения $f(x)=0$.

Задача состоит в нахождении этого корня.

Метод итераций

Уравнение $f(x)=0$ представим в форме

$$x=\varphi(x), \qquad (2)$$

что всегда можно сделать и притом многими способами. Например, можно выделить из уравнения (1) x, остальное перенести в правую часть (это и будет $\varphi(x)$). Или умножить левую и правую часть уравнения на произвольную константу λ и прибавить к левой и правой частям x, т.е. представить (1) в виде

$$x = x + \lambda f(x).$$

При этом $\varphi(x) = x + \lambda f(x)$.

Выберем на отрезке [α, β] произвольную точку x_0 - нулевое приближение и примем в качестве следующего приближения:

$$X_1 = \varphi(x_0),$$
$$X_2 = \varphi(x_1).$$

И вообще пусть x_n получается из x_{n-1} по формуле

$$x_n = \varphi(x_{n-1}). \qquad (3)$$

Этот процесс последовательного вычисления чисел x_n $(n=1, 2, 3,...)$ по формуле (3) называется методом итераций.

Если на отрезке [α, β], содержащем корень x=x* уравнения (2), а также его последовательные приближения $x_0, x_1, \ldots x_n, \ldots$, вычисляемые по методу итераций, выполнено условие

$$|\varphi'(x)| \le q < 1, \qquad (4)$$

то процесс итераций сходится , т.е., увеличивая n, можно получить приближенное, сколь угодно мало отличающееся от истинного значения корня x*. Процесс итераций следует продолжать до тех пор, пока для двух последовательных приближений x_{n-1} и x_n не будет обеспечено выполнение неравенства

$$|x_n - x_{n-1}| \le \frac{1-q}{q}\varepsilon; \qquad (5)$$

при этом всегда будет выполнено неравенство $|x^* - x_n| \le \varepsilon$,

где ε – заданная абсолютная погрешность корня x*.

Или в более простом виде: $|x_n - x_{n-1}| \le \varepsilon$, (6)

при выполнении которого также будет обеспечена заданная точность определения корня x*.

При практическом нахождении корней по методу итераций нужно при переходе от уравнения (1) к уравнению (2) стремиться представить φ(x) так, чтобы производная φ(x) по абсолютной величине была возможно меньше 1. В таком случае корень всегда будет найден, и чем меньше величина q, тем меньше число итераций для этого потребуется.

Пример 1

Методом итераций найти корень уравнения

$$f(x) = \arcsin(2x + 1) - x^2 = 0, \qquad (7)$$

расположенный на отрезке [-0.5; 0] с абсолютной погрешностью $\varepsilon = 10^{-4}$. Определить также число итераций, необходимое для нахождения корня.

Уравнение (7) преобразуем к виду (2) следующим образом:

$$\arcsin(2x + 1) = x^2$$

$$\sin(\arcsin(2x + 1)) = \sin(x^2)$$

$$2x + 1 = \sin(x^2)$$

Это уравнение легко может быть преобразовано к виду

$$x = \varphi(x), \quad \text{где} \quad \varphi(x) = 0{,}5(\sin x^2 - 1). \qquad (8)$$

Находим $\varphi'(x) = x \cos x^2$.

Очевидно, $|\varphi'(x)| = |x \cos x^2| \le 0.5$ для всех $-0.5 \le x \le 0$. Поэтому $q = 0.5$ и процесс итераций сходится.

За начальное приближение можно принять любую точку отрезка [-0.5; 0], например $x_0 = -0{,}4$. Алгоритм нахождения корня уравнения (8) представляет следующую последовательность действий:

1. Полагаем $x = -0{,}4$, $\varepsilon = 0{,}0001$ и $n = 0$.
2. Вычисляем следующее приближение x_{n+1} по формуле

$$X_{n+1} = 0{,}5(\sin x_n^2 - 1). \qquad (9)$$

3. Вычисляем разность $\delta = x_{n+1} - x_n$ и увеличиваем величину n на единицу.
4. Проверяем условие $|\delta| > \varepsilon$. Если это условие выполняется, то возвращаемся к вычислению следующего приближения x_{n+1} по формуле (9), т.е. к п.2. Если условие не выполняется, т.е. $|\delta| \le \varepsilon$, то результатом считаем величину x_{n+1} и заканчиваем вычисления. При этом число n будет равно числу выполненных итераций.

При составлении блок-схемы и программы вводить переменную с индексом x_n нет необходимости, поскольку результатом будет одно число (корень уравнения), а все последовательные приближения в конечном счете не нужны. На каждом этапе вычислений необходимо помнить лишь два соседних приближения, поэтому приближение x_n обозначим через x, а приближение x_{n+1} – через y. По окончании каждой итерации будем полагать x = y.

Таким образом, методом итераций для решения уравнений может быть представлен в виде блок-схемы, представленной на рис. 33.

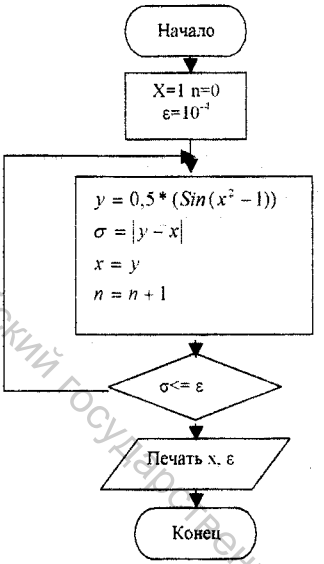


Рис. 33. Блок-схема решения уравнения (7) по методу итераций

Программную реализацию метода итераций представим в двух вариантах:

1) с использованием подпрограммы-функции для описания решаемого уравнения:

```
program primer;
var a,x,y,e,delta:real;
        n:integer;
(****************************************)
function f:real;
begin                                   { подпрограмма-функция,
  y:=0.5*(sin(x*x)-1);                  описывающая уравнение
  f:=y;                                 y= 0,5(sin(x)^2 - 1)}
end;
(*********************************)
begin
writeln('a, e?');
readln(a,e);
n:=0;
x:=a;
repeat
n:=n+1;
y:=f;
delta:=abs(y-x);
```

```
x:=y;
until delta<=e;
writeln('x=',x:7:5,'  n=',n:2);
end.
```

2) с использованием подпрограммы-функции для описания решаемого уравнения и подпрограммы процедуры для описания метода итераций.

```
program primer;
var a,x,y,e,delta:real;
      n:integer;
(***********************************)
function f:real;
begin                                  { подпрограмма-функция,
  y:=0.5*(sin(x*x)-1);                 описывающая уравнение
  f:=y;                                y= 0,5(sin(x)^2 - 1)}
end;
(***********************************)
procedure iter;                        { подпрограмма-процедура,
  begin                                описывающая метод
    n:=0;                              итераций}
  x:=a;
  repeat
   n:=n+1;
   y:=f;
   delta:=abs(y-x);
   x:=y;
  until delta<=e;
 end;
(***********************************)
begin                                  {головная программа}
 writeln('a, e?');
 readln(a,e);
 iter;
writeln('x=',x:7:5,'  n=',n:2);
end.
```

В обоих случаях
результат x=-0.41453 n=7.

ЛЕКЦИЯ 19. ПРИБЛИЖЕННЫЕ МЕТОДЫ РЕШЕНИЯ АЛГЕБРАИЧЕСКИХ И ТРАНСЦЕНДЕНТНЫХ УРАВНЕНИЙ. МЕТОД НЬЮТОНА. МЕТОД ПОЛОВИННОГО ДЕЛЕНИЯ

Метод Ньютона.

Пусть уравнение $f(x)=0$ имеет один корень на отрезке $[\alpha, \beta]$, причем $f'(x)$ и $f''(x)$ определены, непрерывны и сохраняют постоянные знаки на отрезке $[\alpha, \beta]$.

Рассмотрение метода Ньютона начнем с его графического представления (см. рис.21).

Возьмем некоторую точку x_0 отрезка $[\alpha, \beta]$ и проведем в точке $P_0\{x_0, f(x_0)\}$ графика функции касательную к кривой $y=f(x)$ до пересечения с осью Ox. Абсциссу x_1 точки пересечения можно взять в качестве приближенного значения корня. Проведя касательную через новую точку $P_1\{x_1, f(x_1)\}$ и находя точку ее пересечения с осью Ox, получим второе приближение корня x_2. Аналогично определяются следующие приближения.

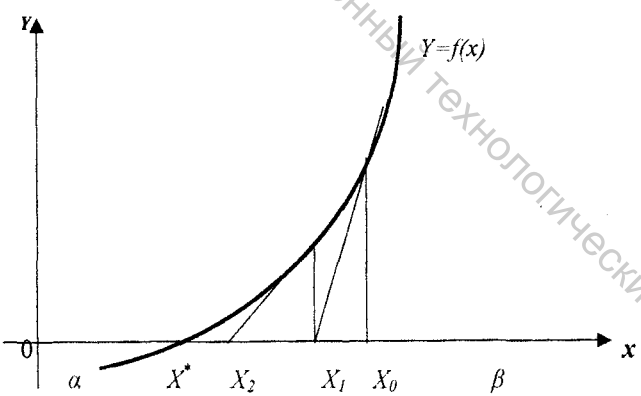


Рис 34.Графическое представление метода Ньютона.

Выведем формулу для последовательных приближений к корню. Уравнение касательной, проходящей через точку P_0, имеет вид:

$y-f(x_0)=f'(x_0)\cdot(x-x_0)$.

Полагая Y=0, находим абсциссу x_1 точки пересечения касательной с осью Ox.

$X_1 = x_0 - f(x_0)/f'(x_0)$.

Следующие приближения находим соответственно по формулам:

$X_2 = x_1 - f(x_1)/f'(x_1)$.

. . . . . .

$X_n = x_{n-1} - f(x_{n-1})/f'(x_{n-1})$.

Процесс вычислений прекратим при выполнении условия

$X_n - x_{n-1} <= \varepsilon$, где ε – заданная абсолютная погрешность корня x^*.

Начальное приближение корня x_0 следует выбирать так, чтобы было выполнено условие

$$F(x_0)*f''(x_0) > 0. \qquad (10)$$

В противном случае сходимость метода Ньютона не гарантируется.

Чаще всего выбирают $x_0 = \alpha$ или $x_0 = \beta$, в зависимости от того, в какой из точек выполняется условие (10).

Замечание

Метод Ньютона эффективен для решения тех уравнений $f(x) = 0$, для которых значение модуля производной $f'(x)$ близ корня достаточно велико, т.е. график функции $f(x)$ в окрестности данного корня имеет большую крутизну.

Пример

Методом Ньютона найти корень уравнения

$$F(x) = \sin x - x + 0.15 = 0 \qquad (11)$$

на отрезке [0.5; 1] с абсолютной погрешностью $\varepsilon = 10^{-4}$.

Решение:

Находим $f'(x) = \cos x - 1$.

Расчетная формула для вычисления корня уравнения (11) по методу Ньютона имеет вид

$$x_n = x_{n-1} - (\sin x_{n-1} - x_{n-1} + 0.15) / (\cos x_{n-1} - 1),$$

где n=1,2,3,. . .

Начальное приближение x_0 выбираем таким образом, чтобы выполнялось условие

$$F(x_0)*f''(x_0) > 0.$$

$f''(x) = -\sin x < 0$ для всех $0.5 < x < 1$. Поэтому нужно подобрать x_0, для которого $F(x_0) < 0$. Это условие выполняется при x = 1. Это значение и выберем за начальное приближение корня.

Алгоритм нахождения корня представляет собой следующую последовательность действий:

1. Полагаем $x_0 = 1$, $\varepsilon = 10^{-4}$, $n = 0$.
2. Вычисляем следующее приближение
$$x_{n+1} = (x_n - \sin x_n - x_n - 0.15) / (\cos x_n - 1).$$
3. Вычисляем разность $\delta = x_{n+1} - x_n$ и увеличиваем n на единицу.
4. Проверяем условие $\delta <= \varepsilon$. Если это условие не выполняется, то возвращаемся к вычислению следующего приближения корня (п.2).Если это условие выполняется, то за результат принимаем величину x_{n+1} и заканчиваем вычисления. При этом число n равно числу выполненных итераций. Как и в методе итераций, переменную с индексом x_{n+1} не вводим, приближение x_n обозначим через x, x_{n-1} – через y.

Таким образом, блок-схема, реализующая метод Ньютона для решения алгебраических и трансцендентных уравнений может быть представлена в следующем виде (см. рис. 35):

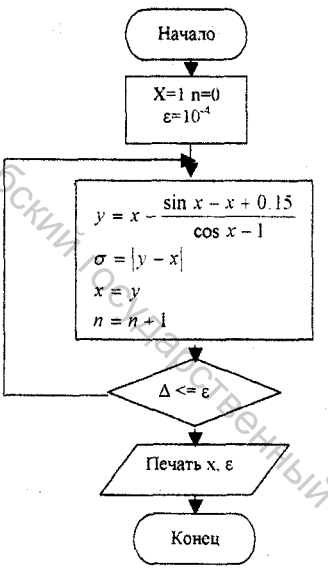


Рис.35 Блок-схема решения уравнения (11) по методу Ньютона

Программу для решения рассмотренного выше уравнения, как и в предыдущем случае, составим с использованием подпрограммы-функции для описания решаемого уравнения и подпрограммы-процедуры для реализации метода решения:

```
program met_new;
uses crt;
 var x,y,a,eps,delta:real;
        n:integer;
(*************************************)
 function f:real;
  begin
   f:=(sin(x)-x+0.15)/(cos(x)-1);
  end;
(*************************************)
procedure metod;
begin
 n:=0;
 repeat
  y:=x-f;
  delta:=abs(y-x);
```

```
 x:=y;
 n:=n+1;
until delta<eps;
end;
(************************************)
begin
 writeln('Введите начальное приближение корня a');
 readln(a);
 writeln('Введите точность eps');
 readln(eps);
 x:=a;
 metod;
 writeln('x=', x:5:3, ' n=',n:2);
 readln;
end.
```

Результат: x=0.98112 n=2

Метод половинного деления

Пусть дано уравнение $f(x)=0$, где функция $f(x)$ непрерывна на отрезке $[\alpha, \beta]$ и $f(\alpha)*f(\beta)<0$.

Если вид функции $f(x)$ достаточно сложный, то вычисление функции $f'(x)$ в методе Ньютона и $\varphi'(x)$, необходимой для оценки сходимости в методе итераций, затруднительно. Для решения таких уравнений можно использовать метод половинного деления, который хотя и требует значительного объема вычислительной работа, но всегда приводит к искомому результату.

Для нахождения корня уравнения $f(x)=0$, принадлежащего отрезку $[\alpha, \beta]$, делим отрезок пополам, т.е. выбираем начальное приближение равным

$x_0 = (\alpha + \beta)/2$.

Если $f(x_0)=0$, то x_0 является корнем уравнения. Если $f(x_0) \neq 0$, то выбираем тот из отрезков $[\alpha, x_0]$ или $[x_0, \beta]$, на концах которого функция $f(x)$ имеет противоположные знаки. Полученный отрезок снова делим пополам и проводим то же рассмотрение и т.д.

Процесс деления отрезка пополам продолжаем до тех пор, пока длина отрезка, на концах которого функция имеет противоположные знаки, не будет меньше наперед заданного числа ε.

Пример.

Методом половинного деления найти корень уравнения

$$f(x) = x - \sqrt{9+x} + x^2 - 4 = 0 \qquad (12)$$

на отрезке [2; 3] с абсолютной погрешностью $\varepsilon = 10^{-4}$.

Алгоритм нахождения корня уравнения (12) представляет следующую последовательность действий:

1. Полагаем $\alpha=2$, $\beta=3$ и $\varepsilon=0{,}0001$.

2. Вычисляем $f(\alpha) = \alpha - \sqrt{9+\alpha} + \alpha^2 - 4 = 0$.

3. Вычисляем $x=(\alpha+\beta)/2$ и значение функции

$f(x) = x - \sqrt{9+x} + x^2 - 4$ в этой точке.

4. Проверяем условие $f(x)=0$. Если это условие выполняется, то считаем x корнем и заканчиваем вычисления. Если условие не выполняется, то переходим к выбору отрезка, на концах которого функция f(x) имеет разные знаки, а именно:

5. Проверяем условие $f(\alpha) * f(x) < 0$. Если это условие выполняется, то полагаем $\beta=x$ и переходим к п.6. Если условие не выполняется, то полагаем $\alpha=x$, $f(\alpha)=f(x)$ и переходим к п.6.

6. Проверяем условие $\beta - \alpha <= \varepsilon$. Если оно не выполняется, то возвращаемся к процессу деления отрезка пополам, т.е. к п.3. Если условие выполняется, то за результат принимаем значение x и заканчиваем вычисления.

Ниже представлены блок-схема алгоритма решения уравнения (12) по методу половинного деления (см. рис. 36) и программа на языке Turbo Pascal .

```
program met_pd;
 var x,y,z,eps,a,b:real;  n:integer;
 (*********************************)
 function f(xx:real):real;
  begin  f:=xx-sqrt(9+xx)-sqr(xx)-4;   end;
 (*********************************)
 procedure metod;
 begin
  y:=f(a); n:=0;
   repeat
    x:=(a+b)/2;
    z:=f(x);
     if z<>0 then
          if z*y>0 then
                    begin a:=x;  y:=z; end
                     else b:=x;
     n:=n+1;
   until (b-a)<eps;
 end;
 (*********************************)
 begin
 writeln('Введите a,b');
readln(a,b);
```

```
 writeln('Введите eps');
readln(eps);
 metod;
 writeln('x=',x:5:3,'  n=',n:2);  readln;
 end.
```

Результат: x=2.25775

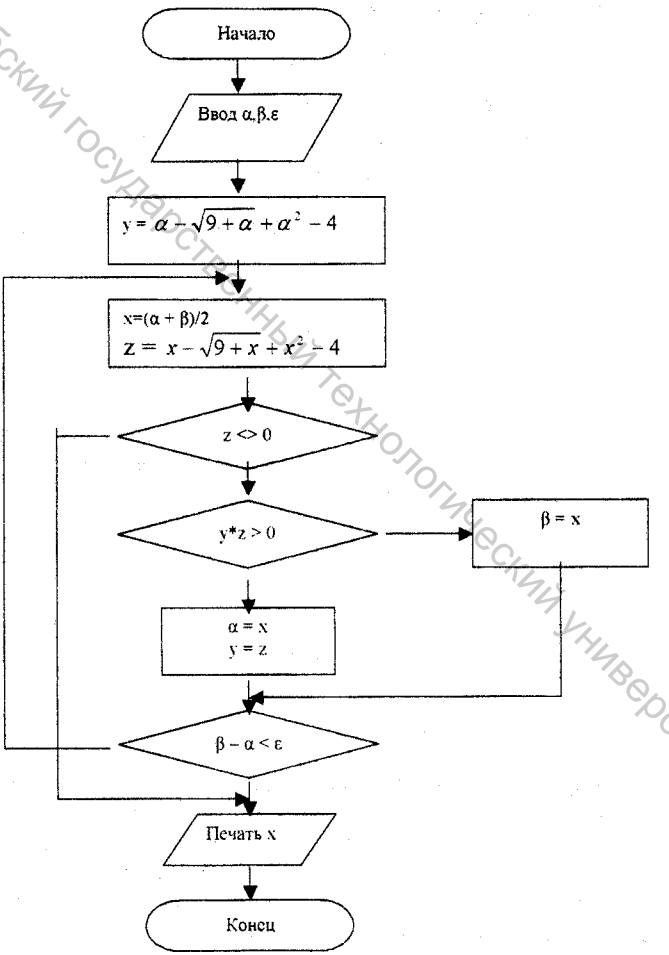


Рис. 36. Блок-схема решения уравнения (12) методом половинного деления

ЛЕКЦИЯ 20. ВЫЧИСЛЕНИЕ ОПРЕДЕЛЕННЫХ ИНТЕГРАЛОВ

Довольно часто в научно-технических задачах возникает необходимость вычисления определенного интеграла или значений первообразной функции. Вспомним некоторые-определения.

1. Функция *F(x)* в данном интервале называется первообразной функцией для функции *f*(x), если во всем этом интервале *f(x)* является производной для функции *F(x)*, т, е.

$$F'(x) = f(x).$$

2. Пусть функция *f (x)* задана на некотором отрезке [a, b]. Разобьем этот отрезок произвольным образом на части $\Delta x_i = x_{i+1} - x_i$ (i=0, 1, 2, ..., n - 1). Возьмем в каждой из частей произвольную точку ξI (i=0, 1, ..., n - 1). Тогда определенным интегралом $\int_a^b f(x)dx$ функции *f(x)* на отрезке от *a* до *b* называется

$$\lim_{\max \Delta x_i \to 0} \sum_{i=0}^{n-1} f(\xi_i)\Delta x_i .$$

Условия существования первообразной и определенного интеграла функции *f(x)* рассматриваются в курсе математического анализа. В курсе математического анализа доказываются предложения:

а) если *F(x)* есть первообразная функция для *f(x)* в интервале, включающем [*a*, *b*], то $\int_a^b f(x)dx = F(b) - F(a)$;

б) если существует $\int_a^b f(x)dx$, то одной из первообразных функций на [*a*, *b*] для *f(x)* является $\int_a^x f(t)dt$.

Таким образом, вычисляя первообразную функцию, мы можем вычислять определенный интеграл и наоборот. Но, как правило, выразить первообразную функцию через элементарные функции не удается. Поэтому приходится прибегать к приближенному интегрированию. Для решения этой задачи существует много численных методов, из которых мы рассмотрим два: метод трапеций и метод Симпсона.

Метод трапеций

Как известно, величина определенного интеграла $\int_a^b f(x)dx$ представляет собой площадь криволинейной трапеции, ограниченной графиком функции $f(x)$, осью абсцисс и двумя прямыми $x=a$ и $x=b$.

Разобьем отрезок интегрирования $[a, b]$ на n равных частей длины $h=(b-a)/n$. В точках разбиения $x_0 = a$, $x_1 = a+h$, ... , $x_n = b$ проведем ординаты y_0, y_1, ... , y_n до пересечения с кривой $y = f(x)$, т.е. $y_i = f(x_i)$, $x_i = a+ih$, i= 0, 1, ..., n. Концы ординат соединим прямолинейными отрезками. Тогда площадь криволинейной трапеции $aABb$ приближенно можно считать равной площади фигуры, ограниченной ломаной линией $aACD$ ... NBb. Площадь этой фигуры, которую мы обозначим через S, равна сумме площадей трапеций

$$S = h\left(\frac{y_0 - y_1}{2} + \frac{y_1 + y_2}{2} + \ldots + \frac{y_{n-1} + y_n}{2}\right) = \frac{b-a}{2n}\left(y_0 + 2y_1 + 2y_2 + \ldots + 2y_{n-1} + y_n\right).$$

Таким образом, приближенное значение интеграла по формуле трапеций запишется в виде

$$\int_a^b f(x)dx \approx \frac{b-a}{2n}\left(y_0 + 2y_1 + \ldots + 2y_{n-1} + 2y_n\right) \qquad (13)$$

При составлении программ целесообразно формулу трапеций (13) представить в виде

$$\int_0^{\frac{\pi}{2}} f(x)dx \approx \frac{b-a}{2n}\left(y_0 + y_n + \sum_{i=1}^{n-1} 2y_i\right),$$

Пример

Найти $\int_0^{\frac{\pi}{2}} \cos x dx$ методом трапеций, разбивая отрезок интегрирования на n=30 частей.

Для вычисления определенного интеграла используются величины

$h_1 = \frac{\pi/2 - 0}{30} = 60,$ — шаг интегрирования;

$x_i = ih_1$

$y_i = cos\, x_i$, i=0, 1 ,.., 30.

При разработке программы используем две подпрограммы-функции: в первой опишем интегрируемую функцию, во второй – метод трапеций.

При составлении блок-схемы и затем программы нет необходимости в использовании индексированной переменной xi, так как при вычислении очередного члена суммы требуется лишь одно значение x. Это значение может быть получено прибавлением шага h к предыдущему значению x.

Далее приводится программа, составленная в соответствии с приведенным алгоритмом и результат ее выполнения.

```
Program int_tr;
Uses crt;
Var a,b,int,x:real;
      n:integer;
(*********************************)
Function Fun(z:real):real;
 Begin
  Fun:=cos(z)
 End;
(*********************************)
Function Integral(a1,b1:real;n1:integer):real;
 Var i:integer;
    h1,x,s:real;
 Begin
  h1:=(b1-a1)/n1;
  s:=0;
  For i:=1 to n1-1 do
   Begin
    x:=a1+i*h1;
    s:=s+2*fun(x);
   End;
  Integral:=h1/2*(fun(a1)+fun(b1)+s);
 End;
(*********************************)
 Begin
  Writeln('Введите a,b,n');
  Readln(a,b,n);
  Int:=Integral(a,b,n);
  Writeln('Int=',int:7:4);
  Readln;
 End.
```

Результат:

Int = 0,9998

Метод Симпсона

Разобьем отрезок интегрирования $[a, b]$ на $2n$ равных частей длины $h=(b-a)/2n$. Пусть точкам разбиения $x_0 = a$, $x_1 = a + h, \ldots$, $x_n = b$ соответствуют значения подынтегральной функции $y_0, y_1, \ldots, y_{2n}$, т. е. $y_i = f(x_i)$, $x_i = a + ih$, $i = 0, 1, \ldots, 2n$. На отрезке $[x_0, x_2]$ подынтегральную функцию $f(x)$ заменим параболой, проходящей через точки (x_0, y_0), (x_1, y_1), (x_2, y_2). Уравнение параболы запишем в виде интерполяционной формулы Ньютона $f(x) = f(x_0) + \tilde{x}\Delta f(x_0) + \frac{\tilde{x}(\tilde{x}-1)}{2}\Delta^2 f(x_0) = y_0 + \tilde{x}\Delta y_0 + \frac{\tilde{x}(\tilde{x}-1)}{2}\Delta^2 y_0$,

где

$$\tilde{x} = \frac{x - x_0}{h}; \Delta y_0 = y_1 - y_0;$$

$$\Delta^2 y_0 = \Delta y_1 - \Delta y_0 = y_2 - 2y_1 + y_0;$$

Интегрируя полученное выражение, имеем (учитывая, что y_0, y_1, y_2 не зависят от x)

$$\int_{x_0}^{x_2} f(x)dx = \int_{x_0}^{x_2} \left\{ y_0 + (y_1 - y_0) + (y_2 - 2y_1 + y_0)\frac{\tilde{x}(\tilde{x}-1)}{2} \right\} dx = \frac{h}{3}(y_0 + 4y_1 + y_2),$$

Аналогично

$$\int_{x_2}^{x_4} f(x)dx = \frac{h}{3}(y_2 + 4y_3 + y_4),$$

.....

$$\int_{x_{2n-1}}^{x_{4n}} f(x)dx = \frac{h}{3}(y_{2n-2} + 4y_{2n-1} + y_{2n})$$

Сложив эти равенства, получаем формулу

$$\int_a^b f(x)dx \approx \frac{b-a}{6n}\left(y_0 + 4y_1 + 2y_2 + \ldots + 4y_{2n-1} + y_{2n}\right), \qquad (14)$$

известную под названием формулы Симпсона.

При составлении программ целесообразно формулу Симпсона (14) представить в виде

$$\int_a^b f(x)dx = \frac{h}{3}\left(y_0 + y_{2n} + \sum_{i=1}^{2n-1}(3+c_i)y_i \right)$$

где

$$c_i = \begin{cases} +1, \text{при } i \text{ нечетном} \\ -1, \text{при } i \text{ четном} \end{cases}$$

Пример

Найти $\int_0^{\pi/2} \sqrt{1-\frac{1}{4}\sin^2 x}dx$ методом Симпсона, разбивая отрезок интегрирования на $2n$=20 частей. Для вычисления определенного интеграла имеем

$h_1 = \frac{\pi/2-0}{2n} = \frac{\pi}{40}$,

$x_i = ih_1$,

$y_i = \sqrt{1-\frac{1}{4}\sin^2 x_i}$, i=0, 1, 2, …, 20.

По соображениям, изложенным в предыдущем примере, нет необходимости в использовании индексированных переменных c_i и x_i. Значения c и x, используемые при вычислении очередного члена, могут быть получены из предыдущих значений c и x рекуррентно, а именно:

$c=-c$, $x=a1+ih$.

Далее приводится программа, составленная в соответствии с приведенным алгоритмом, и результат ее выполнения.

```
Program int_Simp;
Uses crt;
Var a,b,int,x:real;
      n:integer;
(*********************************)
Function Fun(z:real):real;
Begin
 Fun:=sqrt(1-1/4*sqr(sin(z)))
End;
(*********************************)
Function Simpson(a1,b1:real;n1:integer):real;
 Var i,c:integer;
   h1,x,s:real;
 Begin
```

```
 h1:=(b1-a1)/n1;
 s:=0; c:=-1;
 For i:=1 to n1-1 do
  Begin
   x:=a1+i*h1;
   c:=-c;
   s:=s+(3+c)*fun(x);
  End;
 Simpson:=h1/3*(fun(a1)+fun(b1)+s);
End;
(************************************)
Begin
 Writeln('Введите a,b,n');
 Readln(a,b,n);
 int:=Simpson(a,b,n);
 Writeln('Int=',int:7:4);
 Readln;
End.
```

Результат:

```
Int=1,4674
```

ЛИТЕРАТУРА

1. Вальвачев, А. Н. Программирование на языке Паскаль для персональных ЭВМ ЕС / А. Н Вальвачев, В. С. Крисевич. – Минск : Выш. шк. 1991, - 224 с.
2. Вычислительная техника и программирование /под ред. А. В. Петрова/. – Минск : Выш. шк., 1991, - 479 с.
3. Абрамов, В.Г. Введение в язык Паскаль / В. Г Абрамов, Н. П. Трифонов, Г. Н. Трифонова. - Москва : Наука, 1988, - 320 с.
4. Семашко, Г. Л. Программирование на языке Паскаль / Г. Л Семашко, А. И. Салтыков. – Москва : Наука, 1988 - 118 с.
5. Йенсен, К., ПАСКАЛЬ: руководство для пользователя и описание языка / К. Йенсен, Н. Вирт. – Москва : Финансы и статистика, 1982 – 312 с.
6. Грогоно, П. Программирование на языке Паскаль/ П. Грогоно - Москва : Мир, 1982 – 268 с.
7. Турчак Л. И Основы численных методов / Л. И. Турчак – Москва : Наука, 1987 – 328 с.
8. Демидович Б. П. Численные методы анализа / Б. П. Демидович, И. А. Марон. – Москва : Наука, 1967 – 198 с..

СОДЕРЖАНИЕ

Вардомацкая Елена Юрьевна

ОСНОВЫ ИНФОРМАТИКИ И ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

Курс лекций

В авторской редакции

Подписано в печать 14.06.2005. Формат 60x84 1/16. Бумага офсетная.
Гарнитура Таймс. Печать офсетная. Усл. печ. л. 7,00. Уч.-изд. л. 8,2
Тираж 163. Заказ 337.

Издатель и полиграфическое исполнение – учреждение образования
«Витебский государственный технологический университет»
Лицензия №02330/0133005 от 01.04.2004.
210035, Витебск, Московский пр-т, 72.