ЯЗЫКИ И МЕТОДЫ ПРОГРАММИРОВАНИЯ

Смольянинов Игорь Николаевич

ЯЗЫКИ И МЕТОДЫ ПРОГРАММИРОВАНИЯ

- **§ 1. ИСТОРИЯ РАЗВИТИЯ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ**
- Одной из самых революционных идей, приведших к созданию автоматических цифровых вычислительных машин, была высказанная в 20-х годах XIX века Бэббиджем мысль о предварительной записи порядка действий машины для следующей автоматической реализации вычислений - программе.
- И, хотя использованная Бэббиджем запись программы на перфокартах, придуманная для управления ткацкими станками французским изобретателем Жозефом Мари Жаккаром, технически не имеет ничего общего с современными приемами хранения программ в ЭВМ принцип здесь по-существу один. С этого момента начинается история программирования.

- АДУ ЛАВЛЕЙС, одну из немногих современников Чарльза Бэббиджа, кто сумел по достоинству оценить аналитическую машину, называют первым в мире программистом.
- Она теоретически разработала некоторые приемы управления последовательностью вычислений, которые используются в программировании и по сей день, описала одну из важнейших конструкций практически любого современного языка программирования - цикл.

- Революционным моментом в истории языков программирования стало появление системы кодирования машинных команд с помощью специальных символов, предложенной Джоном Моучли, сотрудником Пенсильванского университета.
- Система кодирования, предложенная Моучли, увлекла одну из сотрудниц его компании - Грейс Мюррей Хоппер, которая посвятила всю свою жизнь компьютерам и программированию.
- Она вспоминает, что стала «третьим в мире программистом первого в мире большого цифрового компьютера». Г.Хоппер доказала, чего она стоит как программист. Впоследствии она писала: «Я имела то преимущество, что изучала как технику, так и математику и знала, как работает машина от начала и до конца».

- При работе на компьютере «Марк-I» Г.Хоппер и ее группе пришлось столкнуться со многими проблемами и все, что ими придумано, было впервые.
- В частности, они придумали подпрограммы. Сейчас любой программист не задумываясь использует аппарат подпрограмм в любом языке программирования.
- И еще одно фундаментальное понятие техники программирования впервые ввели Г.Хоппер и ее группа - «отладка». Однажды жарким летним днем 1945 г. неожиданно произошла остановка компьютера «Марк-I». Обнаружилась неисправность одного реле, контакы которого были заблокированы мотыльком, залетевшим неизвестно каким образом в помещение вычислительного центра.
- Вспоминает Г.Хоппер: «Когда к нам зашел офицер, чтобы узнать, чем мы занимаемся, мы ответили, что очищаем компьютер от насекомых (debuging)». С тех пор термин «debuging» (отладка) используется в технических процессах тестирования неисправностей в компьютере, а также в системах программирования.

- На заре компьютерной эры машинный код был единственным средством общения человека с компьютером. Огромным достижением создателей языков программирования было то, что они сумели заставить сам компьютер работать переводчиком с этих языков на машинный код.
- В конце 40-х годов, до прихода Г.Хоппер в фирму Джона Моучли, последний создал систему под названием «Short Code», которая являлась примитивным языком программирования высокого уровня.
- В ней программист записывал решаемую задачу в виде математических формул, а затем, используя специальную таблицу, переводил символ за символом, преобразовывал эти формулы в двухлитерные коды.
- В дальнейшем специальная программа компьютера превращала эти коды в обычный машинный код. Система, разработанная Дж. Моучли, была по существу одним из первых примитивных интерпретаторов.

- Уже в 1951 г. Хоппер создала первый в мире компилятор и ею же был введен сам этот термин. Компилятор Хоппер осуществлял функцию объединения команд и в ходе трансляции производил:
- организацию подпрограмм,
- выделение памяти компьютера,
- преобразование команд высокого уровня (в то время псевдокодов) в машинные команды.
- «Подпрограммы находятся в библиотеке (компьютера), а когда вы подбираете материал из библиотеки - это называется компиляцией» - так она объясняла происхождение введенного ею термина.

- В 1954 г. группа под руководством Г. Хоппер разработала систему, включающую язык программирования и компилятор, которая в дальнейшем получила название MATH-MATIC. После удачного завершения работ по созданию MATH-MATIC Г.Хоппер и ее группа принялись за разработку нового языка и компилятора, который позволил бы пользователям программировать на языке, близком к обычному английскому.
- Необходимость появления подобной системы Хоппер объясняла следующим образом: «Существует много различных людей, которым нужно решать разные задачи. Некоторые из них связаны с обработкой символов, другие - с обработкой слов, и им нужны языки другого типа, а не наши попытки превратить их всех в математиков».

- В 1958 г. появился компилятор FLOW-MATIC. В отличие от ФОРТРАНа - языка для научных приложений - FLOW-MATIC был первым языком для задач обработки коммерческих данных.
- Работы в этом направлении привели к созданию языка КОБОЛ (COBOL - Common Business Oriented Language). Одним из основных консультантов при создании этого языка была Грейс Мюррей Хоппер.

- Середина 50-х годов характеризуется стремительным прогрессом в области программирования. Роль программирования в машинных командах стала уменьшаться. Начали появляться языки программирования нового типа, выступающие в роли посредника между машинами и программистами.
- Первым и одним из наиболее распространенных был Фортран (FORTRAN, от FORmula TRANslator - переводчик формул), разработанный группой программистов фирмы IBM в 1954 г. (первая версия).

- В середине 60-х годов сотрудники математического факультета Дартмутского колледжа Томас Курц и Джон Кемени создали специализированный язык программирования, который состоял из простых слов английского языка.
- Новый язык назвали «универсальным символическим кодом для начинающих» (Beginners All-Purpose Symbolic Instruction Code, или, сокращенно, BASIC). Годом рождения нового языка можно считать 1964 г.
- Сегодня универсальный язык Бейсик (имеющий множество версий) приобрел большую популярность и получил широкое распространение среди пользователей ЭВМ различных категорий во всем мире. В значительной мере этому способствовало то, что Бейсик начали использовать как встроенный язык персональных компьютеров, широкое распространение которых началось в конце 70-х годов.

- В начале 60-х годов все существующие языки программирования высокого уровня можно было пересчитать по пальцам, однако впоследствии их число достигло трех тысяч. Разумеется, подавляющая часть языков не получила сколько-нибудь широкого распространения; в практической деятельности используется не более двух десятков.
- Разработчики ориентировали языки на разные классы задач, в той или иной мере привязывали их к конкретным архитектурам ЭВМ, реализовывали личные вкусы и идеи. В 60-е годы были предприняты попытки преодолеть эту «разноголосицу» путем создания универсального языка программирования.

- Первым детищем этого направления стал PL/1 (Programm Language One), 1967 г. Затем на эту роль претендовал АЛГОЛ-68 (1968 г.). Предполагалось, что подобные языки будут развиваться и усовершенствоваться и вытеснят все остальные.
- Однако ни одна из этих попыток на сегодняшний день не увенчалась успехом (хотя PL/1 в усеченных версиях использовали многие программисты).
- Всеохватность языка приводила к неоправданной, с точки зрения программиста, сложности конструкций, неэффективности компиляторов.

- Языки программирования служат разным целям и их выбор определяется удобностью пользователя, пригодностью для данного компьютера и данной задачи. А задачи для компьютера бывают самые разнообразные: вычислительные, экономические, графические, экспертные и т.д.
- Такая разнотипность решаемых компьютером задач и определяет многообразие языков программирования. По всей видимости, в программировании наилучший результат достигается при индивидуальном подходе, исходящем из класса задачи, уровня и интересов программиста. Например,

- Например, Бейсик широко употребляется при написании простых программ; Фортран является классическим языком программирования при решении на ЭВМ математических и инженерных задач; язык Кобол (COBOL, от Common Business Oriented Language - общий язык, ориентированный на деловые задачи; создан в 1960 г.) был задуман основной язык для массовой обработки данных в сферах управления и бизнеса.
- Еще более специализированным является язык ЛОГО (от греческого logos - слово), созданный для обучения программированию школьников профессором математики и педагогики Сеймуром Пейпертом из Массачусетского технологического института.

- Обучаясь программированию на ЛОГО, дети задают простые команды, которые управляют игрушечной черепашкой, снабженной карандашиком. Отметим и еще достаточно популярный специализированный язык - Пролог (Prolog - PROamming in LOGic), разработанный как язык программирования для создания систем искусственного интеллекта.

- В конце 50-х годов плодом международного сотрудничества в области программирования явился Алгол (ALGOL, от ALGOrithmic Language - алгоритмический язык). Алгол предназначен для записи алгоритмов, которые строятся в виде последовательности процедур, применяемых для решения поставленных задач.
- Специалисты-практики восприняли этот язык далеко неоднозначно, но, тем не менее, его влияние на развитие других языков и теорию программирования оказалось весьма значительным.
- В нашей стране в те годы был создан под руководством Сергея Петровича Ершова транслятор Альфа, который представлял довольно удачную русифицированную версию Алгола. Впоследствии академик Ершов сыграл важнейшую роль в становлении в СССР школьной информатики.

- Развитие идеи Алгола о структуризации разработки алгоритмов нашло наивысшее отражение при создании в начале 70-х годов языка Паскаль швейцарским Никлаусом Виртом.
- Язык Паскаль первоначально разрабатывался как учебный, и, действительно, сейчас он является одним из основных языков обучения программированию в школах и вузах. Однако, качества его в совокупности оказались столь высоки, что им охотно пользуются и профессиональные программисты.
- Не менее впечатляющей, в том числе и финансовой, удачи добился джазист Филип Кан, француз, разработавший систему Турбо-Паскаль. Суть его идеи состояла в объединении последовательных этапов обработки программы - компиляции, редактирования связей, отладки и диагностики ошибок - в едином интерфейсе. Версии Турбо-Паскаля заполонили практически все образовательные учреждения, программистские центры и частные фирмы.

- Период с конца 60-х и до начала 80-х годов характеризуется бурным ростом числа различных языков программирования, сопровождавшим, как это ни парадоксально, кризис программного обеспечения. Этот кризис особо остро переживало ведомство США.
- В январе 1975 г. Пентагон решил навести порядок в хаосе трансляторов и учредил комитет, которому было предписано разработать один универсальный язык.
- На конкурсной основе комитет проработал сотни проектов и, когда стало ясно, что ни один из существующих языков не может их удовлетворить, принял два проекта для окончательного рассмотрения.

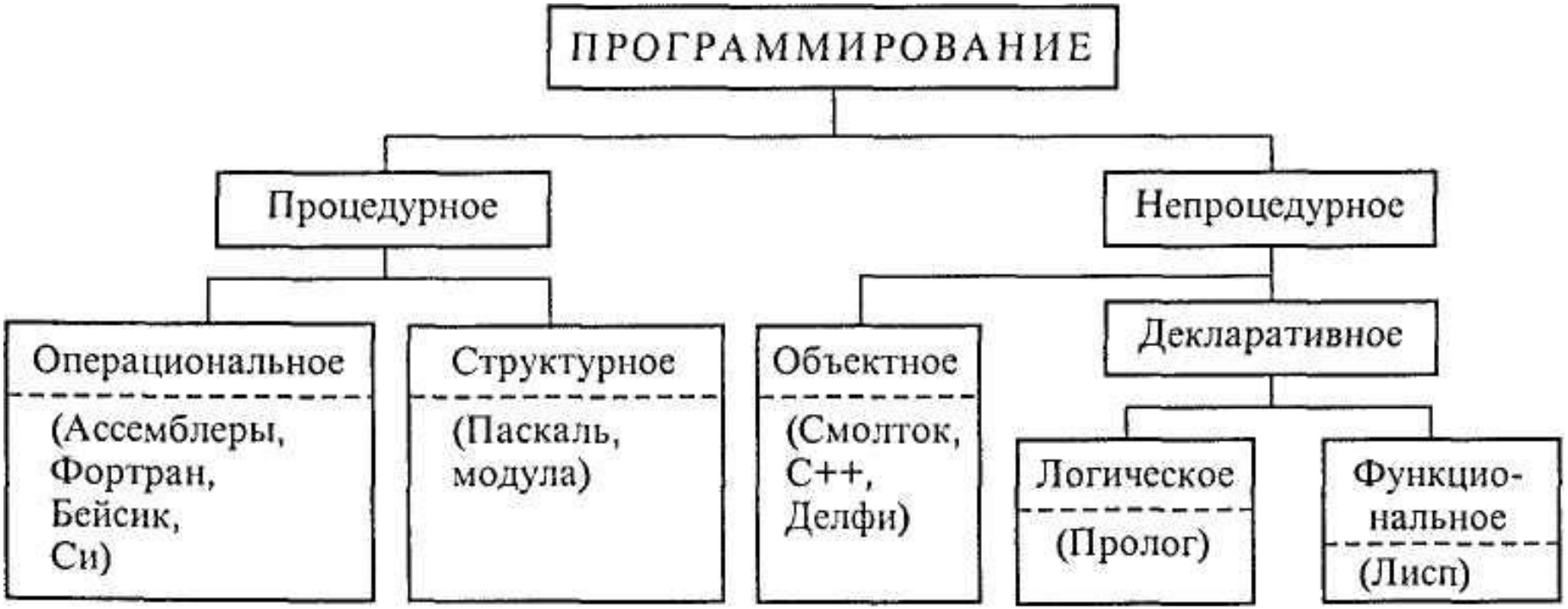
- В мае 1979 г. был объявлен победитель - группа ученых во главе с Жаном Ихбиа. Победивший язык окрестили АДА, в честь Огасты Ады Лавлейс.
- Язык АДА - прямой наследник языка Паскаль.
- Этот язык предназначен для создания и длительного (многолетнего) сопровождения программных систем, допускает возможность параллельной обработки, управления процессами в реальном времени и многое другое, чего трудно или невозможно достичь средствами более простых языков.

- Большой отпечаток на современное программирование наложил язык Си (первая версия - 1972 г.), являющийся очень популярным в среде разработчиков систем программного обеспечения (включая операционные системы).
- Си сочетает в себе черты как языка высокого уровня, так и машинно-ориентированного языка, допуская программиста ко всем машинным ресурсам, чего не обеспечивают такие языки, как Бейсик и Паскаль.
- Следует отметить, что многие языки, первоначально разработанные для больших и малых ЭВМ, в дальнейшем были приспособлены к персональным компьютерам.
- Хорошо вписались в «персоналки» не только Паскаль, Бейсик, Си, Лого, но и ЛИСП, ПРОЛОГ - языки искусственного интеллекта.

- В течение многих лет программное обеспечение строилось на основе операциональных и процедурных языков, таких как Фортран, Бейсик, Паскаль, Ада, Си.
- И сегодня современные версии этих и им подобных языков (Модула, Форт и др.) доминируют при разработке прикладных программных средств.
- Однако по мере эволюции языков программирования получили широкое распространение и другие, принципиально иные, подходы к созданию программ.

- Классическое операциональное и/или процедурное программирование требует от программиста детального описания того, как решать задачу, т.е. формулировки алгоритма и его специальной записи.
- При этом ожидаемые свойства результата обычно не указываются. Основные понятия языков этих групп - оператор и данные.
- При процедурном подходе операторы объединяются в группы - процедуры.
- Структурное программирование в целом не выходит за рамки этого направления, оно лишь дополнительно фиксирует некоторые полезные приемы технологии программирования.

- Принципиально иное направление в программировании связано с методологиями (иногда говорят «парадигмами») непроцедурного программирования.
- К ним можно отнести объектно-ориентированное и декларативное программирование. Объектно-ориентированный язык создает окружение в виде множества независимых объектов.
- Каждый объект ведет себя подобно отдельному компьютеру, их можно использовать для решения задач как «черные ящики», не вникая во внутренние механизмы их функционирования.
- Из языков объектного программирования, популярных среди профессионалов, следует назвать прежде всего Си++, для более широкого круга программистов предпочтительны среды типа Delphi и Visual Basic



Классификация языков программирования

- При использовании декларативного языка программист указывает исходные информационные структуры, взаимосвязи между ними и то, какими свойствами должен обладать результат. При этом процедуру его получения («алгоритм») программист не строит (по крайней мере, в идеале).
- В этих языках отсутствует понятие «оператор» («команда»). Декларативные языки можно подразделить на два семейства - логические (типичный представитель - Пролог) и функциональные (Лисп).
- По всей видимости, непроцедурные языки имеют большое будущее.
- Все сказанное выше можно отобразить схемой - крупноструктурной классификацией языков программирования. В ней указаны основные методологии программирования; в нижнем ряду, в скобках - типичные языки соответствующих групп.

- **§ 2. ЯЗЫКИ ПРОГРАММИРОВАНИЯ ВЫСОКОГО УРОВНЯ**
- Языки программирования - это формальные языки специально созданные для общения человека с компьютером. Каждый язык программирования, равно как и «естественный» язык (русский, английский и т.д.), имеет алфавит, словарный запас, свои грамматику и синтаксис, а также семантику.
- Алфавит - фиксированный для данного языка набор основных символов, допускаемых для составления текста программы на этом языке.
- Синтаксис - система правил, определяющих допустимые конструкции языка программирования из букв алфавита.
- Семантика - система правил однозначного толкования отдельных языковых конструкций, позволяющих воспроизвести процесс обработки данных.

- При описании языка и его применении используют понятия языка. **Понятие** подразумевает некоторую синтаксическую конструкцию и определяемые ею свойства программных объектов или процесса обработки данных.
- Взаимодействие синтаксических и семантических правил определяют те или иные понятия языка, например, операторы, идентификаторы, переменные, функции и процедуры, модули и т.д.
- В отличие от естественных языков правила грамматики и семантики для языков программирования, как и для всех формальных языков, должны быть явно, однозначно и четко сформулированы.

- Языки программирования, имитирующие естественные языки, обладающие укрупненными командами, ориентированными на решение прикладных содержательных задач, называют языками «высокого уровня».
- В настоящее время насчитывается несколько сотен таких языков, а если считать и их диалекты, то это число возрастет до нескольких тысяч. Языки программирования высокого уровня существенно отличаются от машинно-ориентированных (низкого уровня) языков.

- Во-первых, машинная программа в конечном счете записывается с помощью лишь двух символов 0 и 1.
- Во-вторых, каждая ЭВМ имеет ограниченный набор машинных операций, ориентированных на структуру процессора.
- Как правило, этот набор состоит из сравнительно небольшого числа простейших операций, типа: переслать число в ячейку; считать число из ячейки; увеличить содержимое ячейки на +1 и т.п.

- Команда на машинном языке содержит очень ограниченный объем информации, поэтому она обычно определяет простейший обмен содержимого ячеек памяти, элементарные арифметические и логические операции.
- Команда содержит код и адреса ячеек, с содержимым которой выполняется закодированное действие.

Языки программирования высокого уровня имеют следующие достоинства:

- алфавит языка значительно шире машинного, что делает его гораздо более выразительным и существенно повышает наглядность и понятность текста;
- набор операций, допустимых для использования, не зависит от набора машинных операций, а выбирается из соображений удобства формулирования алгоритмов решения задач определенного класса;
- конструкции команд (операторов) отражают содержательные виды обработки данных и задаются в удобном для человека виде;
- используется аппарат переменных и действия с ними;
- поддерживается широкий набор типов данных.

- Таким образом, языки программирования высокого уровня являются машинно-независимыми и требуют использования соответствующих программ-переводчиков (трансляторов) для представления программы на языке машины, на которой она будет исполняться.

МЕТАЯЗЫКИ ОПИСАНИЯ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ

- Интерпретация конструкций языка программирования должна быть абсолютно однозначной, ибо фраза на языке программирования превращается в машинный код автоматически, с помощью программы-транслятора, и любой намек на неоднозначность либо делает эту фразу непереводимой, либо приводит к ошибке.

- В этом отношении языки программирования значительно отличаются от естественных языков, допускающих неоднозначно интерпретируемые фразы, рассчитанные на здравый смысл и жизненный опыт человека - слушателя и исполнителя, способного додумать содержание фразы.
- «Додумывание» не входит в способности компьютеров, поэтому необходимы приемы описания конструкций языков программирования типа: «Оператором присваивания называется ...», причем продолжение подобной фразы на естественном языке чаще всего оказывается либо слишком громоздким, либо неоднозначным, либо и тем, и другим одновременно.

- Для строгого и точного описания синтаксиса языка программирования, как правило, используют специальные метаязыки (языки для описания других языков).
- Наиболее распространенными метаязыками являются металингвистические формулы Бэкуса - Наура (язык БНФ) и синтаксические диаграммы Вирта.

- Язык БНФ (называемый также языком нормальных форм) представляет компактную форму в виде некоторых формул, похожих на математические.
- Для каждого понятия языка существует единственная метаформула (нормальная форма). Она состоит из левой и правой частей. В левой части указывается определяемое понятие, а в правой - задается множество допустимых конструкций языка, которые объединяются в это понятие.
- В формуле используют специальные метасимволы в виде угловых скобок, в которых заключено определяемое понятие (в левой части формулы) или ранее определенное понятие (в ее правой части), а разделение левой и правой частей указывается метасимволом «::=», смысл которого эквивалентен словам «по определению есть».

Например, метаформулы

- <переменная>::=A|B
- <выражение>::=<переменная>|<переменная>+<переменная>|<переменная>-<переменная>

означают, что в том (сугубо модельном) языке, на который эта метаформула распространяется, под термином <переменная> понимается любая из букв A или B, а под термином <выражение> - любая из следующих десяти записей: A; B; A+A; A+B; B+A; B+B; A-A; A-B; B-A; B-B. Знак | следует читать «или».

Правая часть метаформулы может содержать правило построения допустимых последовательностей. Допускаются рекурсивные определения терминов и понятий, т.е. когда в правой части формулы участвует понятие, определяемое левой частью.

Например, пусть необходимо ввести понятие <двоичный код>, под которым понимается любая непустая последовательность цифр 0 и 1. Тогда простое и компактное рекурсивное определение с помощью метаформул выглядит так:

- <двоичная цифра>::= 0 | 1
- <двоичный код>::=<двоичная цифра>|<двоичный код> <двоичная цифра>

- Рекурсия здесь не мешает конструктивному построению понятия <двоичный код>, так как по принятым правилам при первом обращении к рекурсивно определяемому понятию следует ограничиться нерекурсивной частью формулы, т. е. под двоичным кодом понимать двоичную цифру - 0 или 1.
- Но при втором обращении к метаформуле, определяющей двоичный код, мы имеем варианты (конечно, неполные) понятия <двоичный код>, и можем применить рекурсию, которая даст нам следующие варианты этого понятия: 0 1 00 01 10 11, т.е. все возможные одно- и двухцифровые двоичные коды. Очевидно, что при следующих применениях рекурсии мы получим любой возможный двоичный код.

Для задания синтаксических конструкций произвольной длины часто используют фигурные скобки как метасимволы. Фигурные скобки означают, что конструкция может повторяться нуль или более раз. В частности, термин <двоичный код> можно определить по другому, а именно:

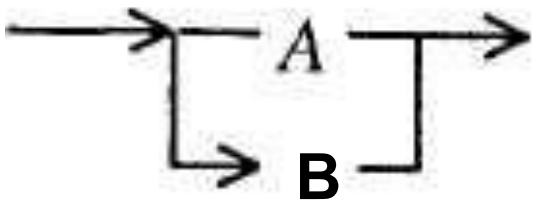
- <двоичный код>::=<двоичная цифра><двоичная цифра>

И еще, для полноты множества синтаксических конструкций, необходимо определить конструкцию <пусто>:

- <пусто>::=

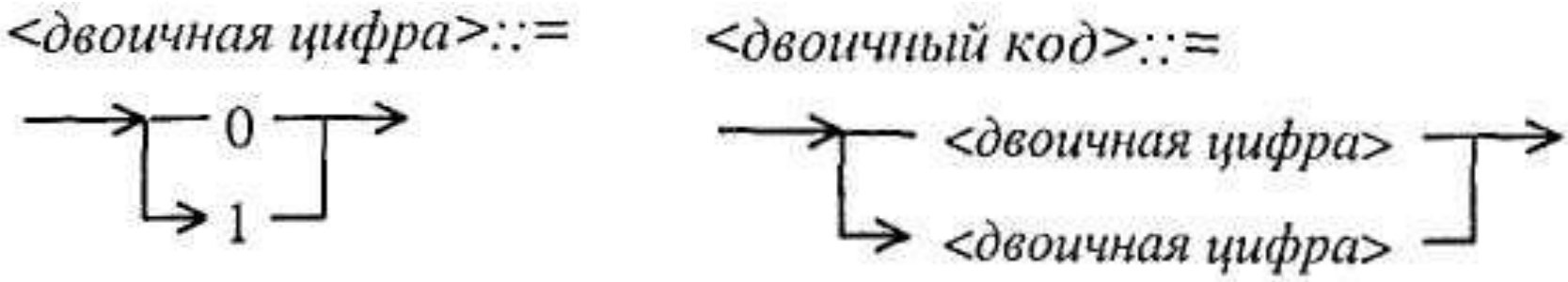
В большинстве учебных пособий по программированию, технических описаний языков, метаформулы рассматриваемого языка представлены полностью.

- **Синтаксическая диаграмма** является графическим представлением значения метапеременной метаязыка. Диаграмма состоит из основных символов или понятий языка.
- Каждая диаграмма имеет входящую и выходящую стрелки, означающие начало и конец синтаксической конструкции и отражающие процесс ее чтения и анализа . Из каждого элемента выходит одна или несколько стрелок, указывающих на те элементы, которые могут следовать непосредственно за данным элементом.
- Для сравнения с метаформулами приведем несколько примеров.
- Синтаксическая диаграмма
- *<переменная >::=*



- *эквивалентна метаформуле <переменная>::= A|B.*

- Еще примеры:



- Металингвистические формулы в некотором виде заложены в трансляторы; с их помощью ведется проверка конструкций, используемых программистом, на формальное соответствие какой-нибудь из конструкций, синтаксически допустимых в этом языке (синтаксический контроль).

ГРАММАТИКА ЯЗЫКОВ ПРОГРАММИРОВАНИЯ

- Описанию грамматики языка предшествует описание его алфавита. Алфавит любого языка состоит из фиксированного набора символов, однозначно трактуемых. Алфавит языков программирования, как правило, связан с литерами клавиатуры печатной машинки. Клавиатуры персональных компьютеров близки к ним по наличию литер.

- Алфавиты большинства языков программирования близки друг другу и основываются на буквах латинского алфавита, арабских цифрах и общепринятых спецсимволах, таких как знаки препинания, математических операций, сравнений и обозначений. Большинство популярных языков программирования в своем алфавите содержат следующие элементы:

- <буква> : : = AaBbCcDdEeFf и т.д.
- <цифра> :: = 0 1 2 3 4 5 6 7 8 9
- <знак арифметической операции >:: = * / + -
- <разделитель> :: = . , ; : ( ) [ ] { } ‘ : =
- <служебное слово> :: = begin end if then else for next и т.д.
- <спецсимвол> :: = <знак арифметической операции> | <разделитель> | <служебное слово>
- <основной символ>::=<буква> | <цифра> | <спецсимвол>
- <комментарий>::=<любая последовательность символов>

- Несмотря на значительные различия между языками программирования, ряд фундаментальных понятий в большинстве из них схожи. Приведем часть этих понятий.
- Оператор - одно из ведущих понятий всех языков программирования (теоретически, за исключением чисто декларативных; но в действительности и они используют родственное понятие).
- Каждый оператор представляет собой законченную фразу языка и определяет однозначно трактуемый этап обработки данных.

- В соответствии с теорией алгоритмов выделяют основные (базисные) операторы языка: присвоения, условный и безусловный переход, пустой оператор.
- К производным, не основным, относят составной оператор, оператор выбора, оператор цикла и оператор присоединения.

Все операторы языка в тексте программы отделяются друг от друга явными или неявными разделителями, например:

S1;S2;...;Sn

Операторы выполняются в порядке их следования в тексте программы. Лишь с помощью операторов перехода этот естественный порядок может быть нарушен.

- Большая часть операторов ведет обработку величин. **Величины** могут быть **постоянными** и **переменными.** Значения постоянных величин не изменяются в ходе выполнения программы. Величина характеризуется **типом, именем** и **значением.** Наиболее распространенные типы величин - числовые (целые и вещественные), символьные, логические. Тип величины определяется ее значением.

- Другая важная классификация величин - **простые** и **структурированные.** Простая величина в каждый момент может иметь не более одного значения. Ей соответствует одна ячейка памяти (поскольку термин «ячейка» несколько устарел, часто говорят «машинное слово») или ее эквивалент во внешней памяти компьютера.
- Структурированная величина, имея одно имя, может иметь разом несколько значений. Эти значения представляют собой элементы (компоненты) величины.
- Самый широкоизвестный пример - массив, у которого элементы различаются по индексам (номерам).
- Вопрос о структурировании величин - входных, выходных и промежуточных - для успеха решения прикладной задачи не менее важен, чем вопрос о правильном написании последовательности операторов.

- Важнейшие характеристики структурированной величины таковы: **упорядоченность** (да или нет), **однородность** (да или нет), **способ доступа** к элементам, **фиксированность числа элементов** (да или нет).
- Так, массив является упорядоченной однородной структурой с прямым доступом к элементам и фиксированным их количеством.

- Всем программным объектам в языках даются индивидуальные **имена.** Имя программного объекта называют идентификатором (от слова «идентифицировать»).
- Чаще всего идентификатором является любая конечная последовательность букв и цифр, начинающаяся с буквы:
- *<идентификатор>::=<буква>* | *<идентифиттор>* | *<буква>*| *<идентификатор><цифра>*

- Как правило, в большинстве языков программирования в качестве идентификатора запрещается использовать служебные слова языка.
- Многим слово «идентификатор» не нравится, и в настоящее время чаще употребляют слово «имя», поскольку
- <имя>::=<идентификатор>.

- Программисты выбирают имена по своему усмотрению. Принципы выбора и назначения имен программным объектам естественны.
- Следует избегать мало выразительных обозначений, не гоняться за краткими именами.
- Имена должны быть понятны, наглядны, отражать суть обозначаемого объекта. Например,
- *Summa, Time, i,j, integral, init* и т. п.
- Некоторым идентификаторам заранее предписан определенный смысл и их называют стандартными, например, Sin - это имя известной математической функции.

- **Описания** или объявления программных объектов связаны с правилами обработки данных. Данные бывают разные и необходимо для каждого из них определить его свойства.
- Например, если в качестве данных выступает массив, то необходимо задать его размерность, границы индексов, тип элементов массива.
- Описательная часть языка программирования является необходимой как для системных программистов - разработчиков трансляторов, которые должны, в частности, проводить синтаксическую и семантическую диагностику программ, - так и для «прикладного» программиста, которому объявления программных объектов часто облегчают процесс разработки и отладки программ.

- В некоторых языках стандартные описания простых числовых и символьных данных опускают (описания по умолчанию), или в них задаются правила описания по имени объекта.
- Например, в Фортране переменные, имена которых начинаются с букв I, J, K, L, M, N, могут принимать целые значения (при отсутствии явного описания типа, которое возможно), т.е. определены как числовые данные целого типа.
- В Бейсике-MSX данные строкового типа присваиваются переменным, имена которых заканчиваются специальным символом $: A$, S1$.

- Особый интерес представляют в языках программирования описания нестандартных структур данных, таких как запись, файл, объект, список, дерево и т.п.
- Приведем список наиболее употребительных обозначений типов данных, используемых в описаниях:
- *Целый - Integer*
- *Вещественный - Real*
- *Логический - Boolean*
- *Символьный - Char*
- *Строковый - String*
- *Массив -Array*
- *Множество - Set*
- *Файл - File*
- *Запись - Record*
- *Объект - Object*

- **Переменные** играют важнейшую роль в системах программирования. Понятие «переменная» в языках программирования отличается от общепринятого в математике.
- Переменная - это программный объект, способный принимать некоторое значение с помощью оператора присваивания. В ходе выполнения программы значения переменной могут неоднократно изменяться.
- Каждая переменная после ее описания отождествляется с некоторой ячейкой памяти, содержимое которой является ее значением. Синтаксис переменной, точнее, ее идентификатора, как правило, имеет вид:

- Семантический смысл переменной заключается в хранении некоторого значения, соответствующего ее типу (например, переменная целого типа может принимать значение произвольного целого числа), а также в выполнении с ней операций пересылки в нее и извлечения из нее этого значения.

- **Функция** - это программный объект, задающий вычислительную процедуру определения значения, зависимого от некоторых аргументов.
- Вводится в языки программирования для задания программистом необходимых ему функциональных зависимостей.
- В каждом языке высокого уровня имеется в наличии библиотека стандартных функций: арифметических, логических, символьных, файловых и т.п.
- Функции -стандартные и задаваемые программистом - используются в программе в выражениях.

- **Выражения** строятся из величин - постоянных и переменных, функций, скобок, знаков операций и т.д.
- Выражение имеет определенный тип, определяемый типом принимаемых в итоге его вычисления значений.
- Возможны выражения арифметические, принимающие числовые значения, логические, символьные, строковые и т.д.
- Выражение 5 + 7 является, несомненно, арифметическим, выражение A + B может иметь самый разный смысл - в зависимости от того, что стоит за идентификаторами A и B.

- **Процедура** - это программный объект, представляющий некоторый самостоятельный этап обработки данных.
- По сути, процедуры явились преемниками подпрограмм, которые были введены для облегчения разработки программ еще на самых ранних стадиях формирования алгоритмических языков.
- Процедура имеет входные и выходные параметры, называемые формальными. При использовании процедуры формальные параметры заменяются на фактические.

- **Модуль** (Unit) - это специальная программная единица, предназначенная для создания библиотек и разделения больших программ на логически связанные блоки.
- По сути, модуль - это набор констант, типов данных, переменных, процедур и функций. В состав модуля входят разделы: заголовок, интерфейс, реализация, инициализация.
- *Заголовок* необходим для ссылок на модуль.
- *Интерфейс* содержит объявления, включая процедуры и функции.
- Раздел *«реализация»* содержит тела процедур и функций, перечисленных в интерфейсной части.
- Раздел *«инициализация»* содержит операторы, необходимые для инициализации модуля.
- Каждый модуль компилируется отдельно, и каждый элемент модуля можно использовать в программе без дополнительного объявления.

§ 3. ПАСКАЛЬ КАК ЯЗЫК СТРУКТУРНО-ОРИЕНТИРОВАННОГО ПРОГРАММИРОВАНИЯ

- Язык Паскаль, начиная с момента своего создания Н. Виртом в 1971 г., играет особую роль и в практическом программировании, и в его изучении. С непревзойденной четкостью в нем реализованы принципы структурного программирования. Паскаль стал первым языком, с которым знакомится большинство будущих программистов в мире.
- Трансляторы для программ, написанных на Паскале, разработаны для различных компьютеров и в настоящее время имеют множество разновидностей. Они являются компиляторами, обрабатывающими разработанные программистами тексты программ.

- Существует много версий языка Паскаль. Различия между ними порой весьма велики. Так, базовая версия Вирта имеет многократно меньшие возможности чем, скажем, версия Турбо-Паскаль 7.0 (первая, фактически - язык для обучения будущих программистов, а вторая - орудие профессиональных разработчиков прикладного программного обеспечения).
- Тем не менее, это версии одного языка, что, в частности, подтверждается их совместимостью «сверху вниз», т.е. любая программа, соответствующая «младшей» версии, соответствует и «старшей» (за исключением малозначащих синтаксических оговорок).
- Приведенные ниже тексты программ и примеры соответствуют (если нет специальных оговорок) версиям не ниже Турбо-Паскаль 3.0.

- Любая Паскаль-программа является текстовым файлом с собственным именем и с расширением .pas. Рассмотрим в качестве примера текст программы 1 решения квадратного уравнения.
- Паскаль-программа имеет вид последовательности символов латинских и русских букв, арабских цифр, знаков операций, скобок, знаков препинания и некоторых дополнительных символов.
- В ней можно выделить описания данных и операторы, описывающие действия, которые надо выполнить машине над этими данными.

- *Программа 1*

```
program KvadUravn;                {заголовок программы}
var                          {список переменных}
a,b,c: real;                      {коэффициенты уравнения}
d,x1,x2: real;                    {вспомогательные переменные}
begin                             {начало программы}
writeln;                          {пропуск строки на экране}
writeln('введи a,b,c'); read(a,b,c);     {ввод данных}
d:=b*b-4*a*c;                          {дискриминант}
if d<0 then                       {если d<0, то}
write (‘корней нет’)                   {печатать}
else                              {иначе}
begin                             {начало серии команд}
x1:=(-b+sqrt(d))/(2*a);
x2:=(-b-sqrt(d))/(2*a);                {вычисляем корни}
write('x1=', x1, 'x2=', x2)            {печать корней}
end                          {конец серии}
end.                              {конец программы}
```

- Схематически программа представляется в виде последовательности восьми разделов:

1. заголовок программы;
2. описание внешних модулей, процедур и функций;
3. описание меток;
4. описание констант;
5. описание типов переменных;
6. описание переменных;
7. описание функций и процедур;
8. раздел операторов.

- Не в каждой программе обязательно присутствуют все восемь разделов, в простейшей программе, например, могут быть только 5-й и 8-й разделы.

- Каждый раздел начинается со служебного слова, назначение которого зафиксировано в Паскале так, что его нельзя употреблять для других целей.
- Рассмотрим пример программы 2, вычисляющей длину окружности и площадь круга по данному радиусу.
- *Программа 2*

```
program circle;
const
pi=3.14159;
var
r,s,l : real;
begin
writeln('введите радиус');
readln(r);
s:=pi*r*r; l:=2*pi*r;
writeln('площадь круга',s:8:4);
writeln('длина окружности',l:8:4)
end.
```

- В этой программе можно выделить четыре раздела.
- Описание заголовка начинается со служебного слова program, описание констант - const, описание переменных - var, раздел операторов начинается с begin. Программа заканчивается служебным, словом end, после которого ставится точка.
- Описания величин и операторы друг от друга отделяются знаком «точка с запятой».
- Для обозначения величин используются имена. Они составляются из латинских букв и цифр, причем первым символом должна быть буква.
- В примере использованы имена величин - pi, r, s и l.
- Имя программы (в примере - circle) выбирается автором и составляется по такому же правилу.
- Постоянные величины (константы) чаще всего бывают числовыми или символьными (но могут быть и других типов, о которых речь пойдет ниже).
- Значения символьных констант заключаются в апострофы.

- Постоянные величины описываются в разделе констант по схеме:
- const <имя> = <константа>
- В разделе констант может быть описано несколько постоянных величин. Например:
- const
- pi=3.14159; k=-15; S='площадь';
- Данные, обрабатываемые программой, могут быть разных типов (числа, символы, строки, массивы и т.д.). Тип определяет область допустимых значений, а также операции и функции, применяемые к величинам этого типа.
- В Паскале имеется несколько встроенных простых типов со стандартными именами.

- Группа типов, значения каждого из которых можно перечислить в некотором списке - **скалярные типы.**
- Для них определена порядковая функция ord(x) - номер значения x в списке (для целочисленного x ord(x)=x); функции pred(x) - значение в списке, предшествующее x, и succ(x) - значение в списке, следующее за x.
- **Упорядоченный тип** - это тип, значения которого упорядочены в обычном смысле. К данным такого типа применимы операции отношения <, >, <= (меньше или равно), >=(больше или равно), =, <> (не равно).

- Для логического типа выполняется неравенство:
- false < true
- Переменные описываются в разделе описания переменных по схеме: var <список имен переменных>: <тип>
- Имена в списке разделяются запятой. В этом разделе может быть описано несколько переменных разного типа, например:
- var a,b,c:real; k,l:integer; p:boolean;

- Над **целыми величинами** (тип integer) определены арифметические операции : * (умножение), div (деление нацело), mod (вычисление остатка от деления), + , - (сложение и вычитание); операции перечислены в порядке старшинства.
- Например: 25 div 4 = 6 ; 25 mod 4 = 1. Целый результат дают некоторые стандартные функции (аргумент функции заключается в круглые скобки):
- abs(x) - абсолютная величина целого x;
- sqr(x) - квадрат значения x;
- trunc(x) - целая часть вещественной величины x;
- round(x) - целое число, полученное из вещественного x по правилу округления;
- random(x) - случайное целое число из интервала от 0 до x.
- Например: trunc(4.7)=4 ; round(4.7)=5 ; sqr(3)=9 . Для данных типа byte определены те же операции и функции, что и для данных типа integer.

- Над **вещественными величинами** определены операции: *, +, -, /, а также стандартные функции при вещественном или целом аргументе: abs(x), sqr(x), sin(x), cos(x), arctan(x), ln(x), exp(x), sqrt(x) - квадратный корень из x, int(x) - целая часть из x, random - случайное число из интервала от 0 до 1. Указанные операции и функции дают вещественный результат.
- Над **логическими величинами** определены операции: not - отрицание, and - конъюнкция, or - дизъюнкция. Логическая функция odd(x) принимает значение true, если целочисленное x является нечетным, и false , если четным.
- Множество всех символов образуют **символьные величины** (тип char), которые являются упорядоченными, причем 'A' < 'B'< 'C' <...< 'Z', 'a' < 'b' <...< 'z', '0' < '1' <...< '9'.

- **Выражения** - это конструкции, задающие правила вычисления значений переменных. В общем случае выражения строятся из переменных, констант, функций с помощью операций и скобок.
- Эта роль выражений отражена в основном операторе языка – **операторе присваивания.** Он имеет следующий вид:
- <имя переменной> **:=** <выражение>
- Тип переменной и тип выражения должны быть согласованы (величины принадлежат одному и тому же типу). Есть исключение: имя переменной может относиться к типу real, а значение выражения - к типу integer.

- *Примеры.*
- l:=2*pi*r; p:=(a+b+c)/2; z:=sqrt(sqr(x)+sqr(y))
- В Паскале можно вводить с клавиатуры числовые и символьные данные. Имеются две встроенные процедуры (подпрограммы) ввода:
- 1) read(<список переменных>);
- 2) readln(< список переменных>).
- При выполнении процедуры read(x1,x2,…,xN) программа прерывается и компьютер ждет ввода с клавиатуры N значений переменных из списка x1, x2, ..., xN. Эти значения - константы соответствующих типов - должны при вводе разделяться пробелами. Набор данных завершается клавишей ввода.

- Процедура readln отличается от read только тем, что при завершении ввода курсор перемещается в начало следующей строки. *Пример.*
- var a,b:real; c:char; d:integer;
- ……..
- read(a, c, d, b) ;
- ……..
- Допустимый ввод: 83.14 k 200 -7.15
- Программа на Паскале может выводить на экран или на принтер значения числовых или символьных выражений. Имеются две процедуры вывода на экран:
- 1) write(<список выражений>);
- 2) writeln(<список выражений>).

- Процедура write(x1,x2,...,xN) печатает на экране значения выражений из списка x1, x2, ..., xN. Процедура writeln отличается от write тем, что переводит курсор в начало следующей строки.
- Для вывода на принтер используются те же процедуры с добавлением служебного слова lst перед списком выражений.
- Пример: write(lst,‘нет решений');
- На бумаге будет напечатан текст «нет решений».

- Для управления печатью используются форматы данных. Пусть x - переменная типа real. Если не использовать форматы, то значение x будет выводиться в «плавающей» форме (типа 1.654887892E-04). Форматы позволяют напечатать вещественное число в естественной форме.
- Пусть m, n - целые числа. Процедура write(x:m:n) выводит на экран значение переменной x в виде десятичной дроби, причем m определяет общее число выводимых символов, включая цифры, точку и знак числа, n - количество цифр после точки. Если количество выводимых символов меньше m, то перед числом добавляются пробелы.

- Пусть, например, x = 387.26. Следующая таблица демонстрирует влияние форматов на вывод значения x:
- оператор строка вывода
- writeln('*',x) * 3.8726000000E+02
- writeln('*',x:8:3) * 387.260
- writeln('*',x:8:1) * 387.3
- Один формат - ширину поля вывода - можно использовать и для вывода значений выражений типов integer, boolean, char.

- 3.2. ОСНОВНЫЕ КОНСТРУКЦИИ ЯЗЫКА
- Паскаль - язык структурного программирования. Это означает, что программист должен выражать свои мысли очень дисциплинированно, с использованием того числа четко оговоренных конструкций, используя как чередование их, так и вложения друг в друга.
- Не рекомендуется (хотя и возможно) использовать оператор перехода goto.
- Реализация последовательности действий (т. е. структуры следования) выполняется с помощью составного оператора:
- begin <последовательность операторов> end

- Раздел операторов в программе всегда является составным оператором. Служебные слова ***begin*** и ***end*** часто называют операторными скобками.
- Для реализации **развилки** в Паскале предусмотрены два оператора: условный оператор и оператор варианта (выбора). Они предназначены для выделения из составляющих их операторов одного, который и выполняется.
- Структура и действие **условного оператора** таковы:

  if <логическое выражение>

  then <оператор 1> else <оператор 2>
- Условный оператор может быть неполным, т.е. не содержать часть «else <оператор 2>». В этом случае, если значение логического выражения равно false, условный оператор не вызывает никаких действий.

- *Пример:* составим программу, которая определяет длину общей части двух отрезков числовой оси, заданных координатами своих концов соответственно a, b и c, d (a < b, c < d).
- Если отрезки имеют общую часть, то левая координата общей части отрезков m равна максимальному из чисел a и c, а правая n - минимальному из чисел b и d.
- *Программа 3*

```
program cross;
    var a, b, c, d, m, n, l: real;
begin
    writeln('введите координаты концов отрезков');
    read(a,b,c,d);
    writeln;
    if a<c then m:=c else m:=a;
    if b<d then n:=b else n:=d;
    if m<n then l:=n-m else l:=0;
    writeln('длина общей части отрезков=',l:6:2)
end.
```

- Оператор варианта имеет следующую форму:
- case <выражение> of

  <список констант 1> : <оператор 1>;

  <список констант 2> : <оператор 2>;

  ……

  <список констант N> : <оператор N>

  end.
- Выражение, стоящее между служебными словами case и of, должно иметь значение ординального типа. Любой список констант может состоять из одной константы.

- Оператор варианта вычисляет значение выражения, записанного после case. Если его значение совпадает с одной из констант в некотором списке, то выполняется оператор, стоящий после этого списка. Если значение выражения не совпало ни с одной константой во всех вариантах, то оператор варианта ничего не делает.

- В качестве примера приведем программу, которая в зависимости от номера месяца выдает сообщение о времени года.
- ***Программа 4***

```
program seasons;
   var k:integer;
begin
   writeln('введите номер месяца');
   readln(k);
   case k of
    1,2,12:writeln('зима');
    3,4,5:writeln('весна');
    6,7,8:writeln('лето');
    9,10,11:writeln('осень')
   end
end.
```

- Для реализации циклов в Паскале имеются три оператора. Если число повторений известно заранее, то удобно воспользоваться оператором цикла с параметром.
- В других случаях следует использовать операторы цикла с предусловием (цикл «пока») или с постусловием (цикл «до»).

- **Цикл с предусловием** является наиболее мощным в Паскале. Другие операторы цикла можно выразить через него. Его форма такова:
- while <логическое выражение> do <оператор>
- Действие: вычисляется значение логического выражения. Если оно равно true, то выполняется оператор, после чего снова вычисляется значение логического выражения, в противном случае действие заканчивается.
- В качестве примера использования такого цикла решим следующую задачу.

- На склад привозят однородный груз на машинах различной грузоподъемности. На компьютер, управляющий работой склада, поступает информация о весе груза очередной машины.
- Составить программу подсчета количества машин, прибывших на склад до его заполнения, если вместимость склада не более 100 тонн.
- Введем обозначения: sum - сумма веса груза, хранящегося в этот момент на складе; num - число разгруженных машин; w - масса груза очередной машины.
- Вначале величины sum и num равны нулю. Цикл разгрузки продолжается, пока выполняется неравенство sum < 100.

- *Программа 5*

```
program store;
    var sum,w:real; num:integer;
begin
    num:=0;sum:=0;
     while sum<100 do
    begin
     writeln('введите вес груза машины');
     readln(w);
     sum:=sum+w;
     if sum<=100 then num:=num+1
     else writeln('груз уже не поместится')
    end;
    writeln('число разгруженных машин =',num:3)
end.
```

- Оператор цикла с постусловием имеет форму:
- repeat <последовательность операторов> until <логическое выражение>
- Действие: выполняется последовательность операторов. Далее вычисляется значение логического выражения.
- Если оно равно true, то действие заканчивается, в противном случае снова выполняется последовательность операторов и т.д.
- Решим предыдущую задачу, применяя цикл с постусловием. Цикл разгрузки заканчивается, если выполняется условие: sum > 100.

- *Программа 6*

```
program store2;
var
    sum,w:real;
    num:integer;
begin
    num:=0; sum:=0;
    repeat
     writeln('введите вес груза машины');
     readln(w);
     sum:=sum+w;
     if sum<=100 then num:=num+1
     else writeln('груз уже не поместится')
    until sum>=100;
writeln('количество разгруженных машин =',num:3)
end.
```

- Оператор цикла с параметром предусматривает повторное выполнение некоторого оператора с одновременным изменением по правилу арифметической прогрессии значения управляющей переменной (параметра) этого цикла. Оператор цикла параметром имеет две формы.
- Форма 1:
- for <параметр>:= <выражение 1> to <выражение 2> do <оператор>
- Параметр, выражение 1, выражение 2 должны быть одного ординального типа. Параметр в этом цикле возрастает. Действие эквивалентно действию следующего составного оператора:

```
begin
    <параметр>:=<выражение 1>;
    while <параметр> <= <выражение 2> do
    begin
      <оператор>;
      <параметр>:=succ(<параметр>)
    end
end.
```

- Если в этом описании отношение <= заменить на >= , а функцию succ на pred, то параметр в цикле будет убывать, в этом случае цикл с параметром принимает форму 2.
- Форма 2:
- for <параметр>:=<выражение 1> downto <выражение 2> do <оператор>
- *Пример:* составим программу, по которой будет напечатана таблица перевода километров в мили (1 миля = 1,603 км). Параметром цикла можно считать целую величину k - количество километров. Пусть эта величина изменяется от 1 до 10 (с шагом 1, разумеется).

- *Программа 7*

```
program mili;
    const a=1.603; b='км'; c='мили';
var k:integer; m:real;
begin
    writeln(b:5,c:7); writeln;
     for k:=1 to 10 do
           begin
           m:=k/a; writeln(k:5,m:6:3)
           end
end.
```

- Запишем в этой программе цикл с параметром в форме 2:

```
for k:=10 downto 1 do
    begin
    m:=k/a; writeln(k:5,m:6:3)
    end.
```

- Тогда значения к в таблице будут убывать от 10 до 1 с шагом 1.

СТРУКТУРЫ ДАННЫХ

- Мы уже познакомились с простыми типами real, integer, boolean, byte, char. В Паскале программист по своему желанию может определить новый тип путем перечисления его элементов - перечисляемый тип, который относится к *простым ординальным типам.*
- Описание перечисляемого типа выполняется в разделе типов по схеме:
- type <имя типа> = <список имен>
- *Примеры:*
- type operator = (plus,minus,multi, divide) ;
- color = (white,red,blue,yelow,purple,green);
- В списке должно быть не более 256 имен.

- Поскольку перечисляемый тип относится к ординальным, то к его элементам можно применять функции ord(x), pred(x), succ(x) и операции отношения.
- Отметим, что данные этого типа не подлежат вводу и выводу с помощью функций ввода/вывода и могут использоваться внутри программы для повышения ее читабельности и понятности.
- Для иллюстрации работы с перечисляемыми типами приведем программу 8, переводящую английские названия дней недели на русский язык.

- *Программа 8*

```
program week;
    type days=(mon,tue,wed,thu,fri,sat,sun);
var d:days;
begin
    for d:=mon to sun do
    case d of
     mon:writeln('понедельник');
     tue:writeln('вторник');
     wed:writeln('среда’);
     thu:writeln('четверг');
     fri:writeln('пятница');
     sat:writeln('суббота');
     sun:writeln('воскресенье')
    end
end.
```

- **Интервальный тип** - это подмножество другого уже определенного ординального типа, называемого *базовым.*
- Интервал можно задать в разделе типов указанием наименьшего и наибольшего значений, входящих в него и разделяющихся двумя последовательными точками, например:

```
type days=(mon,tue,wed,thu,fri,sat,sun);
workdays=mon..fri;
index=1..30;
letter='a'..'z';
```

- Можно задать интервал и в разделе переменных:

```
var a:1..100; b:-25..25;
```

- Операции и функции - те же, что и для базового типа. Использование интервальных типов в программе позволяет экономить память и проводить во время выполнения программы контроль присваиваний.
- *Пример:* если к - номер месяца в году, то вместо описания

  var k:integer;

  можно написать

  var k:1..12;
- Интервальный тип тоже относится к простым ординальным типам.

- СОСТАВНЫЕ СТРУКТУРЫ.
- Данные, с которыми имеет дело ЭВМ, являются абстракциями некоторых реальных или существующих в воображении людей объектов.
- Мы уже познакомились с некоторыми типами данных: стандартными, перечислимыми, интервалами.
- Этими типами можно было обойтись при решении простых задач. Однако естественно и часто очень удобно группировать однотипные данные в последовательности - массивы, строки символов, объединять разнотипные данные об одном и том же объекте в виде записей.

- Значительные удобства представляются пользователю в Паскале при организации однотипных величин в виде множества с соответствующим набором операций: объединения, пересечения и т.д.
- Наконец, последовательность однотипных величин переменной длины можно представить в Паскале в виде файла данных и хранить на внешних носителях, используя его в разных программах.
- Таким образом, подобно составному оператору, содержащему несколько других операторов, составные типы образуются из других типов, при этом существенную роль играет метод образования или структура составного типа.

- Часто используемый составной тип - массив.
- *Массив* - это последовательность, состоящая из фиксированного числа однотипных элементов. Все элементы массива имеют общее имя (имя массива) и различаются индексами. Индексы можно вычислять, их тип должен быть ординальным.
- При описании массивов используются служебные слова ***array*** и ***of***.
- В описании массива указывается тип его элементов и типы их индексов.
- Схема описания такова:
- type <имя типа> = array [<список типов индексов>] of <тип элементов>

- Тип элементов - произвольный, он может быть составным.
- Число типов индексов называется размерностью массива.
- После описания типа массива конкретные массивы можно задать в разделе описания переменных.
- Например:

```
type vector = array [1..10] of real;
 table = array ['A'..'Z',1..5] of integer;
var    a,b:vector;
 c:table;
```

- Обращение к элементу массива осуществляется с помощью задания имени переменной, за которым следует заключенный в квадратные скобки список индексов элемента.
- Например:
  a[7]:=3.1; b[k*k+1]:=0; c['M',3]:=-14;
- Если массивы имеют одно и то же описание, то во многих версиях Паскаля допустимо их копирование, например b:=a.
- Описание массива можно совместить с описанием соответствующих переменных:
- var a,b:array [1..10] of real;
  d:array [byte] of char;

- В Турбо-Паскале разрешена инициализация начальных значений составных перемененных с помощью, так называемых, типизированных констант.
- Типизированные константы используются как переменные того же типа.
- Их применение экономит память, однако они не могут быть использованы для определения других переменных того же типа.
- Схема описания констант массива:

  const <имя массива>:<тип массива>= (<список значений элементов>)
- Тип массива может быть описан ранее:
- type digits=array[1..5] of char;
- const a:digits=(‘0’,’2’,’4’,’6’,’8’);

- *Пример:* используя массив, составим программу, которая напечатает на экране 20 чисел Фибоначчи.
- Последовательность Фибоначчи определяется равенствами

  a[1]=a[2]=1;

  a[к]=a[к-1]+a[к-2] при к>2.
- Использование массива позволяет создать эффективную программу.
- Для вывода каждого члена последовательности отведем на экране 5 позиций.

- *Программа 9*

```
program fibon;
   const n=20;
   var a:array[1..n]  of integer;
    k:integer;
begin
   a[1]:=1; a[2]:=1;
   for k:=3 to n do a[k]:=a[k-1]+a[k-2];
   for k:=1 to n do write(a[k]:5);
writeln
end.
```

- Рассмотрим часто встречающуюся задачу упорядочения членов числовой последовательности по какому-либо признаку.
- *Пример:* упорядочить члены числовой последовательности по возрастанию.
- Используем метод упорядочения, носящий имя «пузырек».
- Будем просматривать пары соседних элементов последовательно справа налево и переставлять элементы в паре, если они стоят неправильно:
- 5,3,2,4,1 -> 5,3,2,1,4 -> 5,3,1,2,4 -> 5,1,3,2,4 -> 1,5,3,2,4

- В начале просмотра присвоим некоторой логической переменной значение true: p:=true;
- если при просмотре пар была хотя бы одна перестановка, изменим значение логической переменной на противоположное: p:=false;
- это означает, что последовательность еще не была упорядочена и просмотр пар надо повторить.
- Цикл просмотров заканчивается, если после очередного просмотра выполняется условие: p:true.
- Последовательность зададим в программе как константмассив из 10 элементов - целых чисел.

- *Программа 10*

```
program bubble;
    uses crt;
    const a:array[1..10] of integer=(19, 8,17, 6,15, 4,13, 2,11, 0) ;
var b,i:integer; p:boolean;
begin
clrscr;
for i:=1 to 10 do write(a[i]:3); writeln; writeln;
    repeat
      p:=true;
      for i:=10 downto 2 do
      if a[i]<a[i-1] then
      begin
            b:=a[i]; a[i]:=a[i-1]; a[i-1]:=b; p:=false end
    until p=true;
    for i:=1 to 10 do write(a[i]:3) ;
    writeln;
    readkey;
end.
```

- Обработка элементов двумерных массивов (матриц) обычно выполняется с помощью двойного цикла.
- Один цикл управляет номером строки, другой - номером столбца.
- При решении задач на ЭВМ часто возникает необходимость в использовании последовательностей символов.
- Такую последовательность можно описать как массив символов, однако в Паскале для таких целей имеется специальный тип -string[n] - *строка* из n символов, где n<= 255.
- Способы описания переменных - строк - аналогичны описанию массивов.

- 1. Строковый тип определяется в разделе описания типов, переменные этого типа - в разделе описания переменных:

```
type word : string[20];
var    a,b,c : word;
```

- 2. Можно совместить описание строкового типа и соответствующих переменных в разделе описания переменных:

```
var a,b,c : string[20];
      d : string[30];
```

- 3. Можно определить строковую переменную и ее начальное значение как констант-строку:

```
const l:string[11]='информатика';
```

Символы, составляющие строку, занумерованы слева направо; к ним можно обращаться с помощью индексов, как к элементам одномерного массива.

- Для переменных одного строкового типа определен лексикографический порядок, являющийся следствием упорядоченности символьного типа:
- 'fife' < 'tree' (так как 'f ' < 't'); '4' > '237' (так как '4' > '2').
- Кроме логических операций <, >, =, для величин строкового типа определена некоммутативная операция соединения, обозначаемая знаком плюс:
- a:='кол'+'о'+'кол'; (в результате a='колокол').

- Для строковых величин определены следующие четыре стандартные функции.
- 1. Функция соединения - concat(s1,s2,...,sk). Значение функции - результат соединения строк s1,s2,...sk, если он содержит не более 255 символов.
- 2. Функция выделения - copy(s,i,k). Из строки s выделяется k символов, начиная с i-того символа:

  a:=copy('крокодил',4,3); (в результате a='код').
- 3. Функция определения длины строки - length(s). Вычисляется количество символов, составляющих текущее значение строки s:

  b:=length('каникулы'); (b=8).
- 4. Функция определения позиции - pos(s,t). Вычисляется номер позиции, начиная с которого строка s входит первый раз в строку t; результат равен 0, если строка s не входит в t:

  c:=pos('ом','компьютер'); (c=2).

- В Паскале определены также четыре стандартные процедуры для обработки строковых величин:
- 1. Процедура удаления delete(s,i,k). Из строки s удаляется к символов, начиная с i-того символа.

  s: = 'таракан'; delete(s, 5, 2) ; (в результате s=‘таран').
- 2. Процедура вставки - insert(s,t,i). Строка s вставляется в строку t, начиная с позиции i:

  t:=‘таран'; insert('ка',t,5); (t='таракан').
- 3. Процедура преобразования числа в строку символов - str(k,s). Строка s получается «навешиванием» апострофов на число к:

  str(564,s); (s=‘564’).
- 4. Процедура преобразования строки из цифр в число - val(s,k,i). Число i=0, если в строке s нет символов, отличных от цифр, в противном случае i=позиции первого символа, отличного от цифры:

  val('780',k,i); (k=780; i=0).

- Рассмотрим несколько программ, в которых используются строковые величины.
- 1. Составить программу, определяющую количество гласных в русском тексте, содержащем не более 100 символов.
- Здесь удобно определить констант-строку, состоящую из всех 18 строчных и заглавных русских букв, и в цикле проверить, будет ли очередной символ заданного текста элементом констант-строки.

- *Программа 11*

```
program vowel;
    const c:string[18] = 'аеиоуыэюяАЕИОУЫЭЮЯ';
    var a:string[100];
     k,n:integer;
begin
    writeln(‘введите текст’);
    readln(a);
    n:=0;
    for k:=1 to length(a) do
     if pos(a[k],c)>0
           then n:=n+1;
    writeln('кол. гласных=',n)
end.
```

- 2. Заменить в арифметическом выражении функцию sqr на exp. Замена выражения sqr на exp достигается последовательным применением процедур delete и insert:
- *Программа 12*

```
program stroka;
    var a,b:string[40];
     k:integer;
begin
    writeln('введите строку <= 40 символов');
    readln(a); b:=a;
    repeat k:=pos('sqr',b);
     if k>0 then
     begin
          delete(b,k,3);
          insert('exp',b,k);
     end
    until k=0;
writeln('старая строка=‘,a);
writeln('новая строка=',b);
end.
```

- 3. Ввести и упорядочить по алфавиту 10 латинских слов. В программе определим массив из 10 элементов-строк и упорядочим его элементы методом пузырька.
- *Программа 13*

```
program order;
    const s=10;
    type word=string[20];
    var i,j,k:1.. s; b:word; p:boolean; list:array[1..s] of word;
begin
    writeln ('введите список слов');
    for i:=1 to s do readln(list[i]);
    repeat p:=true;
      for i:=s downto 2 do
            if list[i]<list [i-1] then
            begin
            b:=list[i]; list[i]:=list[i-1]; list[i-1]:=b; p:=false
            end
    until p=true;
    writeln('упорядоченный список слов:');
    for i:=1 to s do writeln (list [i])
end.
```

- *Множество* в Паскале имеет такой же смысл, как и в алгебре - это неупорядоченная совокупность отличных друг от друга однотипных элементов.
- Число элементов множества не должно превышать 255. В качестве типа элементов может быть любой скалярный тип, кроме типа integer и его интервалов, содержащих числа > 255.
- Тип элементов множества называется базовым. При описании множественного типа используются служебные слова set и of.

- Задание конкретного множества определяется правилом (конструктором) - списком элементов или интервалов, заключенным в квадратные скобки.
- Пустое множество обозначается двумя символами [ ].
- Множественный тип можно определить в разделе описания типов по схеме:
- type <имя> = set of <тип элементов>;
- Например:

  type t=set of byte;

  var a : t ;

- Можно совместить описание множественного типа и соответствующих переменных:

  var code : set of 0..7;

  digits : set of ‘0’ . . ‘9’ ;

- Можно описать переменную множественного типа и задать ее первоначальное значение в разделе описания констант, как константмножество. Тип множества можно описать ранее, например,

```
type    up=set of 'A'    .    .    ' Z' ;
        low=set of 'a' . . 'z';
const  upcase : up=['A' . . 'Z' ];
        vocals :low=['a', 'e', 'i', ‘o', ‘u', ‘y'];
        delimiter:set of char=['   '../',':'   ..'?'];
```

- Для данных множественного типа определены операции объединения, пересечения и дополнения множеств, обозначаемые в Паскале соответственно знаками +, * и -, а также отношения равенства множеств (A=B), неравенства (A<>B), включения (A<=B,A>=B).
- Логическая операция принадлежности x in A принимает значение true, если элемент x принадлежит множеству A и false в противном случае.
- Так как к элементам множества прямого доступа нет, то операция in часто используется для этой цели.
- Заметим, что операции отношения на множествах выполняются быстрее, чем соответствующие операции на числах, поэтому их выгодно применять в программах.
- *Пример:* составить программу, анализирующую латинский текст и печатающую в алфавитном порядке все найденные в нем буквы, а затем все ненайденные.

- Пусть alfa- множество всех букв латинского алфавита. Будем вводить заданный текст с клавиатуры символ за символом, одновременно формируя множество E - множество латинских букв текста.
- В конце текста введем символ *. Затем с помощью операции in будем проверять, какие буквы алфавита имеются во множестве E.
- Множество N - ненайденных букв в тексте - определяется оператором: N := alfa - E .

- *Программа 14*

```
program search;
    const alfa:set of char=['a' . .'z'];
    var c:char; E,N:set of char;
begin
    clrscr;
    E:=[]; writeln('введите текст, конец ввода -*'); read(c);
    while c<>‘*’ do
    begin
      if c in alfa then E:=E+[c]; read(c)
    end;
    writeln;
    if E=alfa then writeln('найдены все латинские буквы')
    else begin
      N:=alfa-E;
      writeln('найдены:');
      for c:='a' to 'z' do if c in E then write(c);
      writeln; writeln(‘не найдены:');
      for c:='a' to 'z' do if c in N then write(c);
      writeln
    end
end.
```

- Переменные множественного типа удобно применять в задачах, где порядок данных не имеет значения, например при моделировании случайных событий.
- *Пример:* составить программу «спортлото 5 из 36», которая позволяет человеку ввести с клавиатуры пять натуральных чисел из интервала 1..36, затем генерирует случайным образом пять различных чисел из того же интервала и объявляет величину выигрыша по правилу: если угаданы человеком 0, 1 или 2 числа, объявляется проигрыш; если угаданы 3 числа, объявляется выигрыш 3 рубля; если угаданы 4 числа, объявляется выигрыш 100 рублей; если угаданы 5 чисел, объявляется выигрыш 1000 рублей.

- В программе используются обозначения: mn - множество натуральных чисел из интервала 1..36, a - множество чисел, задуманных человеком, x - множество чисел, генерируемых компьютером, z=a*x - пересечение множеств a и x; i, k, s - переменные, значения которых принадлежат интервалу 1..36.
- Случайное число из этого интервала генерируется оператором: s:=random(35)+1. Программа сначала выводит на экран сообщение о выигрышных номерах, затем определяет величину выигрыша.

- *Программа 15*

```
program lottery;
    type mn = set of 1 . . 36;
    var x,a, z: mn; i, k, s: 0 . . 36;
begin
    writeln; a: = [ ];
    for i:=1 to 5 do
    begin write('введите ',i,' -тое число '); readln(k); a:=a+[k] end;
    randomize; k:=0; x:=[];
    while k<5 do
    begin
      s:=random(35)+1;
      if not(s in x) then begin k:=k+1; x:=x+[s]
    end
    end; writeln;
    writeln('выигрыш выпал на следующие номера : ');
    for i:=1 to 36 do if (i in x) then write(i,' ');
    writeln; z:=a*x; k:=0;
    for i:=1 to 36 do if (i in z) then begin writeln('угадано: ',i); k:=k+1 end;
    case к of
    0, 1, 2 : writeln('вы проиграли ');
    3: writeln('получите 3 руб');
    4: writeln('получите 100 руб');
    5: writeln('получите 1000 руб')
    end
end.
```

- ***Записи*** (комбинированный тип) - одна из наиболее гибких и удобных структур данных, применяющихся при описании сложных объектов, которые характеризуются различными свойствами, а также при создании различных информационных систем.
- Запись - это последовательность, состоящая из фиксированного числа величин разных типов, называемых **полями** или **компонентами записи.**

- Так же, как и массив, запись содержит ряд отдельных компонент, но компонентами записи могут быть данные различных типов. Например, адресные данные (индекс, город, улица, номер дома, квартиры) можно представить как запись (record):

```
type address = record
    index          : string[6];
    city           : string[20];
    street         : string[20];
    haus,flat      : integer
end;
```

- Из примера видно, что тип «запись» описывается по схеме

```
type имя типа записи : record
    имя поля 1 : тип;
    имя поля 2 : тип;
    ……................
    имя поля N : тип
end;
```

- Как и при описании массивов, можно совместить описание типа записи и соответствующих переменных. Например, данные о двух студентах можно описать так:
- var stud1,stud2 : record

fio: string[20]; fac: string[10]; grup: string[8] end;

- Переменную типа «запись» и ее первоначальное значение можно определить как константа-запись в разделе констант по схеме

  const <имя>: <имя типа> = <константное значение>
- Константное значение - это список имен полей и соответствующих значений, заключенный в круглые скобки. Элементы списка разделяются знаком «точка с запятой». Например, запись о начале координат можно определить так:

```
type point = record
      x, y,  z : integer end;
const o: point = (x:0; y:0;  z:0);
```

- С компонентами записи можно обращаться как с переменными соответствующего типа. Обращение к компонентам записи осуществляется с помощью указания имени поля через точку.
- Пусть, например, переменная x имеет тип address, т.е. в программе имеется описание var x: address. Тогда допустимы следующие присваивания:

  x.haus := 52; x.street :='пр.Мира'; x.city:= 'Красноярск'; x.flat:=135; x.index :='660049'
- Проиллюстрируем работу с записями на задаче, в которой требуется найти сумму и произведение двух комплексных чисел:
- z1=a1+i*b1 и z2=a2+i*b2

- *Программа 16*

```
program compl;
type compl = record
       re : real;
       im : real
     end;
var z1,z2,s,p :compl;
begin
     writeln('компл.число a+i*b вводите двумя числами a и b: ');
     write(' введи 1 число: '); readln(z1.re,z1.im);
     write('введи 2 число: '); readln(z2.re,z2.im);
     s.re := z1.re  + z2.re;
     s.im := z1.im + z2.im;
     p.re := z1.re  * z2.re - z1.im * z2.im;
     p.im := z1.re  * z2.im + z2.re * zl.im;
     writeln('s=',s.re:4:2,' + i *',s.im:4:2);
     write('p=',p.re:4:2,' + i *',p.im:4:2)
end.
```

- Громоздкость обозначений в программе компенсируется большей наглядностью алгоритма за счет структуризации данных. Во многих случаях, если требуется производить операции с полями фиксированной записи, можно для сокращения обозначений использовать оператор присоединения with. Его структура такова:

  with <имя записи> do <оператор>;

- В этом случае в операторе, написанном после служебного слова do, имена полей указанной записи описываются без имени записи и точки. Например, печать суммы s в предыдущем примере можно организовать с использованием оператора with так:

  with s do writeln('s=',re:4:2,'+i*',im:4:2);

- В операторах присваивания разрешается использовать не только имена полей, но и имена записей.
- Тип поля может быть записью.

- Например:

```
man = record
    fio : record
     fam,  im,  otch : string[10];
     end;
    data :  record
     day :  1..31;
     mes:  1..12;
     god : integer
     end;
    pol : char;
    telef : record
     dom,rab :  string[10];
     end;
end;
```

- В Паскале разрешается использовать тип «запись» при описании других составных типов данных, например, можно построить массив записей.
- Рассмотрим пример эффективного использования записей в программе начисления стипендии студентам по шаблону:

№	ФИО	Эк1	Эк1	Эк1	БАЛЛ	сумма	проф	итого
1	Иванов И.И.	4	4	3	11	50.00	0.25	49.75

- Предположим, что вводится список группы с соответствующими оценками за экзамены. Графа «Балл» вычисляет суммарную оценку за семестр. Графа «Сумма» определяет размер стипендии по упрощенному правилу: если нет двоек и балл равен 15, то стипендия 75 руб.; при условии, что 12 < «Балл» < 15 стипендия 62 руб 50 коп., а если 9 < «Балл» < 12 , то - 50 руб. (в других случаях сумма равна нулю). В графе «Проф» указывается профсоюзный взнос в размере 0,5% от стипендии, а графа «Итого» определяет сумму денег к выдаче.
- В программе перед распечаткой итоговой ведомости можно предусмотреть упорядочение записей по убыванию в графе «Балл».

- *Программа 17*

```
program spisok;
type stud = record
       fio          :string[20];
       ex1, ex2, ex3:2 . . 5;
       bal          :6 . . 15;
       sum          :real;
       nalog              :real;
       itog         :real;
       end;
var x : array[1..3O] of stud;
i,k,m,n    : integer;
y : 6..15;
z : stud;
begin
     write('введи число студентов: '); readln(n);
     for i:= 1 to n do with x[i] do
     begin
       write('введи ФИО ',i,'-го студента: ');
       readln(fio); writet(‘введите его три оценки: ');
       readln(ex1,ex2,ex3);
     end;
```

```
for i:= 1 to n do with x[i] do
      begin bal:=ex1+ex2+ex3;
      if (ex1=2) or (ex2=2) or (ex3=2)
        then sum:=0
      else if bal=15 then sum:=75
      else if bal>12 then sum:=62.5
      else if bal>9 then sum:=50
      else sum:=0;
      nalog:=sum*0.005; itog:=sum-nalog;
      end;
for k:= 1 to n-1 do
      begin y:=x[k].bal; m:=k;
      for i:=k+1 to n do if y<x[i].bal then
        begin y:=x[i].bal; m:=i end;
         z:=x[k]; x[k]:=x[m]; x[m]:=z;
        end;
      writeln; writeln('СТИПЕНДИАЛЬНАЯ ВЕДОМОСТЬ  ');
      for i:=1 to 64 do write('-'); writeln;
        write ('  N   |   ФИО   | эк1 |   эк2   |   эк3   I   балл   |   сумма | проф | итого |');
      for i:=1 to 64 do write('-'); writeln;
      for i:=1 to n do with x[i] do
        begin write(i:3,fio:20,ex1:4,ex2:4,ex3:4);
        writeln(bal:5,sum:9:2,nalog:8:2,itog:7:2);
        end
end.
```

- 3.4. ПРОЦЕДУРЫ И ФУНКЦИИ
- Описание и вызов. В Паскале подпрограммы называются процедурами и функциями и описываются в разделе с тем же названием.
- Процедура имеет такую же структуру, как и программа, но с двумя отличиями:
- заголовок процедуры имеет другой синтаксис и включает служебное слово procedure;
- описание процедуры заканчивается точкой с запятой (а не точкой).
- Все имена, описанные в программе до процедуры, действуют во всей программе и в любой ее подпрограмме (если они там не описаны заново). Они называются глобальными, в отличие от локальных имен, описанных в процедуре и действующих лишь в ней.

- Данные для обработки могут передаваться процедуре через глобальные имена или через аргументы процедуры. В процедуре каждый аргумент имеет свое имя - формальный параметр, описываемый в заголовке процедуры по схеме

  procedure <имя> (<список описаний формальных параметров>)
- Описание формальных параметров может иметь вид

  <список имен>: <тип> или var <список имен>: <тип>
- В первом случае говорят о параметрах-значениях, во втором - о параметрах-переменных. В простейшем случае заголовок процедуры может содержать только имя процедуры.
- Оператор вызова процедуры имеет вид
- <имя процедуры> (<список выражений>);

- Указанные выражения называют **фактическими параметрами**.
- Их список должен точно соответствовать списку описаний формальных параметров процедуры.
- Во время вызова процедуры каждому параметру-значению присваивается значение соответствующего фактического параметра и поэтому их обычно используют для передачи входных данных.
- Параметры-переменные следует использовать для представления результатов процедуры.
- *Пример:* составим программу, которая с помощью строки символов разделит экран на части, где напечатает таблицу квадратных корней для чисел 1, 2,..., 10 и таблицу натуральных логарифмов для чисел 1, 2,..., 5.
- Печать строки символов оформим как процедуру. Так как никакую информацию передавать из процедуры в программу не надо, то аргументы процедуры (вид и количество символов) будут описаны как параметры-значения.
- Заметим, что процедура в программе выполняется пять раз.

- *Программа 18*

```
program section;
    var x:integer;
procedure line(a:integer; c:char) ;
    var j:integer;
begin
      for j:=1 to a do write(c);
            writeln
end;
begin
    line(35,'-'); writeln('таблица квадратных корней');
    line(35, '-');
    for x:=1 to 10 do writeln(x:8,sqrt(x):8:4);
    line(35,'-'); writeln('таблица натуральных логарифмов');
    line(35, '-');
    for x:=1 to 5 do writeln(x:8,ln(x):8:4);
    line(35, '*')
end.
```

- Функция - это подпрограмма, определяющая единственное скалярное, вещественное или строковое значение. Отличия подпрограммы-функции от процедуры:
- заголовок функции начинается со служебного слова function и заканчивается указанием типа значения функции:

  function <имя> (список описаний формальных параметров): <тип>;
- раздел операторов функции должен содержать хотя бы один оператор присваивания имени функции;
- обращение к функции - не оператор, а выражение вида

  <имя функции> (<список фактических параметров>).
- Функции (и процедуры) могут использовать свое имя в собственном описании, т.е. могут быть рекурсивными.

- *Пример:* составим программу, которая для заданных четырех натуральных чисел a, b, c, d напечатает наибольшие общие делители первой и второй пар чисел и сравнит их по величине.
- В программе определим **рекурсивную функцию** nod(x,y) по формулам

$$nod(x,y) = \begin{cases} x, & \text{если } y = 0 \\ nod(y,x), & \text{если } x < y \\ nod(x \bmod y, y), & \text{если } x > y \end{cases}$$

- Применяя эти формулы к числам 21 и 15, последовательно находим nod(21,15) = nod(6,15) = nod(15,6) = nod(3,6) = nod(6,3) = nod(0,3) = nod(3,0) = 3.

- *Программа 19*

```
Program four;
    var a,b,c,d,m,n:integer;
     function nod(x,y:integer):integer;
            var h:integer;
     begin
     if y=0 then h:=x
     else if x<y then h:=nod(y,x)
            else h:=nod(x mod y, y);
            nod:=h
     end;
begin
    writeln('введите 4 натуральных числа’);
    read(a,b,c,d); writeln;
    m:=nod(a,b); n:=nod(c,d);
    writeln ('нод(‘, a, ', ' ,b, ') =' ,m) ;
    writeln('нод(' ,c,',',d,')=',n);
    if m>n then writeln('первый > второго')
     else if m<n then writeln(' первый < второго')
            else writeln('нод пар равны')
end.
```

- **Внешние библиотеки.** Как известно, подпрограммы (процедуры и функции) используются в программах с целью их структурирования, а также при многократных повторениях некоторых частей программы.
- Процедуры и функции описываются в программных единицах в разделе описания подпрограмм. Они являются внутренними для этих программных единиц.
- Бывают случаи, когда одни и те же подпрограммы могут использоваться в различных программах одного и даже нескольких пользователей.
- В подобных ситуациях целесообразно создавать внешние подпрограммы, которые можно в необходимый момент подключать в любые программы.
- Как правило, внешние подпрограммы объединяют в отдельные пакеты, так называемые, библиотеки внешних подпрограмм. Могут создаваться личные библиотеки, специализированные библиотеки коллективного пользования и др.

- С одной из таких библиотек - встроенной библиотекой стандартных подпрограмм - пользователи имеют дело практически всегда.
- В состав этой библиотеки входят процедуры и функции вычисления значений ряда элементарных функций: синуса, косинуса, экспоненты и т.д., процедуры и функции обработки символьных величин, процедуры ввода-вывода и др.
- Встроенная библиотека подключается к любой программе автоматически при компиляции. Поэтому откомпилированный файл с расширением .com (иногда называемый «комовским»), как правило, занимает в 8 -10 раз больше места в памяти, чем исходный текст.

- Внешние подпрограммы создаются обычным образом в виде отдельного файла или файлов. Для подключения внешних подпрограмм в программе пользователя в разделе описания ставится директива $I имя файла.
- С этого момента все процедуры и функции внешнего файла становятся внутренними для программы, и на все входящие в него процедуры и функции распространяется правило локальных и глобальных переменных.
- В этой связи, директива подключения внешнего файла должна размещаться после описания всех ею используемых глобальных параметров, процедур и функций.
- *Пример.* Создадим внешнюю библиотеку из двух процедур и одной функции. Первая процедура программы 20 очищает экран, выдает приветствие, затем после нажатия клавиши <Пробел> снова очищает экран.
- Вторая процедура возводит число a в степень b. Третья подпрограмма-функция вычисляет значение экспоненты с некоторым грубым приближением на основе ряда Тейлора.

- *Программа 20*

```
procedure PRIVET;
    var a:char;
    begin
    clrscr; gotoxy(20,10);
    write('здравствуйте, желаю успехов!');
    repeat                      {цикл позволяет}
    gotoxy(35,50); write('пробел');     {сменить экран}
    read(a);                    {по нажатию клавиши}
    until a=' ';                        {* 'пробел' }
    clrscr;
    end;
procedure STEPEN(a,b:real;var y:real);
    begin
     y:=exp(b*ln(a));
    end;
function MEXP(x:real):real;
    begin
     mexp:=1+x+x*x/2+x*x*x/6+x*x*x*x/24;
    end;
```

- Пусть представленные три подпрограммы записаны в файл с именем lab.pas. А теперь составим программу, использующую созданную внешнюю библиотеку.
- *Программа 21*

```
program primer1;
uses crt;
{$i lab}     {директива подключения библиотеки}
var a,b : real;
begin
PRIVET;
STEPEN(2,4,a); writeln('2 в степени 4 =',a); b:=MEXP(1);
   write('машинная exp(1)=',EXP(1):6:4,' моя exp(1)=',b:6:4);
end.
```

- В программе используется стандартная функция - экспонента EXP(1) и наша подпрограмма MEXP(1).

- **Модули** используют в более поздних версиях Паскаля для создания библиотек и разделения больших программ на логически связанные независимые друг от друга составные части.
- В состав модуля входят следующие разделы: заголовок, интерфейс, реализация, инициализация. Заголовок необходим для ссылок на модуль. Интерфейс содержит объявления, включая процедуры и функции, представленные списком заголовков и доступные пользователям в теле основной программы.
- Раздел «реализация» содержит тела процедур и функций, перечисленных в интерфейсной части модуля. Раздел «инициализация» содержит операторы, необходимые для инициализации модуля.

- Таким образом модуль - это набор констант, типов данных, переменных, процедур и функций.
- Каждый модуль компилируется отдельно; результат компилляции - файл с расширением .tpu (Turbo Pascal Unit).
- Каждый элемент модуля можно использовать в программе пользователя без дополнительного объявления, для чего достаточно записать имя используемого модуля в директиве Uses в начале программы после его заголовка.

- В Турбо-Паскале версии 5.0 и выше применяют стандартные модули CRT, GRAPH и др. В этих модулях содержатся сервисные процедуры и функции по работе с экраном дисплея, с клавиатурой, графическими примитивами и т.п.
- Модули подключаются к программе путем специальной команды, размещаемой сразу после заголовка:
- uses <имя модуля>
- Программист может сам создать модуль. Ниже приведен пример с соответствующими комментариями.

- *Пример.* Создать модуль, дополняющий математические возможности Паскаля арифметическими действиями над комплексными числами.
- Будем представлять комплексные числа парами действительных: (a,b). Как известно, действия над ними выполняются по правилам
- (a,b) + (c,d) = (a+c,b+d),
- (a,b)-(c,d) = (a-c,b-d),
- (a,b) * (c,d) = (a*c-b*d , a*d+b*c),
- (a,b) / (c,d) = ((a*c+b*d)/(c*c+d*d), (b*c-a*d)/(c*c+d*d)).

- Создаваемый модуль будет включать четыре процедуры:
- Sum - сумма,
- Raz -разность,
- Proiz - произведение,
- Chastn - частное.
- Этот модуль может быть отдельно откомпилирован. После этого любая программа, написанная на Паскале, может получить доступ к интерфейсным объектам (в данном случае - процедурам) этого модуля с помощью директивы Uses CompChisla.
- Обратим внимание, что в интерфейсной части модуля от процедур присутствуют лишь заголовки, а в части «реализация» от заголовков процедур остаются лишь их имена.

- *Программа 22*

```
unit CompChisla;
interface
    procedure Sum(a,b,c,d: real; var x,y: real);
    procedure Raz(a,b,c,d: real; var x,y: real);
    procedure Proiz(a,b,c,d: real; var x,y: real);
    procedure Chstn(a,b,c,d: real; var x,y: real);
implementation
    procedure Sum;
      begin x:=a+c; y:=b+d
      end;
    procedure Raz;
      begin x:=a-c; y:=b-d
      end;
    procedure Proiz;
      begin x:=a*c-b*d; y:=a*d+b*c
      end;
    procedure Chstn;
      var z:real;
      begin z:= c*c+d*d; x:=(a*c+b*d)/z; y:=(b*c-a*d)/z
      end;
end.
```

- 3.5. РАБОТА С ФАЙЛАМИ
- Файл (последовательность) - это одна из наиболее фундаментальных структур данных. Программная организация компьютеров, их связь с внешними устройствами основаны на файловой структуре.
- Файлы позволяют решить две проблемы:
- возможность формирования и сохранения значений для последующего использования другими программами (например, в программах многократной обработки информационных систем, таких как платежные ведомости, различные АСУ, базы данных, необходимость длительного хранения информации очевидна);
- взаимодействие программ с внешними устройствами ввода-вывода: дисплеем, принтером, АСП и т.п.
- В Паскале эти проблемы снимаются с помощью структурированных данных файлового типа.
- Файловый тип данных в программе задается следующем образом:
- type <имя файлового типа> = file of <тип компонентов>

- В качестве типа компонентов файла разрешается использовать любой тип данных, кроме файлового.
- Например:

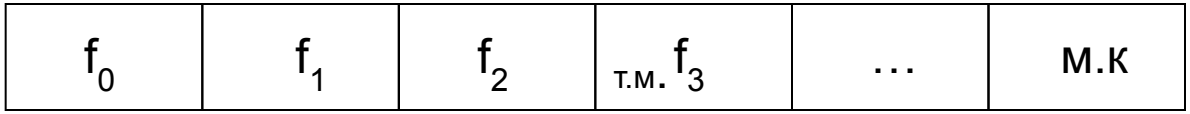
```
type
    intfile = file of integer;
    refile = file of real;
    chfile = file of char;
    ran =1 . . 10;
    st = set of ran;
    vector = array[ran] of real;
    compl = record
    re,im : integer;
end;
    setfile = file of st;
    vecfile = file of vector;
    compfile= file of compl;
```

- Описание файловой переменной задается обычным способом в разделе описаний. Например:
- var f: intfile; или var f : file of integer;

- Файловая переменная является буфером между Паскаль-программой и внешним устройством и должна быть логически с ним связана. Связь осуществляется оператором языка Паскаль:
- assign (<имя файловой переменной>,'<имя устройства>')
- Как правило, файлы для хранения данных связаны с устройством внешней памяти на магнитных носителях (дисковод) и носят название **внешние файлы.**
- Если, например, файл с именем primer.dat логически связан с дисководом A:, то все ДАННЫЕ, помещаемые в файл, будут храниться на этом дисковом накопителе, а установка «окна» между программой и файлом будет определяться через файловую переменную f оператором
- assign(f,’primer.dat’)

- Если внешним устройством является принтер, то связь осуществляется оператром assign(f, ‘lst:'). Здесь lst - логическое имя печатающего устройства.
- Ниже приведены логические имена внешних устройств ввода-вывода:
- con - консоль; trm - терминал; kbd - клавиатура; lst - принтер; aux - буфер сети; usr-драйвер пользователя.
- После осуществления связи файловая переменная f отождествляется с соответствующим файлом.
- Для работы с файлом его необходимо открыть, а по окончании работы - закрыть. Файл открывается для чтения оператором reset(f), для записи - оператором rewrite(f).
- Чтение и запись данных осуществляется известными командами read/write, только в начале списка помещается имя файловой переменной:
- read (f, <список ввода>); readln (f, <список ввода>); write(f, <список вывода>); writeln(f, <список вывода>).

- Закрытие файла осуществляется командой close(f).
- Условно файл можно представить в виде ленты, у которой есть начало, а конец не фиксируется. Компоненты файла записываются на эту ленту последовательно, друг за другом:



- Здесь т. м. - текущий маркер, указывающий на рабочую позицию (окно) файла; м.к. (маркер конца файла) - специальный код, автоматически формируемый вслед за последним элементом файла.
- Такого рода файлы называются **файлами** последовательного доступа. В исходной версии Паскаля **файлов прямого доступа,** для которых можно непосредственно «достать» любую компоненту, не предусмотрено; однако, в Турбо-Паскале элементы прямого доступа есть (например, через функцию seek, см. ниже).
- Команда rewrite(f) - открыть файл для записи - устанавливает файл в начальное состояние режима записи; текущий маркер устанавливается на маркер конца файла. Если в файле f до этого была информация, то она уничтожается.
- В открытом для чтения командой reset(f) файле текущий маркер устанавливается на нулевое состояние, однако содержимое файла не утрачивается.

- Команда закрытия файла close(f) обязательна, поскольку эта команда формирует маркер конца файла, что в большинстве случаев является необходимым (например, для работы с функцией eof(f), см. ниже).

В системе Турбо-Паскаль предусмотрены встроенные функции по работе с файлами:

- filesize(f) - текущее количество компонент открытого файла;
- filepos(f) - номер текущей позиции маркера;
- rename(f,имя) - переименование файла, связанного с f;
- erase(f) - уничтожение файла;
- execute(f) - выполнение COM-файла;
- chain(f)- выполнение CHN-файла;
- seek(f,N) - устанавливает маркер на позицию N;
- eof(f) - возвращает TRUE, если найден конец файла;
- ealn(f) - возвращает TRUE, если найден конец строки.

- На практике широко используются **текстовые файлы,** которые состоят из литерных (логических) строк.
- Поэтому в языке Паскаль предусмотрен стандартный файловый тип TEXT (он не является file of char, скорее всего, это - file of string[n]).
- Логические строки бывают разной длины, в том числе и нулевой. В конец каждой строки помещается специальный символ «конец строки» (eoln - «end of line»).
- В качестве печатного символа конца строки используют литеру #. Текстовый файл (text) является строго последовательным, к нему не применимы некоторые встроенные функции, в частности, seek.
- В отличие от типизированных файлов, с текстовым файлом нельзя одновременно проводить операции чтения (read) и записи (write). Однако, допустимы операторы writeln и readln. Числовые данные, целые и вещественные, в текстовом файле должны записываться через пробел.

- Ниже приведены несколько примеров, иллюстрирующих работу с файлами.
- *Пример 1.* Вывод данных на печатающее устройство - принтер (lst:).
- *Программа 23*

```
program print;
var
fal :text; x :real; name :string[25];
begin
    assign(fal,‘lst:’); rewrite(fal); x:=2.5;
    name:='Слава';
    writeln(fal,x:8:2) ;
    writeln(fal,'Привет,   ',name); close(fal)
end.
{Здесь файловая переменная fal связывается с
принтером lst:, и запись в файл fal практически
означает вывод на печать}
```

- *Пример 2.* Создание и сохранение в файле «xxx.dat» последовательности целых чисел от 10 до 20.
- *Программа 24*

```
program zapis; var
    f: file of integer; i: integer;
begin
    assign(f,'xxx.dat'); rewrite(f);
    for i:=10 to 20 do write(f,i); close  (f);
end.      {После выполнения программы формируется внешний файл
    xxx.dat}
```

- *Пример 3.* Считывание первых пяти компонент из существующего файла «xxx.dat» и вывод на дисплей квадратов этих значений.

Программа 25

```
program read;
    var ff: file of integer; j,i : integer;
begin
    assign(ff,'xxx.dat') ; reset(ff);
    for j:=1 to 5 do begin read(ff,i); writeln(i*i); end;
close(ff);
end.
```

- *Пример 4.* В текстовом файле (text) «slov.txt» содержится русский текст. Определить сколько гласных букв в тексте.
- *Программа 26*

```
program text;
   var ft : text; n :  integer; ch : char; st : set of char;
begin
   assign (ft,'slov.txt'); reset(ft);
   st :=  ['А', 'Е', 'И’, 'О’, 'У, 'Ы', 'Э', 'Ю’, 'Я'] ;
   st := st+[‘а’,'е','и','о’,'у','ы','э','ю’,'я'];
   n:=0;
   while not eof(ft) do
   begin
    read(ft,  ch);  if ch in st then n:=n+1;
   end;
   close(ft);
   writeln; write('кол-во гласных букв =',n);
end.
```

- Поскольку длина текста (файла) неизвестна, то в цикле «пока» используется логическая функция eof(f), которая возвращает значение TRUE, если найден конец файла.
- *Пример 5.* Шифрование и дешифрование текста.
- Программа шифрования заданного текста (sekret) использует следующее правило шифровки: каждая буква в тексте заменяется на букву, расположенную на n позиций вправо от искомой в русском алфавите. Причем последним символом алфавита является пробел ‘ ‘, а далее алфавит продолжается циклически.
- Значение смещения n находится во внешнем файле 'n.key', который формируется программой key. Зашифрованный текст выводится во внешний файл с именем «шифр.txt» , а также на дисплей.
- Программа дешифровки (retsek) считывает шифрованный текст из файла «шифр.txt) и выводит на экран дисплея искомый текст.

- *Программа 27 (а)*

```
program key;
var
n: integer; f: file of integer;
begin
assign (f,'n.key’);
rewrite(f);
write(‘введите ключ(смещение): ');
readln(n);
write(f,n); close(f);
end.
```

- *Программа 27 (б)*

```
program sekret;
     var
     slovo,anslovo: string[100];
     alfavit : string[33];
     n, i, k, p : integer;
     fkl : file of integer;
     fs : text;
begin
alfavit:='абвгдежзийклмнопрстуфхцчшщъыьэюя ';
assign(fkl,'n.key'); reset(fkl); read(fkl,n); close(fkl);
writeln; write('введи текст: ');
readln(slovo); anslovo:=' ';
for k:=1 to length(slovo) do
begin for i:=l to 33 do
     if slovo[k]=alfavit[i] then begin p:=i+n;
       if p >33 then p:=p mod 33;
       anslovo:=anslovo+alfavit[p];
     end;
end;
assign(fs,'шифр.txt'); rewrite(fs); write(fs,anslovo);close(fs);
writeln; write(anslovo);
end.
```

Программа 27 (в)

```
program retsek;
     var slovo, anslovo : string[100];
     alfavit : string[33];
     n,i,k,p : integer;
     fi : file of integer;
     f : text;
begin
alfavit:='абвгдежзийклмнопрстуфхцчшщъыьэюя ';
assign(fi,'n.key'); reset(fi); read(fi,n); close(fi);
assign(f,'шифр.txt'); reset(f); read(f,anslovo); close(f)
slovo:=‘ ';
for k:=1 to length(anslovo) do
begin for i:=1 to 33 do
     if anslovo[k]=alfavit[i]  then begin
     p:=i-n;  if p < 1 then p:=33-p mod 33;
     slovo:=slovo+alfavit[p];
     end;
end;
writeln; write('текст шифровки:   ',slovo);
end.
```

- 3.7. РАБОТА С ГРАФИКОЙ
- **Машинная (компьютерная) графика** - одно из важных направлений в современной прикладной информатике.
- В отличие от базового Паскаля, современные версии содержат мощные средства разработки графических программ.
- Рассмотрим часть соответствующих возможностей Турбо-Паскаля, в котором они реализованы с помощью стандартного модуля Graph.
- Модуль представляет собой мощную библиотеку графических подпрограмм универсального назначения, рассчитанную на работу с наиболее распространенными графическими адаптерами CGA, EGA, VGA, SVGA IBM-совместимых персональных компьютеров.

- Подключение модуля Graph.tpu к программе выполняется директивой
- uses graph;
- Инициализация графического экрана осуществляется с помощью процедуры Initgraph. Драйвер поддерживает тот или иной режим экрана, табл. 3.1.

Таблица 3.1
Некоторые сведения о драйверах и определяемых ими режимах

Адаптер	Драйвер	Режим (Номер, имя)	Разрешимость	Число страниц
EGA	EGA	0 Egalo	640x200	4
		1 Egahi	640x350	2
VGA	VGA	0 Vgalo	640x200	4
		1 Vgalo	640x350	2
		2 Vgalo	640x480	1

- Процедура инициализации в Турбо-Паскале имеет три аргумента:

Initgraph(<драйвер>,<режим>,'<путь к драйверу>')

Она может быть выполнена так:

```
uses graph;
   var gd, gm: integer;    {переменные gd и gm
определяют драйвер и режим}
begin
gd:=vga; gm:=vgahi;
initgraph(gd,gm,'d:\tp55');
……..
```

Первые две команды можно заменить одной:

```
gd:=detect
```

- Целая константа detect=0 в модуле Graph автоматически распознает драйвер и устанавливает режим максимального разрешения для данной машины.
- Процедура closegraph освобождает память от драйвера и устанавливает режим работы экрана, который был до инициализации графики.
- Для обнаружения ошибок в графике применяются функции graphresult и grapherrormsg (код ошибки).
- Последняя выдает строку сообщения о характере ошибки, соответствующей коду. Инициализация графического режима с проверкой ошибок может быть выполнена в программе следующим образом:

```
uses graph;
var gd, gm, errorcod: integer;
begin
   gd:=detect; initgraph(gd,gm,' ') ;
   errorcod:=graphresult;
   if errorcod<>ogrok then
   begin
   writeln ('ошибка графики');
   writeln(grapherrormsg(errorcod));
   halt
   end;
```

- Процедура Halt останавливает выполнение программы и возвращает управление операционной системе.
- Для формирования палитры используется система смешения красного, зеленого и синего цветов и изменения яркости луча. Цвет задается номером из списка цветов палитры в интервале 0 .. 15.
- Процедуры setcolor(<цвет>) и setbkcolor(<цвет>) устанавливают текущий цвет рисунка и цвет фона. При инициализации графики по умолчанию устанавливается черный фон и белый цвет рисунка.
- В табл. 3.2 указаны основные процедуры для модуля Graph, применяющиеся для построения простейших геометрических примитивов.

- Координаты точек воспринимаются в «экранной» системе координат, в которой начало - верхний левый угол экрана, ось «x» направлена вниз, ось «y» - направо.
- Максимальные значения координат определяются разрешимостью экрана (см. табл. 3.1).
- Первый аргумент процедуры setlinestyle(a,b,t) a - стиль линии второй параметр b - «образец» - имеет значение 4, если a=4, в остальных случаях b=0; третий параметр t - толщина линии - может иметь значение 1 (нормальная толщина) или 3 (жирная линия).

Таблица **3.2**
Основные процедуры модуля Graph

Заголовок процедуры	Геометрический смысл
putpixel(x,y,c)	Построить точку (x,y) цветом **c**
setlinestyle(a,b,t)	Установить стиль, образей и толщину линий
line(xl,yl,x2,y2)	Соединить две точки отрезком
rectangle(xl,yl,x2,y2)	Построить прямоугольник с заданными концами диагонали и сторонами, параллельными осям координат
circle(x,y,r)	Построить окружность с центром (x,y) и радиусом r
arc(x,y,a,b,r)	Построить дугу окружности: a,b - начальный и конечный угол в градусах
ellipse(x,y,a,b,rx,ry)	Построить эллиптическую дугу: rx, ry - полуоси эллипса
setfillstyle(t,c)	Установить стиль закраски и ее цвет
fillellipse(x,y,rx,ry)	Построить закрашенный эллипс, используя цвет рисунка
floodfill(x,y,cg)	Закрасить фигуру до границы с цветом cg; (x,y) - внутренняя точка фигуры
bar(xl,yl,x2,y2)	Построить столбец, используя тип и цвет закраски
pieslice(x,y,a,b,r)	Построить и закрасить сектор круга
sector(x,y,a,b,rx,ry)	Построить и закрасить эллиптический сектор
settextstyle(f,n,d)	Установить шрифт, направление вывода и размер символа текста
outtextxy(x,y,st)	Вывести строку st, начиная с точки (x,y)
outtext(st)	Вывести строку, начиная с точки расположения текущего указателя

- Первый аргумент процедуры setfillstyle(t,c) - тип закраски t - принимает значения из интервала 0..12. Наиболее употребителен тип t= 1 - заполнение фигуры текущим цветом.
- Для вывода текста на графический экран сначала выполняется процедура settextstyle(f,n,d), устанавливающая шрифт f, направление вывода n и размер символов (параметр d).
- При f = 0 используется стандартный точечный шрифт, встроенный в систему Турбо-Паскаль. С использованием других шрифтов познакомимся ниже. Направление вывода n принимает значения 0 (горизонтальный вывод) и 1 (вертикальный вывод).

- Размер букв определяется параметром d, принимающим значения из интервала 1.. 10. Если d = 1 и f = 0, то каждый символ занимает квадрат 8*8 точек, при d > 1 сторона квадрата умножается на d.
- Далее, с помощью процедуры outtextxy(x,y,st) строка st выводится на экран, начиная с точки (x,y).
- Например:
- settextstyle(0,0,2);
  outtextxy(100,200,'горизонтальная строка’);
  outtextxy(100,230,'размер увеличен вдвое');

- **Примеры графических программ**
- *Пример 1:* программа рисует звездное небо с 400 «звездами», вспыхивающими постепенно, и полную желтую луну.
- *Программа 29*

```
program sky;
    uses crt,graph;
    var k,gd,gm:integer;
begin
    gd:=detect;
    initgraph(gd,gm,‘ '); randomize;
    for k:=1 to 400 do
    begin
      putpixel(random(640),random(480),random(15)+1);
      delay(10)
    end;
    setfillstyle(1,14); setcolor(14); circle(550,80,30);
    floodfill(550,80,14); repeat until keypressed;
closegraph
end.
```