Государственное образовательное учреждение высшего профессионального образования

донской государственный технический университет

каф. «ПОВТ и АС»

Лекции по курсу

«Языки программирования высокого уровня»

Разработал

доц., к.т.н. Землянухин В.Н..

Ростов-на-Дону

2008

**Методические рекомендации по организации изучения дисциплины.**

Рабочей программой и учебным планом на изучение дисциплины отводится один семестр. В конце семестра, в качестве итогового контроля, предусмотрен экзамен.

В течение первых 5-ти недель студенты изучают основы алгоритмизации на псевдокоде, с помощью структурограмм Насси- Шнедермана, блок-схем. Затем типовые алгоритмы реализуются на Паскале (Object pascal).

В итоге студент должен изучить программирование стандартных типов данных, три вида циклов, условные операторы и оператор выбора, процедуры и функции. Должен знать приемы структурного программирования. Уметь писать, и отлаживать простые программы.

**Лекции.**

Лекция 3.1.1. Введение в программирование и языки см. в лекции 1. Остальные разделы лекции в 5.1.1: стр. 5-9. **(5.1.15)Иванова Г.С. Основы программирования: учебник для вузов. М.: Издательство МГТУ им. Баумана, 2004:** стр.12-24

Лекции 3.1.2. Алгоритмизация. 5.1.1: стр.10-53. в этом пособии достаточно подробно рассмотрены все вопросы алгоритмизации. И единственное издание, где подробно изложены схемы Насси- Шнейдермана. В учебнике (5.1.15): стр.12-24.

Лекции 3.1.3. Программирование на языке высокого уровня (Паскаль, Object Pascal). 5.1.1: стр. 82-98; (5.1.15): стр 28-49 здесь подробно вводятся нотации Бэкуса- Наура и синтаксические диаграммы.5.1.2: стр. 14-39.

Лекции 3.1.4 Концепция типов данных. 5.1.1: стр. 82-97; 5.1.2: 58-90; (5.1.15): 31- 40;

Лекции 3.1.5 Управляющие операторы языка. 5.1.1: стр.99-151; 5.1.2.: стр. 40-57; (5.1.15): стр.50-69;

Лекции 3.1.6 Структурированные типы данных. 5.1.1 : стр.186-201; 5.1.2.: стр. 60-80;( 5.1.15): стр. 77-136.

Лекции 3.1.7 Подпрограммы. 5.1.1 стр: 204-249; 5.1.2.: стр. 90-107;( 5.1.15): стр. 144-178.

Лекции 3.1. Файловый тип. Процедуры и функции обработки файлов 5.1.1 стр: 296-317; 5.1.2.: стр. 78-89;( 5.1.15): стр. 188-209.

Изучение курса в указанной последовательности и в объеме и содержании в приведенной литературе гарантирует качественное усвоение дисциплины. Для тех студентов, кто хочет получить дополнительные знания рекомендуется чтение дополнительной литературы 5.2.1.-5.2.7, а также периодической литературы: «Мир ПК», «Открытые системы», «Byte» и др. журналы.

**Лекция 1.Этапы решения задач на ЭВМ. Жизненный цикл ПО.**

     В основе деятельности по созданию и использованию программного обеспечения лежит понятие **жизненного цикла,** который является моделью создания и использования программного обеспечения, отражающей его различные состояния, начиная  с момента возникновения необходимости в данном программном изделии и заканчивая моментом его полного выхода из употребления у пользователей. Для каждого конкретного программного изделия количество, состав и последовательность этапов жизненного цикла определяется на ранних стадиях его создания. Непосредственный выбор определяется различными ограничениями и особенностями предметной области.

Решение любой задачи с использованием ЭВМ состоит из нескольких  
взаимосвязанных этапов, среди которых чаще всего выделяют следую-  
щие:

1. Техническое задание (постановка задачи).
2. Формализация (математическая постановка задачи).
3. Выбор (или разработка) метода решения.
4. Разработка алгоритма (алгоритмизация).
5. Выбор языка программирования.
6. Структура данных.
7. Оптимизация.
8. Подготовка отладки.
9. Тесты и методы «ручной» проверки (без использования ЭВМ).
10. Запись программы на конкретном языке программирования.
11. Тестирование и отладка.
12. Вычисления и обработка результатов.
13. Документирование.

Последовательное выполнение перечисленных этапов составляет полный цикл разработки, отладки и счета программы. Приведенное разделение является условным.

Рассмотрим более подробно некоторые наиболее общие и необходимые этапы.

*Постановка задачи.* При постановке задачи первостепенное внимание должно быть уделено выяснению конечной цели и выработке общегоподхода к исследуемой проблеме: выяснению, существует ли решение  
поставленной задачи и единственно ли оно; изучению общих свойстврассматриваемого физического явления или объекта; анализу возможно-стей конкретной ЭВМ и данной системы программирования. На этомэтапе требуется глубокое понимание существа поставленной задачи  
Правильно сформулировать задачу иногда не менее сложно, чем ее решить.

*Формализация.* Формализация, как правило, сводится к построениюматематической модели рассматриваемого явления, когда в результате

анализа существа задачи определяются объем и специфика исходныхданных, вводится система условных обозначений, устанавливается принадлежность решаемой задачи к одному из известных классов задачбирается соответствующий математический аппарат. При этом нужно уметь сформулировать на языке математики конкретные задачи физики, механики, экономики, технологии и т. д. Для успешного преодоленияэтого этапа требуются не только солидные сведения из соответствующейпредметной области, но и хорошее знание вычислительной математики,  
т. е. тех методов, которые могут быть использованы при решении задачина машине.

Полная постановка многих сложных задач невыполнима средствамивычислительной техники. Поэтому эти задачи нужно упрощать. Грамот-ное упрощение задачи невозможно без хорошего представления о том,какие факторы и параметры наиболее важны для изучаемой задачи, а какие менее существенны. При этом также очень важно знать, какаяиз возможных расчетных схем может привести к упрощениям вычислительного характера, обусловленным выбором вычислительного метода.Если имеющихся средств недостаточно, тогда необходимо разработать  
новый подход, новые методы исследования.

*Выбор метода решения.* После того как определена математическаяформулировка задачи, надо выбрать метод ее решения. Вообще говоряприменение любого метода приводит к построению ряда формули к формулировке правил, определяющих связи между этими формулами.Все это разбивается на отдельные действия так, чтобы вычислительныйпроцесс мог быть выполнен машиной. При выборе метода надо учитывать, во-первых, сложность формул и соотношений, связанных с тем илииным численным методом, во-вторых, необходимую точность вычислений и характеристики самого метода. На выбор метода решения большоевлияние оказывают вкусы и знания самого пользователя.

Этот этап - важнейший в процессе решения задачи. С ним связанымногочисленные неудачи, являющиеся результатом легкомысленногоподхода к ошибкам вычислений. При решении задачи на ЭВМ необходимо помнить, что любой получаемый результат является приближенным.Если известен алгоритм точного решения, то, кроме случайных ошибок(сбоев в работе ЭВМ), возможны ошибки, связанные с ограниченнойточностью представления чисел в ЭВМ. При вычислениях, заключающихся в нахождении результата с заданной степенью точности, возникает  
дополнительная погрешность, которую, если возможно, оцениваютна данном этапе (до выхода непосредственно на ЭВМ). Эта погрешностьопределяется выбранным численным методом решения задачи.

*Разработка алгоритма.* Данный этап заключается в разложении вычислительного процесса на возможные составные части, установлениипорядка их следования, описании содержания каждой такой части в тойили иной форме и последующей проверке, которая должна показать,обеспечивается ли реализация выбранного метода. В большинстве случаев не удается сразу получить удовлетворительный результат, поэтому составление алгоритма проводится методом проб и устранения ошибоки для получения окончательного варианта требуется несколько шаговкоррекции и анализа.

Как правило, в процессе разработки алгоритм проходит несколькоэтапов детализации. Первоначально составляется укрупненная схема ал-горитма, в которой отражаются наиболее важные и существенные связимежду исследуемыми процессами (или частями процесса). На последующих этапах раскрываются (детализируются) выделенные на предыдущихэтапах части вычислительного процесса, имеющие некоторое самостоятельное значение! Кроме того, на каждом этапе детализации выполняетсямногократная проверка и исправление (отработка) схемы алгоритма. Подобный подход позволяет избежать возможных ошибочных решений.

Ориентируясь на крупноблочную структуру алгоритма, можно быстрее и проще разработать несколько различных его вариантов, провестиих анализ, оценку и выбрать наилучший (оптимальный).

Эффект поэтапной детализации алгоритма во многом зависит от того,как осуществляется его структуризация: расчленение алгоритмическогопроцесса на составные части, что должно определяться не произволомпользователя (программиста), а внутренней логикой самого процесса.Каждый элемент крупноблочной схемы алгоритма должен быть макси-мально самостоятельным и логически завершенным в такой степени,чтобы дальнейшую его детализацию можно было выполнять независимоот детализации остальных элементов. Это упрощает процесс проектиро-вания алгоритма и позволяет осуществлять его разработку по частям од-новременно нескольким людям.

В процессе разработки алгоритма могут использоваться различныеспособы его описания, отличающиеся по простоте, наглядности, компактности, степени формализации, ориентации на машинную реализациюн другим показателям. В практике программирования наибольшее распространение получили:

1. Словесная запись алгоритмов.
2. Схемы алгоритмов.
3. Псевдокод (формальные алгоритмические языки).

4. Структурограммы (диаграммы Насси-Шнейдермана).

Разработка алгоритмов является в значительной степени творческим,эвристическим процессом, как правило, требует большой эрудиции, изобретательности, нестандартных и нетрадиционных подходов к решениюзадачи.

*Составление программы.* Представление алгоритма в форме, допускающей ввод в машину и последующий перевод на машинный язык, от-носится к этапу составления программы (программированию), т. е. разработанный алгоритм задачи необходимо изложить на языке, который будет  
понятен ЭВМ непосредственно или после предварительного машинногоперевода. От выбора языка программирования зависит процесс отладкирограммы, во время которого программа принимает окончательный рабочий вид. Таким языком может быть язык программирования Паскаль.

Этот язык был предложен в 1970 г. профессором Никлаусом Виртом из Цюриха (Швейцария). Он был назван в честь известного математика Блеза Паскаля, который изобрел один из первых калькуляторов. Языку Паскаль принадлежит особая роль в современном программировании:опираясь на результаты, полученные при разработке алгоритмическогоязыка Алгол-60, он заложил основы современной методологии программирования. Основной тезис его разработки - «язык должен быть очевидным и естественным отражением фундаментальных и наиболее важныконцепций алгоритмов». Широкое распространение языка Паскаль на всех современных ЭВМ свидетельствует о его высокой практической ценности в различных сферах применения.

*Отладка программы.* Составление программы представляет собойтрудоемкий процесс, требующий от исполнителя напряженного внимания. Практика показывает, что в вычислениях следует избегать поспеш-ости и придерживаться золотого правила: «лучше меньше, да лучше».Но на предыдущих этапах столько возможностей допустить ошибку, что,как бы мы тщательно ни действовали, первоначально составленная программа обычно содержит ошибки и машина или не может дать ответаили приводит неправильное решение.

Отладка начинается с того, что программа, аккуратно записаннаяна бланке, проверяется непосредственно лицом, осуществившим подго-товку и программирование задачи. Выясняется правильность написания  
программы, выявляются смысловые и синтаксические ошибки и т. п. Зам программа вводится в память ЭВМ и ошибки, оставшиеся незаме-ченными, выявляются уже непосредственно с помощью машины.

Опытный пользователь ЭВМ знает, что необходим действенный контроль над процессом вычислений, позволяющий своевременно обнару-

живать и предотвращать ошибки. Для этого используются различного рода интуитивные соображения, правдоподобные рассуждения и контрольные формулы. Начинающий пользователь часто считает отладкуизлишней, а получение контрольных точек - неприятной дополнительной работой. Однако очень скоро он убеждается, что поиск пропущеннойошибки требует значительно большего времени, чем время, затраченноена контроль.

Гарантией правильности решения может служить, например:

а)проверка выполнения условий задачи (например, для алгебраического уравнения найденные корни подставляются в исходное уравнениеи проверяются расхождения левой и правой частей);

б) качественный анализ задачи;

в) пересчет (по возможности другим методом).

Для некоторых сложных по структуре программ процесс отладкиможет потребовать значительно больше машинного времени, чем собст-венно решение на ЭВМ, так как плохо спланированные процессы алго-ритмизации, программирования и отладки приводят к ошибкам, которыемогут быть обнаружены лишь после многократных проверок.

*Вычисления и обработка результатов.* Только после того как появится полная уверенность, что программа обеспечивает получение правильных результатов, можно приступать непосредственно к расчетампо программе.

После завершения расчетов наступает этап использования результатов вычислений в практической деятельности или, как говорят, этап внедрения результатов. Интерпретация результатов вычислений снова относится к той предметной области знаний, откуда возникла задача.

**Лекция 2 Алгоритмы, Способы описания.**

  Под алгоритмом, как правило, понимается точное предписание, которое задает вычислительный процесс, начинающийся из некоторой совокупности исходных данных и направленный на получение полностью определяемого исходными данными результата. Обосновывать принципы алгоритмизации целесообразно в рамках анализа свойств алгоритмов. Выделяют следующие свойства алгоритмов.

1.   *Дискретность* – свойство, отражающее выполнение вычислительного процесса, задаваемого алгоритмом по шагам.

2.   *Детерминированность* – свойство, означающее, что на каждом шаге любой полученный результат определяется результатами, полученными на предшествующих шагах.

3.    *Элементарность* – свойство, согласно которому действие, которое выполняется на каждом шаге должно быть простым.

4.    *Направленность* – свойство, обуславливающее необходимость указания, что является результатом того шага, на котором нельзя выполнить определенную в нем операцию.

5.    *Массовость* – свойство, согласно которому алгоритм может быть применен к любой совокупности из допустимого множества исходных данных.

     Одной и той же задаче может соответствовать множество алгоритмов, поэтому, для обоснования выбора одного из них для решения конкретной задачи, применяют совокупность показателей, основными из которых являются следующие.

1.    Временная сложность – показатель, отражающий время выполнения алгоритма или количество шагов его выполнения.

2.    Емкостная сложность – показатель, по которому производят оценку количества данных, необходимых для выполнения алгоритма.

3.    Сложность описания – показатель, отражающий длину описания алгоритма, количество инструкций операторов.

     Для представления алгоритма используют различные средства, в том числе граф-схемы алгоритмов, каждая из которых представляет собой граф. Понятие элементарного графа дополнено введением надстройки, которая задает вычисления, каждому из которых соответствует путь в этом графе и последовательность меток узлов, лежащих на этом пути. Узлы графа, с помощью введения типизации, интерпретируются как действия, а дуги задают порядок передачи управления. Для каждого алгоритма существует одна начальная дуга и одна или множество конечных. В табл.1 представлены типы узлов граф-схемы алгоритма в соответствии с ГОСТ 19.701-90 «Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения».

Таблица 1

|  |  |  |
| --- | --- | --- |
| **символ** | **название** | **интерпретация** |
|  | данные | данные, носитель данных не определён |
|  | процесс | функция обработки данных любого вида (выполнение определённой операции или группы операций, приводящие к изменению значения, формы или размещения  информации или к определению, по которому из нескольких направлений потока следует двигаться) |
|  | предопределённый процесс | предопределённый процесс, состоящий из одной или нескольких операций или шагов, которые определены в другом месте (в подпрограмме, модуле) |
|  | подготовка | модификация команды или группы команд с целью воздействия на некоторую последующую функцию |
|  | решение | решение или функция переключательного типа, имеющая один вход и ряд альтернативных выходов, один и только один из которых может быть активизирован после вычисления условий, определённых внутри этого символа; соответствующие результаты вычислений могут быть записаны по соседству с линиями, отображающими эти пути |
|  | граница цикла | начало и конец цикла; обе части имеют один и тот же идентификатор; условия инициализации, приращения, завершения, и т.д. помещаются внутри символа в начале или в конце в зависимости от расположения операции, проверяющей условия |
|  | соединитель | выход в часть схемы и вход из другой части схемы; используется для обрыва линии и продолжении её в другом месте; соответствующие символы-соединители должны содержать одно и то же уникальное обозначение |
|  | терминатор | вход из внешней среды и выход во внешнюю среду(начало и конец программы) |
| **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** | линия | поток данных или управления; для направлений справа -налево и снизу-вверх добавляются стрелки |
| **- - - - - - - - - - - - - -** | пунктирная линия | используется для обведения аннотируемого участка |
| **- - - - - - - - - [** | комментарий | пояснительные записи и примечания |
| **\_\_\_\_\_\_  . . .**  **\_\_\_\_\_** | пропуск | пропуск символа или группы символов, в которых не определены ни тип ни число символов; используется только в символах линии или между ними; служит для отображения общих решений с неизвестным числом повторений. |

     Приведем основные свойства граф-схем.

1.        Графическое представление.

2.        Поддержка описания управляющей части алгоритма.

3.        Возможность реализации синтаксического контроля.

4.        Возможность проверки управляющей части алгоритма.

5.        Отсутствие возможности верификации информационной части.

     Необходимо отметить, что по граф-схеме алгоритма можно произвести оценку таких характеристик алгоритма, как временная сложность и сложность описания, однако, возможность оценки емкостной сложности отсутствует. На рис. 1 представлен пример граф-схемы алгоритма.

## Основы алгоритмизации и программирования

**Программа** — это детальное и законченное описание алгоритма средствами языка программирования. Исполнителем программы является компьютер. Для выполнения компьютером программа должна быть представлена в машинном коде — последовательности чисел, понимаемых процессором. Написать программу в машинных кодах вручную достаточно сложно. Поэтому сегодня практически все программы создаются с помощью языков программирования, которые по своим синтаксису и семантике приближены к естественному человеческому языку. Это снижает трудоемкость программирования. Однако, текст программы, записанный с помощью языка программирования, должен быть преобразован в машинный код. Эта операция выполняется автоматически с помощью специальной служебной программы, называемой **транслятором**.

Трансляторы делятся на два типа: **интерпретаторы** и **компиляторы**.

Интерпретатор переводит в машинный код и выполняет очередной оператор (команду) программы. Если команда повторяется, то интерпретатор рассматривает ее как встреченную впервые.

Компилятор переводит в машинный код исходный текст программы целиком. Поэтому достоинство компиляторов — быстродействие и автономность получаемых программ. Достоинство интерпретаторов — их компактность, возможность остановить в любой момент выполнение программы, выполнить различные преобразования данных и продолжить работу программы.

Примерами служебных программ — интерпретаторов являются GW Basic, Лого, школьный алгоритмический язык, многие языки программирования баз данных. Компиляторами являются Turbo Pascal, С++, Delphi.

### Средства создания программ

В общем случае для создания программ нужно иметь следующие компоненты

* текстовый редактор — для набора исходного текста программы;
* компилятор — для перевода текста программы в машинный код;
* редактор связей — для сборки нескольких откомпилированных модулей в одну программу;
* библиотеки функций — для подключения стандартных функций к программе.

Современные системы программирования включают в себя все указанные компоненты и называются **интегрированными системами**.

Исходный текст программы можно получить без записи его вручную в текстовом редакторе. Существуют системы визуального программирования — **RAD**-среды (Rapid Application Development), которые, не исключая возможности записи программы вручную, позволяют создавать текст программы автоматически, путем манипуляций со стандартными элементами управления, включенными в RAD-среду. Поэтому для RAD-среды понятие «программирование» часто заменяют понятием «проектирование».

По способу разработки программ можно выделить два подхода:

* процедурное программирование — это программирование, при котором выполнение команд программы определяется их последовательностью, командами перехода, цикла или обращениями к процедурам;
* объектно-ориентированное программирование – программирование, при котором формируются программные объекты, имеющие набор свойств, обладающие набором методов и способные реагировать на события, возникающие как во внешней среде, так и в самом объекте (нажатие мыши, срабатывание таймера, превышение числовой границы и т.д.). Таким образом, выполнение той или иной части программы зависит от событий в программной системе.

Объектно-ориентированное программирование (ООП) не исключает, а охватывает технологию процедурного программирования.

### Основные системы программирования

Из универсальных языков программирования наиболее популярны следующие: Basic; Pascal; C++; Java.

Для языка Basic существует много версий, реализованных и как интерпретаторы и как компиляторы. В России Basic традиционно используется в курсе информатики средней школы. Среда визуального программирования Microsoft Visual Basic используется как программная поддержка приложений MS Office.

Язык Pascal является компилируемым и широко используется как среда для обучения программированию в ВУЗах. RAD-средой, наследующей его основные свойства, является среда Borland Delphi.

Для языка C++ RAD-средой является Borland C++ Builder. Этот компилируемый язык часто используется для разработки программных приложений, в которых необходимо обеспечить быстродействие и экономичность программы.

Язык Java — интерпретируемый язык — позволяет создавать платформенно-независимые программные модули, способные работать в компьютерных сетях с различными операционными системами. RAD-средой для него является Symantec Cafe.

### Основные этапы развития языков программирования

Языки программирования развивались одновременно с развитием ЭВМ. С начала 50-х годов это были низкоуровневые языки (машинные и ассемблеры). В 1956 году появился язык Фортран, а в 1960 — Алгол-60. Это языки компилирующего типа, существенно уменьшившие трудоемкость программирования. Языки ориентированы на выполнение математических вычислений. В дальнейшем возникло большое количество различных языков, претендовавших на универсальность (PL/1) или для решения конкретных задач (COBOL — для деловых задач, ЛОГО — для обучения, Пролог — для разработки систем искусственного интеллекта). С середины 60-х до начала 80-х разработаны и получили распространение языки Pascal, Basic, Си, Ада и другие.

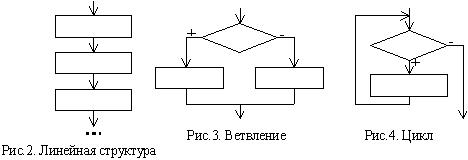
Принципиально новым этапом в развитии языков программирования стало появление методологии непроцедурного (ООП) программирования (см. выше). Основные достоинства ООП — быстрота разработки интерфейса программного приложения, возможность наследования свойств программных объектов.

Можно подойти и к другому понятия алгоритма: Алгоритм — это предписание некоторому исполнителю выполнить конечную последовательность действий, приводящую к некоторому результату.

В программировании алгоритм является фундаментом программы, а основным **исполнителем** — компьютер. На стадии тестирования алгоритма исполнителем может быть сам программист.

Алгоритм может быть записан с помощью блок-схемы, текстовым предписанием, с помощью рисунков, таблично или на специальном алгоритмическом языке. Наиболее популярны блок-схемы и предписания. Преимущество блок-схем — в наглядности алгоритма.

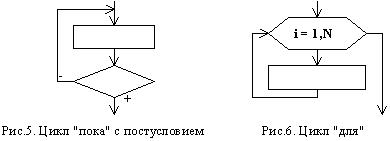
В теории алгоритмов доказано, что любой, сколь угодно сложный алгоритм может быть составлен из трех основных алгоритмических структур: линейной, ветвления и цикла, показанных, соответственно на рис. 2, 3, 4.



Линейная структура предполагает последовательное выполнение действий, без их повторения или пропуска некоторых действий. Обычно программисты стремятся к тому, чтобы алгоритм имел линейную структуру.

Структура "ветвление" предполагает выполнение одной из двух групп действий в зависимости от выполнения условия в блоке ветвления. На рис. 3 знаком "+" показано выполнение условия, а знаком "-" — его невыполнение. Часто используется неполная команда ветвления, когда один из блоков действия отсутствует.

Структура "цикл" имеет несколько разновидностей. На рис. 4 показан цикл типа "пока" с предусловием. Действия внутри этого цикла повторяются пока выполняется условие в блоке ветвления, причем сначала проверяется условие, а затем выполняется действие. Достаточно часто используются другие типы цикла, показанные на рис. 5 и 6.



В цикле с постусловием проверка условия выхода из цикла выполняется после очередного действия. Цикл "для" является модификацией цикла "пока" для ситуации, когда заранее известно количество повторений некоторых действий. Запись в блоке заголовка цикла на рис.6 показывает пример описания заголовка цикла, в котором действия повторяются столько раз, сколько целых значений приобретает параметр цикла i от своего начального значения 1 до конечного N с шагом 1. Обычно шаг не указывается, если он равен 1.

В языках программирования имеются команды, реализующие показанные выше структуры.

При разработке блок-схемы допускается делать любые записи внутри блоков, однако эти записи должны содержать достаточно информации для выполнения очередных действий.

**Пример 1**

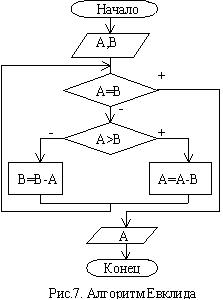
Разработать блок-схему алгоритма Евклида, определяющего наибольший общий делитель (НОД) двух натуральных чисел A и B.

В основе алгоритма Евклида лежит правило:

НОД(A,B)= НОД(min(A,B), |A-B|),

где НОД(A,B) — наибольший общий делитель двух натуральных чисел A и B.

Основной идеей решения задачи является многократное применение указанного выше правила, после которого большее из чисел очередной пары уменьшается. Решение получено, когда числа оказываются равны друг другу. Поскольку количество повторений заранее неизвестно, в алгоритме следует применить цикл "пока" с предусловием (рис. 7).



### Методика разработки алгоритмов

Разработке алгоритма предшествуют такие этапы, как формализация и моделирование задачи. **Формализация** предполагает замену словесной формулировки решаемой задачи краткими символьными обозначениями, близкими к обозначениям в языках программирования или к математическим. **Моделирование** задачи является важнейшим этапом, целью которого является поиск общей концепции решения. Обычно моделирование выполняется путем выдвижения гипотез решения задачи и их проверке любым рациональным способом (прикидочные расчеты, физическое моделирование и т.д.). Результатом каждой проверки является либо принятие гипотезы, либо отказ от нее и разработка новой.

При разработке алгоритма используют следующие основные принципы.

1. **Принцип поэтапной детализации алгоритма** (другое название — "проектирование сверху-вниз"). Этот принцип предполагает первоначальную разработку алгоритма в виде укрупненных блоков (разбиение задачи на подзадачи) и их постепенную детализацию.
2. **Принцип "от главного к второстепенному"**, предполагающий составление алгоритма, начиная с главной конструкции. При этом, часто, приходится "достраивать" алгоритм в обратную сторону, например, от середины к началу.
3. **Принцип структурирования**, т.е. использования только типовых алгоритмических структур при построении алгоритма. Нетиповой структурой считается, например, циклическая конструкция, содержащая в теле цикла дополнительные выходы из цикла. В программировании нетиповые структуры появляются в результате злоупотребления командой безусловного перехода (GoTo). При этом программа хуже читается и труднее отлаживается.

Говоря о блок-схемах, как о средстве записи алгоритма, можно дать еще один совет по их разработке. Рекомендуется после внесения исправлений в блок-схему аккуратно перерисовывать ее с учетом этих исправлений. Аккуратность записи есть аккуратность мысли программиста. Аккуратно записанный и детализованный алгоритм упрощает его программирование и отладку.

Рассмотрим задачу с достаточно сложным алгоритмом решения для того, чтобы, во-первых, продемонстрировать этапы 1 - 3 рассмотренной методики и, во-вторых - принцип поэтапной детализации алгоритма.

Постановка задачи. Существует способ обойти шахматным конем доску, побывав на каждом поле по одному разу. Построить алгоритм обхода доски.

Идея решения задачи. Очередной ход следует делать на то поле, с которого на другие поля меньше всего ходов.

Формализация задачи. Назовем термином "потенциал поля" количество допустимых ходов коня. Введем следующие обозначения:

* С — матрица 8\*8, содержащая потенциалы полей (фрагмент C показан на рис. 13);
* R — матрица 8\*8, содержащая решение задачи в виде номеров ходов коня;
* Sx, Sy — массивы из 8 элементов, содержащие смещения коня относительно текущей координаты, необходимые для реализации правила буквы "Г":

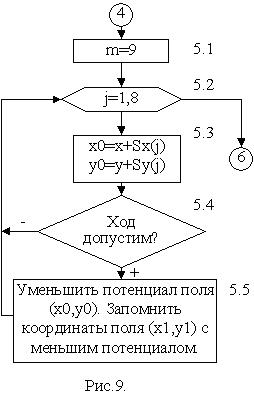
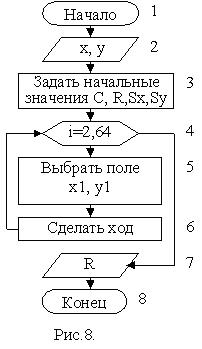
Sx = ( 1, 2, 2, 1,-1,-2,-2,-1);  
Sy = (-2,-1, 1, 2, 2, 1,-1,-2).

* x, y — текушие координаты коня;
* x1,y1 — координаты поля с минимальным потенциалом для текущих (x, y);
* m — значение минимального потенциала допустимого поля.

Будем учитывать пройденные поля путем задания соответствующим элементам матрицы C значения 9, т.е. значения вне множества допустимых потенциалов.

**Разработка алгоритма решения задачи**

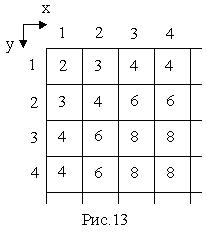
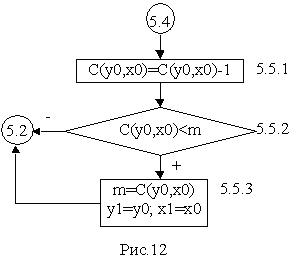
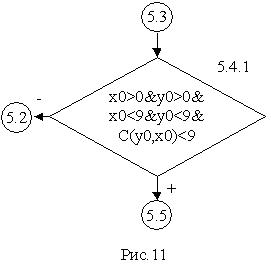
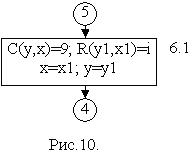
На рис. 8 показан укрупненный алгоритм решения поставленной задачи. На рис. 9 - 12 показаны основные шаги поэтапной детализации основного алгоритма.



Следует обратить внимание на нумерацию блоков в детализирующих блок-схемах. Число до первой точки является номером детализируемого блока в основной схеме. Число после первой точки является номером блока в схеме детализации первого уровня и т.д.

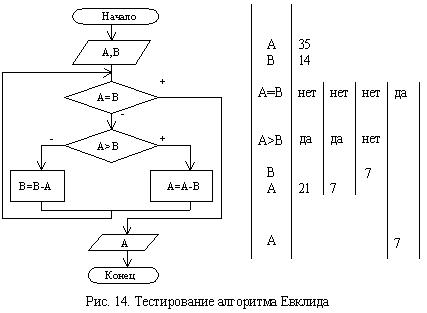
Входы в детализирующие блок-схемы и выходы из них показаны окружностями с номерами блоков — источников информации и получателей результатов.

Значком & на рис. 11 обозначена логическая операция И.



**Пример 3**

Поскольку тестирование вручную алгоритма решения задачи о шахматном коне было бы достаточно громоздким, рассмотрим технологию тестирования  на примере алгоритма Евклида (рис. 14).



Для тестирования вручную следует оставить достаточно свободного места справа от блок-схемы. Контрольный пример не должен быть слишком сложным - это затрудняет тестирование, но и не быть тривиальным - это может привести к случайному совпадению с правильным решением. В первом столбце таблицы справа от блок-схемы, записываются переменные или условия, значения которых могут изменяться. Начиная со второго столбца сверху-вниз записываются результаты выполнения алгоритма. Начало нового цикла соответствует добавлению нового столбца таблицы. Исполнитель (разработчик алгоритма) должен выполнять команды формально, строго придерживаясь предписаний в блок-схеме.

 Задача 2

Вычислить сумму



Решение

Моделирование задачи 2 показывает, что для вычисления суммы можно использовать цикл For-Next, поскольку заранее известно количество слагаемых. Кроме того, не следует вычислять заново факториал для очередного слагаемого: гораздо проще вычислить факториал, опираясь на значение факториала для предыдущего слагаемого. Аналогично организуется чередование знаков слагаемых: введем целую переменную Z=1 и будем в цикле выполнять команду Z=-Z.

Для решения задачи введем следующие величины:

* S — искомая сумма;
* F — значение факториала;
* C — значение числителя;
* Z — знак слагаемого (+1 или –1).

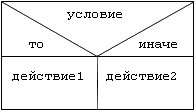
Таким образом, в задаче 2 дано: N, X; надо получить S.

Контрольный пример: при N=4 и X=3 получим S = -0.125.

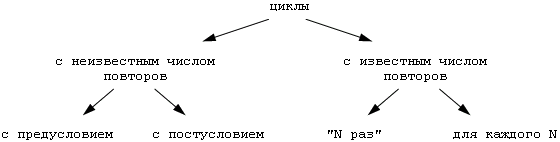
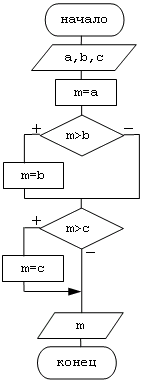
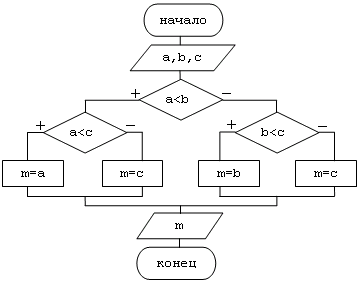
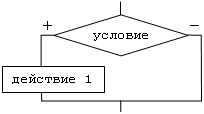
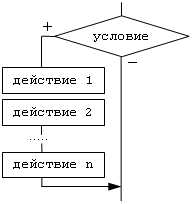
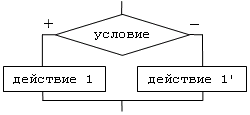
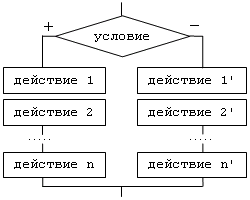
Алгоритм решения задачи показан на рис. 2.

**Лекция 3. Алгоритмы ветвления и циклов**

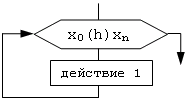
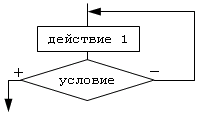
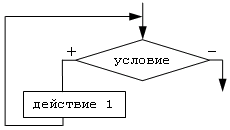
1. Структурограмма:



Ветвление - управляющая структура, организующая выполнение лишь одного из двух указанных действий в зависимости от справедливости некоторого условия.   
Условие - вопрос, имеющий два варианта ответа: да или нет.   
Запись ветвления выполняется в двух формах: полной и неполной.   
Полная форма:   
  
Неполная форма:   
  
Пример: найти наименьшее из трех чисел.   
1 вариант решения:   
  
2 вариант решения:   
  
  
  
 Алгоритмическая конструкция цикла.   
  
Цикл - управляющая структура, организующая многократное выполнение указанного действия.



Цикл "пока":   
  
Выполнение цикла "пока" начинается с проверки условия, поэтому такую разновидность циклов называют циклы с предусловием. Переход к выполнению действия осуществляется только в том случае, если условие выполняется, в противном случае происходит выход из цикла. Можно сказать что условие цикла "пока" - это условие входа в цикл. В частном случае может оказаться что действие не выполнялось ни разу. Условие цикла необходимо подобрать так, чтобы действия выполняемые в цикле привели к нарушению его истинности, иначе произойдет зацикливание.   
Зацикливание - бесконечное повторение выполняемых действий.   
Цикл "до":   
  
Исполнение цикла начинается с выполнения действия. Таким образом тело цикла будет реализовано хотя бы один раз. После этого происходит проверка условия. Поэтому цикл "до" называют циклом с постусловием. Если условие не выполняется, то происходит возврат к выполнению действий. Если условие истинно, то осуществляется выход из цикла. Таким образом условие цикла "до" - это условие выхода. Для предотвращения зацикливания необходимо предусмотреть действия, приводящие к истинности условия.   
Цикл с параметром, или цикл со счетчиком, или арифметический цикл - это цикл с заранее известным числом повторов.   
  
В блоке модификации указывается закон изменения переменной параметра.   
**Xo** - начальное значение параметра   
**h** - шаг   
**Xn** - последнее значение параметра   
Для создания циклов с параметром необходимо использовать правила:



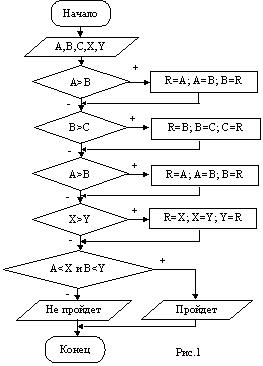
1. Параметр цикла, его начальное и конечное значения и шаг должны быть одного типа
2. Запрещено изменять в теле цикла значения начальное, текущее и конечное для параметра
3. Запрещено входить в цикл минуя блок модификации
4. Если начальное значение больше конечного, то шаг - число отрицательное
5. После выхода из цикла значение переменной параметра неопределенно и не может использоваться в дальнейших вычислениях
6. Из цикла можно выйти не закончив его, тогда переменная параметр сохраняет свое последнее значение

Задача 1

Дан кирпич прямоугольной формы со сторонами A, B, C и прямоугольное отверстие в стене со сторонами X и Y. Определить, пройдет ли кирпич в отверстие, если допускается располагать его грани только параллельно сторонам отверстия.

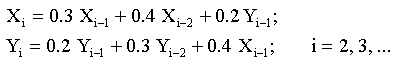
Решение

Основная идея решения заключается в упорядочивании сторон кирпича и отверстия по возрастанию. Если меньшая сторона кирпича меньше меньшей стороны отверстия, а вторая по размеру сторона кирпича меньше большей стороны отверстия, то кирпич пройдет, иначе — нет. Таким образом, требуется решить подзадачи упорядочивания сторон кирпича и сторон отверстия. Поскольку цель задачи — дать ответ "пройдет" или "не пройдет", будем выполнять упорядочивание, не вводя новых переменных. Для сторон отверстия задача решается просто: если X>Y, то обменять значения X и Y. Упорядочить стороны кирпича можно за три обмена: A и B, B и C и вновь A и B. Блок-схема алгоритма решения задачи показана на рис. 1.



Задача 4

Последовательность значений Xi, Yi вычисляется по формулам:



Вычислить при X0=X1=1 и Y0=Y1=2.



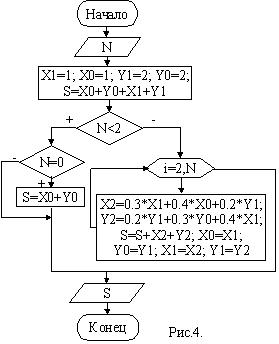
Решение

В данной задаче не следует применять массивы — лучше использовать простые переменные. Поэтому для численного эксперимента введем следующие обозначения: X2, X1, X0, Y2, Y1, Y0 – соответственно для Xi, Xi-1, Xi-2, Yi, Yi-1, Yi-2. Результаты численного эксперимента показаны в таблице.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **i** | **0** | **1** | **2** | **3** | **4** |
| **X0** |  |  | 1 | 1 | 1,1 |
| **X1** |  | 1 | 1 | 1,1 | 1,01 |
| **X2** | 1 | 1 | 1,1 | 1,01 | 1,007 |
| **Y0** |  |  | 2 | 2 | 1,4 |
| **Y1** |  | 2 | 2 | 1,4 | 1,32 |
| **Y2** | 2 | 2 | 1,4 | 1,32 | 0,987 |
| **Сумма** | 3 | 6 | 8,5 | 10,83 | 12,925 |

Таким образом, в задаче 4 в качестве контрольного примера можно использовать следующий: при N=4 должны получить сумму S=12.925.

Из таблицы видно, что после вычисления очередных значений X2 и Y2 и добавления их к сумме, необходимо изменить значения X1, Y1, X0, Y0 для подготовки очередного цикла. При этом следует начинать с X0 и Y0: X0=X1; Y0=Y1, а затем изменить X1 и Y1: X1=X2: Y1=Y2. Следует также учесть, что для вычисления суммы может быть задано любое целое N>=0, но применение формул начинается только c N=2. Блок-схема алгоритма решения показана на рис. 4.



**Лекция 4 Программирование на алгоритмическом языке**

История создания языка.   
  
Первая версия языка Паскаль была разработана в 1968 году. Ее разработчиком является швейцарский ученый Никлаус Вирт. Свое название язык получил в честь создателя первой механической вычислительной машины француза Блеза Паскаля. На основе языка Паскаль в 1985 г. фирма Borland выпустила версию Turbo Pascal версии 3.0. С этого времени язык Паскаль используется во всем мире в учебных заведениях в качестве первого изучаемого языка программирования.   
В пакете Turbo Pascal 4.0 были устранены ошибки и ограничения компилятора предыдущей версии. Наиболее важным нововведением была unit-концепция, позаимствованная из языка МОДУЛА-2. Это позволило разрабатывать крупные программные продукты. В версии 5.0 появился интегрированный отладчик. Был реализован аппарат перекрытий overlays. В этой версии были исправлены и улучшены библиотеки графических процедур, которым была обеспечена совместимость с графическими адаптерами класса VGA. Появились новые возможности справочной системы Help.   
В версии 6.0 была реализована концепция объектно-ориентированного программирования с полным набором прикладных задач для пользователя. В оболочку был встроен интегрированный текстовый редактор. В этой версии впервые использовалась мышь для управления работой.   
В 1992 г. появилась последняя на сегодняшний день версия языка Turbo Pascal - 7.0. В ней сохранились все достоинства предыдущих версий:

* многооконный режим работы
* возможность использования мыши
* возможность использования Ассемблера
* возможность создавать объектно-ориентированные программы

К улучшениям этой версии относятся:

* возможность выделять определенным цветом различные элементы исходного текста
* расширен язык программирования. Появился типизированный адресный оператор, открытые массивы и строки
* улучшен компилятор
* изменен интерфейс пользователя

Алфавит и программа на Паскале(Турбо Паскаль -ТП).   
  
Алфавит языка состоит из нескольких разделов:

1. Латинские буквы: A a B b…
2. Цифры: 0 1 2..9
3. Знаки математических операций: + - \* /
4. Знаки математических отношений: < > =
5. Знаки препинания: . , : ;
6. Специальные знаки: { } [ ] ( ) $ ^

Программа записанная на языке TP может содержать следующие разделы:

1. Заголовок
2. Раздел меток
3. Раздел констант
4. Раздел типов
5. Раздел переменных
6. Раздел процедур и функций
7. Раздел операторов

Все программы обязательно имеют раздел заголовок и раздел операторов. Остальные составляющие могут отсутствовать. При отсутствии некоторых частей программы общий порядок их следования сохраняется.   
Разделы между собой разделяются знаком ";"   
Раздел операторов заключается в операторные скобки. Это зарезервированные слова begin, end. Раздел операторов заканчивается точкой. Запись внутри операторных скобок ведется с отступом в три знака.   
Раздел "заголовок" начинается с зарезервированного слова, за которым указывается имя программы. В качестве имени может использоваться любой набор символов алфавита с несколькими исключениями:

1. Нельзя использовать зарезервированные слова
2. Нельзя начинать имя с цифры
3. При использовании имени не используется пробел

Простые типы данных.   
  
Любые данные ТП характеризуются своими типами. Тип определяет:

1. Формат представления данных в памяти компьютера
2. Множество допустимых значений, принимаемое переменной или константой, принадлежащей к выбранному типу
3. Множество допустимых операций и функций применимых к этому типу

Тип переменной определяется при ее декларации. Одна из базовых концепций Паскаля заключается в жесткой проверке соответствия типов в операциях присваивания.   
Типы данных в языке ТП делятся на 5 основных классов:

1. Простые типы
2. Структурированные типы
3. Ссылочные типы
4. Процедурные типы
5. Объектные типы

К простым типам относятся: целочисленные типы, логический тип, символьный тип, перечисляемый тип, интервальный тип, вещественные типы.   
Среди этих видов выделяют подмножества типов, отличных от вещественного, называемых порядковым типом.   
Порядковые типы обладают четырьмя характеристиками:

1. Все возможные значения данного порядкового типа представляют собой упорядоченное множество и каждое возможное значение связано с порядковым номером, который является целым числом
2. Значения любого порядкового типа, за исключением целочисленного начинается с порядкового номера ноль (следующий порядковый номер 1, 2, 3…)
3. Порядковым номером значения целочисленного типа является само значение
4. В любом порядковом типе каждому значению кроме первого есть предыдущее и каждому значению кроме последнего есть последующее

К данным любого порядкового типа можно применить любую из пяти операций:

1. Стандартная операция Ord возвращает порядковый номер указанного значения. Значение указывается в скобках
2. Стандартная операция Pred возвращает значение, предшествующее указанному, если эта функция применяется к первому значению данного типа, то выдается сообщение об ошибке
3. Стандартная операция Succ возвращает следующее значение за указанным, если операция применяется к последнему элементу типа, то выдается сообщение об ошибке
4. Стандартная операция Low возвращает наименьшее значение в диапазоне порядкового типа, указанного данного
5. Стандартная операция High возвращает наибольшее значение в диапазоне порядкового типа, указанного данного

В TP имеется 5 предопределенных, целочисленных типов. Каждый тип обозначает определенное подмножество целых чисел:

|  |  |  |
| --- | --- | --- |
| Тип | Диапазон | Формат |
| Короткое целое shortint | -128..127 | 8 бит со знаком |
| Целое integer | -32768..32767 | 16 бит со знаком |
| Длинное целое longint | -2147483648..2147483647 | 32 бита со знаком |
| Длиной в байт byte | 0..255 | 8 бит без знака |
| Длиной в слово word | 0..65535 | 16 бит без знака |

Верхнее граничное значение и нижнее граничное значение целочисленных типов задаются как константы и имеют соответствующее имя.   
В тексте программы данные целочисленных типов записываются в десятичном или шестнадцатеричном формате и не должны содержать десятичные точки.   
Пример:   
  1 - целый тип   
  1.0 - не целый тип   
  100 - десятичный формат (100)   
  #100 - шестнадцатеричный формат (256)   
Над целочисленными данными возможно выполнение операций сложения, вычитания и умножения, а также операций сравнения. Арифметические действия над операндами целочисленного типа предполагают восьмибитовую, 16-битовую или 32-битовую точность вычислений, в соответствии со следующими правилами:

1. Тип целой константы представляет собой встроенный целочисленный тип с наименьшим диапазоном, включающий значение данной константы
2. В случае бинарной операции оба операнда преобразуются к общему типу. Общим типом будет целочисленный тип с наименьшим диапазоном, включающим значение обоих операндов. Действие выполняется в соответствии с точностью общего типа и результатом будет общий тип
3. Выражение справа в операторе присваивания вычисляется независимо от размера или типа переменной слева
4. Любые операнды размера в байт преобразуются в операнды размера слово

К логическим типам относятся данные типов Boolean.   
Значением каждого данного логического типа могут являться 2 значения: TRUE (1) и FALSE (0).   
Для данных логического типа применимы только две операции сравнения: равно и не равно.   
Переменные типа Boolean занимают один байт.Предполагается, что тип Boolean имеет порядковые значения 0 и 1.

Символьный тип (char) представляет собой тип данных, предназначенный для хранения одного символа (буквы, знака или кода). В переменную этого типа может быть помещен любой из 256 символов расширенного кода ASCII.   
Переменная типа char занимает один байт памяти. Значения типа char задаются в апострофах. Кроме того можно задавать значения используя код из таблицы ASCII. Над данными символьного типа можно выполнять операции сравнения.   
Перечисляемый тип определяется как упорядоченный набор идентификаторов, заданный путем их перечисления. При этом список идентификаторов разделенных запятой указывается в круглых скобках. Задается перечисляемый тип в разделе type.   
Пример:   
  type A=(2, 4, 1, 7);   
         B=('c', 'L', '3', '|');   
Значения переменных перечисляемого типа не могут вводиться с клавиатуры и выводиться на экран.   
Интервальный тип данных определяется посредством задания подмножества значений одного из ранее определенных типов. Можно использовать все простые типы, за исключением вещественного. При задании диапазона указывается наименьшее и наибольшее значения, разделенные двумя точками. При этом оба значения обязательно одного типа.   
К вещественному типу относится подмножество вещественных чисел, представленных в формате с плавающей точкой и фиксированным числом цифр.   
В ТП имеется 5 видов вещественных типов:

|  |  |  |  |
| --- | --- | --- | --- |
| Тип | Диапазон | Точность | Формат |
| Real (вещественное) | 2.9\*10-39..1.7\*1038 | 11-12 знаков | 6 байт |
| Single (с одинарной точностью) | 1.5\*10-45..3.4\*1038 | 7-8 знаков | 4 байта |
| Double (с двойной точностью) | 5.0\*10-324..1.7\*10308 | 15-16 знаков | 8 байт |
| Extended (с повышенной точностью) | 3.4\*10-4932..1.1\*104932 | 19-20 знаков | 10 байт |
| Comp (сложное) | -9.2\*1018..9.2\*1018 | 19-20 знаков | 8 байт |

Действия над типами с одинарной, двойной, повышенной точностью и сложным типом могут выполняться только при наличии числового сопроцессора. Поэтому считается что постоянно доступным является только тип Real.   
  
  
 Константы, переменные и оператор присваивания.   
  
Константа - это идентификатор отмечающий значение, которое не может изменяться. Идентификатор константы не может быть включен в свое собственное описание. Константы должны объявляться в декларационной части программы до момента их использования в вычислениях. Эта декларационная часть начинается с зарезервированного слова const. При декларации указывается имя константы, символ равенства и значение этой константы. В ТП применяется 5 видов констант простых типов:

1. Целочисленные константы. В качестве значений может использоваться любое целочисленное данное в десятичном или шестнадцатеричном формате (year=2003)
2. Вещественные константы определяются числами, записанными в десятичном формате данных (time=0.2e+4, yyy=304.0)
3. Символьные константы могут быть определены только посредством символов таблицы ASCII. При этом сам символ заключается в апострофы (var1='A')
4. Строковые константы определяются произвольной последовательностью символов, заключенных в апострофы (stroke='IBM')
5. Типизированные константы (переменные с начальным значением) . Каждой типизированной константе ставится в соответствие имя, тип, начальное значение (year1:integer =1995)

Переменной называется элемент программы, который предназначен для хранения, коррекции и передачи данных внутри программы.   
Раздел описания переменных начинается с зарезервированного слова var.   
Для объявления переменной необходимо указать имя переменной и ее тип. Однотипные переменные могут перечисляться через запятую перед указанием их типа.   
Пример:   
  a: integer;   
  b: boolean;   
  c, b: real;   
  e: integer;   
Все переменные делятся на глобальные и локальные. Глобальными являются переменные, объявленные вне процедур и функций, а локальными - объявленные внутри процедур и функций.   
ТП накладывает ряд ограничений на использование переменных:

1. Среди глобальных переменных не может быть двух с одинаковыми идентификаторами;
2. Среди локальных переменных в пределах одной процедуры или функции не может быть двух с одинаковыми идентификаторами;
3. В тексте программы любой глобальный идентификатор может дублировать любой локальный идентификатор, т.к. даже при одинаковых именах они хранятся в разных участках памяти.

Оператор присваивания - это основной оператор любого языка программирования. Данный оператор позволяет поместить определенное значение в необходимую вам переменную.   
Оператор присваивания имеет вид:   
    идентификатор:= выражение;   
При составлении выражений могут быть использованы следующие математические функции:

|  |  |  |
| --- | --- | --- |
| Имя функции | Математическое значение | Тип результата |
| a mod b | Остаток деления a на b | Целое |
| a div b | Целая часть деления a на b | Целое |
| abs (a) | |a| | Совпадает с типом аргумента |
| sqr (a) | a2 | Совпадает с типом аргумента |
| sqrt (a) |  | Вещественное |
| sin (a) | sin a | Вещественное |
| cos (a) | cos a | Вещественное |
| arctan (a) | arctg a | Вещественное |
| ln (a) | ln a | Вещественное |
| exp (a) | ea | Вещественное |

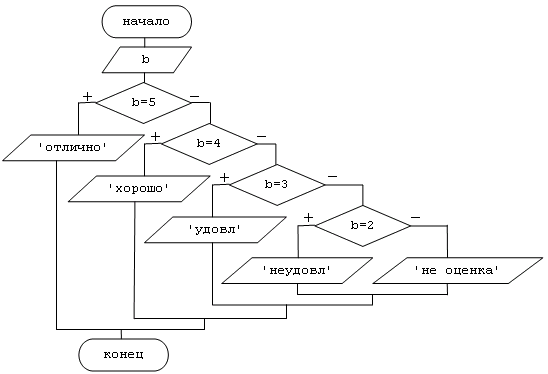
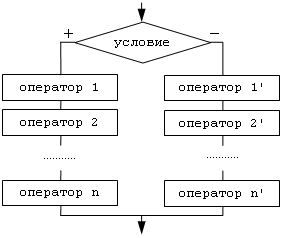
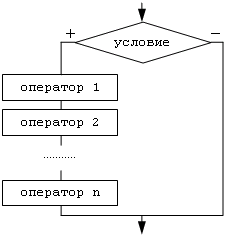
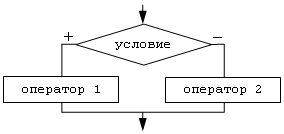
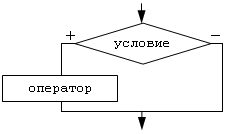
При составлении сложных выражений осуществляется приоритет выполнения операций:

1. not, @
2. \*, /, div, mod, and, shl, shr
3. +, -, or, xor
4. =, <>, <=, >=, >, <, in

Операторы ввода/вывода.   
  
ТП содержит четыре оператора ввода/вывода: read, readln, write, writeln.   
Оператор read осуществляет ввод данных с клавиатуры и размещение их в стандартном файле ввода input. Вводимые данные размещаются в качестве значений переменных, имена которых перечислены в круглых скобках за оператором read.   
    read (a, b, c);   
Вводятся данные тоже списком, в котором они разделяются пробелом. Ввод заканчивается нажатием **Enter**. Курсор, отмечающий позицию следующего ввода/вывода остается за последним введенным данным.   
Оператор readln выполняет аналогичные действия и переводит курсор на следующую строку.   
Оператор write осуществляет вывод на экран или печатающее устройство с одновременным размещением в стандартном файле вывода output. Оператор может выводить сообщение или значение переменной. Сообщения записываются в апострофах. Для вывода значения переменной указывается имя переменной. Сообщения и переменные можно чередовать в одном списке, разделяя их запятыми. Курсор остается за последним выведенным данным.   
Оператор writeln выполняет аналогичные действия и переводит курсор на следующую строку.   
Операторы write и writeln допускают т.н. форматированный вывод данных.   
    write (a:5:2);   
Первое из чисел указывает сколько экранных знаков отводится под вывод. Второе число указывает количество знаков после запятой в числе и может отсутствовать.

**Лекция 6. Управляющие операторы**

Безусловные конструкции.   
  
Оператор языка представляет собой неделимый элемент программы, который позволяет выполнять определенные алгоритмические действия. Все операторы можно условно разделить на две группы: простых операторов и структурированных операторов. К простым относятся те операторы, которые не содержат других операторов. К структурированным - те, которые состоят из других операторов.   
В ТП 7.0 существует всего один оператор безусловного перехода Goto и четыре безусловных функции: Break, Continue, Exit, Halt.   
Оператор безусловного перехода Goto представляет собой простой оператор, используя который можно изменять порядок выполнения операторов в программе. Общий вид оператора безусловного перехода:   
    goto <метка> , где <метка> - это идентификатор или целое число от 0 до 9999, объявленное в разделе меток label.   
Применение оператора безусловного перехода в ТП - программе является нежелательным, т.к. его присутствие нарушает структурную целостность и наглядность. Такую программу трудно читать, отлаживать и модифицировать.   
Функция Break позволяет досрочно закончить цикл.   
Функция Continue - позволяет начать новую итерацию цикла, даже если предыдущая не была завершена.   
Функция Exit - позволяет завершить работу текущего программного блока.   
Функция Halt (n), где n - некоторое целое число - позволяет завершить работу программы с кодом завершения n.   
  
  
3.2. Условные конструкции.   
  
1) неполная форма с одним оператором   
  
2) полная форма с одним оператором   
  
3) неполная форма с несколькими операторами   
  
4) полная форма с несколькими операторами   
  
  
1) IF условие THEN оператор;   
2) IF условие THEN оператор1 ELSE оператор2;   
3) IF условие THEN BEGIN   
                оператор1;   
                оператор2;   
                …   
                операторN;   
            END;   
4) IF условие THEN BEGIN   
                оператор1;   
                оператор2;   
                …   
                операторN;   
            END ELSE   
            BEGIN   
                оператор1;   
                оператор2;   
                …   
                операторN;   
            END;   
  
Пример: ввести оценку студента в баллах и сообщить ее название.   
  
  
Begin   
  Read(b)   
  If b=5 then Write('отлично') else   
    If b=4 then Write('хорошо') else   
      If b=3 then Write('удовл.') else   
        If b=2 then Write('неудовл.') else   
        Write('это не оценка');   
End.   
  
  
 Конструкция выбор.   
  
Ситуации, реализующие систему вложенных ветвлений могут быть разрешены с использованием конструкции выбор.   
Оператор выбора является структурированным и использует в своей записи операторы case, of, else, end и операторные скобки по необходимости.   
В самом общем виде оператор выбора можно записать так:   
Case порядковая переменная of   
    значение1: begin оператор1; оператор2; …; операторN; end;   
    значение2: begin оператор1; оператор2; …; операторN; end;   
    …   
    значениеM: begin оператор1; оператор2; …; операторN; end;   
    else begin оператор1; оператор2; …; операторN; end;   
end;   
  
Пример: ввести оценку студента в баллах и сообщить ее название.   
Begin   
  Read(b)   
  Case b of   
    5: Write('отлично');   
    4: Write('хорошо');   
    3: Write('удовл.');   
    2: Write('неудовл.');   
    else Write('это не оценка');   
  end;   
End.   
  
Порядковая переменная, значение которой при выполнении программы определяет ветвь в операторе выбора, подлежащую выполнению, может принадлежать любому целочисленному типу. В случае, когда для нескольких значений выполняемые действия одинаковы, их можно указать один раз, а сами значения перечислить через запятую.   
Пример: напечатать количество дней во введенном месяце:   
Begin   
  Read(m);   
  Case m of   
    янв, мар, май, июл, авг, окт, дек: Write('31');   
    апр, июн, сен, ноя: Write('30');   
    фев: Write('28');   
    else Write ('это не месяц');   
  end;   
End.   
  
  
3.4. Циклические конструкции.   
  
**1. Цикл с предусловием.**   
Для реализации циклов с предусловием используется составной оператор, включающий оператор while, do, операторные скобки.   
В общем виде цикл реализуется записью:   
    while <условие> do <действие>;   
Если тело цикла содержит более одного действия, то необходимо использовать операторные скобки:   
    while <условие> do   
    begin   
      <оператор 1>;   
      <оператор 2>;   
      ...   
      <оператор n>;   
    end;   
  
**2. Цикл с постусловием.**   
Для реализации цикла используется составной оператор, состоящий из операторов repeat и until.   
В общем виде цикл записывается так:   
    repeat   
      <действие>;   
    until <условие>;   
  
Пример: задано целое число. Вывести на печать все цифры введенного числа.   
1 способ:

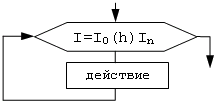


|  |  |
| --- | --- |
|  | var a,b:longint;      Begin        read(a);        repeat          b:=a mod 10;          writeln(b);          a:=a div 10;        until a=0;      End. |

2 способ:

|  |  |
| --- | --- |
|  | var a,b:longint;      Begin        read(a);        while a<>0 do          begin            b:=a mod 10;            write(b:3);            a:=a div 10;          end;      End. |

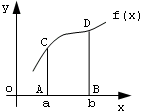
**3. Цикл с параметром.**   
  
Для реализации в языке Pascal используется составной оператор, состоящий из операторов for, to, downto, do и при необходимости из операторных скобок. Переменная параметр обязательно объявляется в декларационной части программы и может принадлежать одному из порядковых типов.   
Если при изменении переменной параметра необходимо использовать переход к следующему значению, то используется оператор to; если переход необходимо осуществить к предыдущему значению, то используется оператор downto. Тогда в общем виде цикл записывается так:   
    for I:=I0 to In do   
      begin   
        <оператор 1>;   
        <оператор 2>;   
        ...   
        <оператор n>;   
      end;



**Лекция 6. Процедуры и функции.**   
  
Процедуры и функции используются в тех случаях, когда:

1. Алгоритм или программа содержат одинаковые действия, различающиеся, возможно исходными данными;
2. Решаемая задача состоит из нескольких задач, меньших по объему и сложности;
3. Решением задачи занимается коллектив программистов.

Функции пользователя.   
  
Работа с функцией в Паскаль-программе состоит из двух частей: объявление функции и обращение к функции. Объявление функции производится в специальном разделе декларационной части Паскаль-программы непосредственно перед разделом операторов. Начинается объявление с заголовка функции. В общем виде заголовок имеет следующие разделы:   
   function <имя функции> (<список параметров>): <тип возвращаемого результата>, где function - зарезервированное слово.   
В качестве имени функции может использоваться любой допустимый идентификатор.   
Список параметров содержит перечисление всех формальных аргументов с указанием их типа. Однотипные аргумент можно перечислить списком.   
Тип возвращаемого результата - один из стандартных типов языка Паскаль.   
Пример:   
   function factorial (n:integer):real;   
   function mm (a, b:real; c:byte; d:char):integer;   
При обращении к функции из раздела операторов Паскаль-программы указывается имя функции и список фактических параметров.   
Обращение к функциям возможно из оператора вывода.   
В качестве фактических параметров могут использоваться как значения, так и ссылки на другие переменные. В любом случае количество фактических и формальных параметров одинаково и передача значений происходит в порядке записи.   
Пример:   
   F:=factorial(5);   
   T:=factorial(2)-7;   
   M:=factorial(n);   
   P:=mm(d,c,b,a);   
  
Задача 1: вычислить факториалы первых 10 натуральных чисел:   
var i: byte;   
   f: real;   
function factorial (n: byte): real;   
var i: byte;   
   f: real;   
begin   
   f:=1;   
   for i:=1 to n do   
      f:=f \* i;   
   factorial:=f;   
end;   
begin   
   for i:=1 to 10 do   
      begin   
         f:=factorial(i);   
         writeln(f);   
      end;   
end.   
  
Задача 2: вычислить 10 натуральных степеней для каждого из первых 10 натуральных чисел:   
var i, j : byte;   
   p: real;   
function stepen (b: integer; n: byte): real;   
var i: byte;   
   a: real;   
begin   
   a:=1;   
   for i:=1 to n do   
      a:=a \* b;   
   stepen:=a;   
end;   
begin   
   for i:=1 to 10 do   
      for j:=1 to 10 do   
         begin   
            p:=stepen (i, j);   
            write (p);   
         end;   
end.   
  
  
Процедуры пользователя.   
  
Функции являются частным случаем, т.е. подвидом процедур. Следовательно, все свойства функций справедливы для процедур.   
Объявляются процедуры в описательной части программы, в одном разделе с функциями. Порядок объявления независимых друг от друга процедур и функций не важен, если процедура использует в себе обращение к функции или другой процедуре, то последнии обязательно объявляются раньше.   
Объявление процедур начинается с заголовка:   
   procedure <имя процедуры> (<список параметров>);, где procedure - зарезервированное слово.   
Список параметров процедур содержит формальные параметры двух видов. Те, значения которых не возвращаются в программу, и параметры с возвращаемыми значениями.   
Последние в списке отмечаются зарезервированным словом var.   
Пример:   
   procedure xxx(a: byte; var b: byte; c,d: real; var j: char);   
Для обращения к процедуре в тексте программы указывается имя процедуры и, в скобках, список фактических параметров. В этом списке параметрам с возвращаемыми значениями обязательно соответствуют переменные.   
Пример:   
   xxx (5, b, c, 8, i);   
  
Задача 1: вычислить факториалы первых 10 натуральных чисел:   
var i: byte;   
   f: real;   
procedure factorial (n: byte; var f: real);   
var i: byte;   
begin   
   f:=1;   
   for i:=1 to n do   
      f:=f \* i;   
end;   
begin   
   for i:=1 to 10 do   
      begin   
         factorial(i, f);   
         writeln(f);   
      end;   
end.   
  
Задача 2: вычислить 10 натуральных степеней для каждого из первых 10 натуральных чисел:   
var i, j : byte;   
   p: real;   
procedure stepen (b: integer; n: byte; var f: real);   
var i: byte;   
begin   
   f:=1;   
   for i:=1 to n do   
      f:=f \* b;   
      stepen:=a;   
end;   
begin   
   for i:=1 to 10 do   
      for j:=1 to 10 do   
         begin   
            stepen (i, j, p);   
            write (p);   
         end;   
end.   
  
  
 Использование функций в приближенных вычислениях.   
  
Числовым рядом называется выражение вида U1+U2+…+Un+…, где U1, U2,…, Un - элементы ряда. Этот ряд можно записать в виде   
Сумма Sn=U1+U2+…+Un называется n-ой частной суммой ряда.   
Сумма U1(x)+U2(x)+…+Un(x)+…, в которой каждый элемент Un(x) является функцией аргумента x, называется функциональной.   
Степенным называется функциональный ряд вида a0+a1x+a2x2+…+anxn+…, где a0, a1, a2,…,an,… - постоянные числа, называемые коэффициентами ряда.   
Степенной ряд называется рядом Тейлора.   
  
  
5.5. Использование процедур в приближенных вычислениях.   
  
Пусть требуется вычислить   
  
Значение равно площади криволинейной трапеции ACDB, сторона CD которой является частью графика функции f (x).   
  
1 способ: преобразуем криволинейную трапецию в обычную, заменив сторону CD отрезком.



|  |  |
| --- | --- |
|  |  |

2 способ: заменим сторону CD криволинейной трапеции на отрезок, параллельный стороне AB и проходящий через одну из вершин C или D.

|  |  |
| --- | --- |
|  |  |

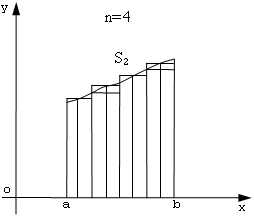
3 способ: восстановим в середине отрезка AB перпендикуляр до пересечения с графиком функции f(x). Проведем через найденную точку отрезок, параллельный AB.

|  |  |
| --- | --- |
|  |  |

Для нахождения приближенного значения определенного интеграла разобъем отрезок AB на две равные части (величину 2 обозначим через n). В середине каждого отрезка восстановим перпендикуляры и построим соответствующие прямоугольники. Вычислим площадь S1.

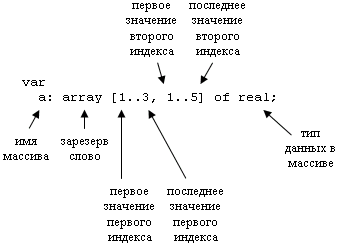
|  |  |
| --- | --- |
|  |  |

Т.к. выполнив вычисления один раз не возможно оценить их точность, то продолжим разбиение отрезка AB (n=4).   
  
После проведенных вычислений можно оценить полученную точность, которая равна |S1-S2|. Если точность оказывается достаточной, то последнее вычисленное значение S2 будет считаться значением определенного интеграла. Если точность не достаточна, то последнее разбиение считается начальным, т.е. S1=S2 и количество разбиений увеличивается вдвое, т.е. n=2\*n.   
  
  
5.6. Использование библиотек стандартных процедур в программах. Модуль Crt. Текстовые режимы использования экрана.   
  
Язык программирования Паскаль содержит ряд предопределенных процедур, разделенных по темам на несколько библиотек (модулей). Файлы с библиотеками процедур имеют расширение **tpu** и хранятся в папке **units** основной директории **tp**.   
Подключение библиотек к программе производится сразу за заголовком программы. Для этого используется служебное слово uses. За ним перечисляются через запятую имена файлов библиотек.   
Модуль Crt содержит процедуры и функции, испльзующиеся для работы на текстовом экране. Примером процедуры этого модуля может служить очистка экрана clrscr. Процедуры этого модуля всегда работают в активном окне. Окном считается прямоугольная область, определяемая координатами верхнего левого и нижнего правого угла. По умолчанию активным окном является экран. Размеры экрана, по умолчанию, - 80x25 знакомест.   
Заданный по умолчанию режим экрана можно изменить, используюя процедуру textmode (<константа режима>). Константа 0 задает черно/белый режим с размером экрана 40x25.   
1 - цветной режим 40x25   
2 - черно/белый режим 80x25   
3 - цветной режим 80x25   
7 - черно/белый режим и монохромный дисплей   
256 - загружаемый шрифт - 43 строки в EGA и 50 строк в VGA.   
  
Процедуры этого модуля могут обеспечивать работу со звуком. Частоту звука определяет процедура sound (<число герц>). Длительность звука регулируется процедурой delay (<длительность звука в мс.>) и процедурой nosound - отключение звука.   
Короткий звуковой сигнал можно обеспечить, используя символ #7 в списке вывода оператора write. В списке вывода также можно использовать еще 4 специальных символа:   
#8 - смещение курсора влево на одну позицию   
#10 - сдвиг курсора на одну строку вниз   
#13 - перемещение курсора на левую границу окна   
Комбинация #13#10 соответствует нажатию клавиши Enter.   
  
Для активирования нового окна следует определить его границы. Для этого используется процедура window (x1, y1, x2, y2). Координаты x1, y1, x2, y2 - абсолютные, все остальные координаты в окне - относительные, а точкой отсчета считается верхний левый угол окна. В пределах окна курсор можно переместить на овую позицию, используя процедуру gotoxy (x, y). Если x, y выходят за пределы окна, то процедура игнорируется. Для изменения цветовой гаммы окна используются следующие процедуры:   
textbackground (<…>) - изменяет цвет фона; в качестве аргументов выступают числа 0..7   
textcolor (<…>) - изменяет цвет текста; в качестве аргументов выступают числа 0..15.   
Для организации мерцающих цветов текста к константе цвета необходимо прибавить 128.



**Лекция 7-10. Структурированные типы данных**

Традиционно к структурированным типам данных в языке Pascal относят массивы, множества и записи. Иногда структурированным считают строковый тип. На самом деле строки занимают промежуточное положение между простыми и структурированными типами, но т.к. большинство их свойств реализуются как свойства сложных данных, то строки рассматриваются как структурированный тип.   
  
  
 Массивы.   
  
Объявление переменной массива происходит в разделе var. При этом используется зарезервированное слово array, указывается тип массива и его размерность перечислением индексов, а также объявляется тип данных в массиве.   
Пример 1:   
  
Пример 2:   
   var masiv: array ['a'..'z'] of integer;   
Для обращения к элементу массива в Pascal-программе указывается имя массива и в квадратных скобках индексы элемента.   
Пример 1:   
   a [2,4]   
а) ввести значение:   
   read (A[2,4]);   
б) изменить значение:   
   A[2,4]:=5;   
в) сообщить значение:   
   write (A[2,4]);   
  
Пример 2:   
   masiv ['b'];   
   masiv['b']:=47\*24;   
  
Задача ввода элементов массива.   
а) линейного:



|  |  |
| --- | --- |
|  | program vvod;     var A: array [1..10] of integer;        i:integer;     begin        for i:=1 to 10 do           read (A[i]);     end. |

б) двумерного:

|  |  |
| --- | --- |
|  | program vvod;     const n=5; m=7;     var A: array [1..n,1..m] of real;        i,j: integer;     begin        for i:=1 to n do           for j:=1 to m do              read (A[i,j]);     end. |

Задача 2: найти минимальный элемент в массиве:

|  |  |
| --- | --- |
|  | program min;     const n=10;     var i: byte;        a: array [1..n] of real;        min: real;     begin        for i:=1 to n do read (A[i]);        min:=A[1];        for i:=1 to n do           if min>A[i] then min:=A[i];        write (min);     end. |

Сортировка массивов.   
  
Сортировкой называется процесс расположения элементов массива в порядке убывания (возрастания) из значений.   
Пример :   
  
Алгоритм выполнения сортировки называется методом сортировки. К наиболее распространенным методам относятся:



1. Простым выбором
2. Простой перестановкой
3. Пузырьковый метод
4. На каждом шаге находится минимальный (максимальный) неотсортированной части. Он меняется с первым элементом в неотсортированной части, после чего отсортированная часть увеличивается на один элемент. На первом шаге весь массив считается неотсортированным. Сортировка заканчивается за (n-1) шаг.   
   Пример: 241795

|  |  |
| --- | --- |
| 1 шаг: 1 | 42795  2 шаг: 12 | 4795  3 шаг: 124 | 795  4 шаг: 1245 | 97  5 шаг: 124579 |  |

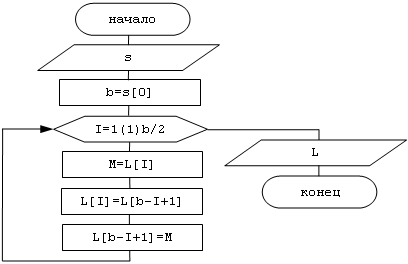
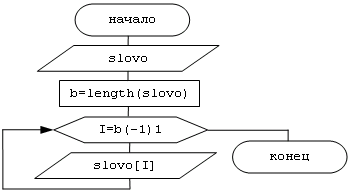
1. На каждом шаге массив делится на отсортированную и неотсортированную части. Первый элемент из неотсортированной части сравнивается с каждым элементом отсортированной части, начиная с последнего. Если найден элемент, больший сравниваемого, то они меняются местами. Шаг закончен когда просмотрены все отсортированные элементы. Сортировка закончена когда просмотрены все неосортированные элементы. На первом шаге отсортироованным считается первый элемент.   
   Пример: 2 | 41795

|  |  |
| --- | --- |
| 1 шаг: 24 | 1795  2 шаг: 21 | 4795               124 | 795  3 шаг: 1247 | 95  4 шаг: 12479 | 5               124759               124579 |  |

1. На каждом шаге сравниваются все соседние элементы. В случае необходимости они меняются местами. Сортировка считается законченной за nn действий или на шаге, когда не выполнено ни одной перестановки.   
   Пример: 241795

|  |  |
| --- | --- |
| 1 шаг: 214795   true  2 шаг: 124579   true  3 шаг: 124579   false |  |

Строковый тип данных.   
  
Переменная типа строка предназначена для обработки цепочек символов. Каждый символ является элементом типа char. Строки могут вводиться с помощью стандартных операторов read/readln и выводиться стандартными операторами write/writeln.   
Объявляются переменные типа строка в разделе var. При объявлении указываются идентификатор переменной, зарезервированное слово string и, в квадратных скобках, целое число - максимально возможная длина строки. Наибольшая длина строки составляет 256 символов. Если переменная имеет значение с максимальной длиной строки, то при объявлении переменной ограничиваются зарезервированным словом.   
Пример:   
var   
   identificator\_1: string;   
   identificator\_2: string[20];   
   identificator\_3: string[255];   
Значение строкового типа также как и значение типа char при записи внутри программы заключаются в апострофы.   
Пример:   
   identificator\_1:='это - компьютер';   
   identificator\_1[1]:='э';   
Простейшая операция которую Pascal позволяет выполнить со строками - это операция конкатенации, или сцепления, или объединения строк в операторе присваивания. Операция записывается с помощью знака "+".   
Пример:   
   identificator\_1:='это' + '-' + 'компьютер';   
  
Для обработки строковых данных используется ряд встроенных функций:   
1) Length (L) - определяет длину строки, являющуюся значением переменной L. Значение, возвращаемое этой функцией является целочисленным и отображает реальную длину строки, т.е. может не совпадать со значением длины строки, объявленным при декларации.   
Пример 1:   
var   
   L: string[15];   
   A: byte;   
Begin   
   L:='Урок';   
   A:=length(L);   
   Write(A);   
End.   
  
Пример 2:   
Begin   
   write(length('Урок'));   
End.   
2) Upcase (C) - преобразует любой символ в прописной. Переменная C может иметь значение типа char, либо являться одним элементом из строки. Русские символы обрабатываться этой функцией не могут.   
3) Copy (L, A, B) - позволяет копировать фрагмент строки являющейся значением переменной L, начиная с позиции A в количестве B, где A и B - целые числа, причем значение A не превышает длины строки L, а значение B не превышает (длина строки L - A). Если эти правила нарушены, то ошибки компиляции не произойдет, но возможно совершение логической ошибки в программе.   
4) Pos (L, M) - возвращает результат целочисленного типа, являющийся номером позиции, с которой строка L входит в строку M. Если строки L нет в строке M, то результат - 0.   
5) Insert (L, M, A) - вставляет строку L в строку M, начиная с позиции с номером A. Фактически, вставка производится перед указанной позицией.   
6) Delete (L, A, B) - удаляет из строки L B символов, начиная с позиции A.   
Если номера позиций в функциях Insert и Delete не соответствуют длине рассматриваемых строк, то произойдет ошибка компиляции.   
Пример 1: переставить буквы введенного слова в противоположном порядке. Например, ввели "урок", получили - "кору":   
  
Пример 2: изменить введенное значение строковой переменной на слово, записанное теми же символами, но в обратном порядке. Новую переменную не использовать:   
  
Пример 3: сравнение слов в Pascal.

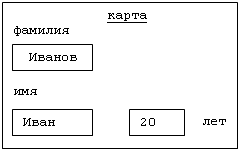


|  |  |
| --- | --- |
|  | Program slovo;  Var     word1, word2: string[60];  Begin     readln(word1);     readln(word2);     if word1 > word2 then writeln ('>')        else begin     if word1 = word2 then writeln ('=')        else writeln ('<');  End. |

Множества.   
  
Множеством называется упорядоченная совокупность данных одного типа, записанных без повторений и отсортированных по возрастанию. Максимальное множество состоит из 256 элементов. Для объявления множеств используется зарезервированное слово set, за которым указывается тип элемента множества или сами элементы.   
Пример:   
type A: 5..9;   
var   
   B: set of A;   
   C: set of char;   
   D: set of '1'..'5';   
Внутри программы элементы множества записываются в квадратных скобках. Для перечисления используется интервальный, перечисляемый типы или их комбинации.   
Пример: e:['A'..'Q', 'T', 'x'..'z']   
Элементы множеств нельзя вводить с клавиатуры и выводить стандартными операторами, т.к. элементы множества относятся к перечисляемому типу.   
Над множествами можно выполнять следующие операции:

1. Объединение (+). Результатом будет множество, состоящее из всех элементов первого и второго множеств без повтора:   
   Пример:   
   [1..3, 6, 9..11] + [2..4, 7, 10..12] = [1, 2, 3, 4, 6, 7, 9, 10, 11, 12] = [1..4, 6, 7, 9..12]
2. Пересечение (\*). Результатом будет множество, состоящее из тех элементов, которые присутствуют как в первом, так и во втором множествах.   
   Пример:   
   [1..3, 6, 9..11] \* [2..4, 7, 10..12] = [2, 3, 10, 11]
3. Разность двух множеств (-). Результатом будет множество, состоящее из тех элементов первого множества, которых нет во втором.   
   Пример:   
   [1..3, 6, 9..11] - [2..4, 7, 10..12] = [1, 6, 9]
4. Операция in - проверяет принадлежность элемента множеству. Результатом операции будет логическое значение (true или false).   
   Пример:   
   [2] in [1..4] (true)   
   [7] in [1..7] (false)
5. Сравнение. Равными называются множества, состоящие из одинаковых элементов. Большим будет множество, у которого больше элементов. Из двух множеств с равным количеством элементов большим будет то, первое несовпадающее значение которого больше.

Пример: из введенной последовательности символов, признаком конца которой является '0', сформировать множество заглавных и строчных латинских букв.   
var   
   c: char;   
   a, pl: set of 'A'..'Z';   
   b, sl: set of 'a'..'z';   
   i: char;   
Begin   
   pl:= [0];   
   sl:= [0];   
   repeat   
      read(c);   
      if [c] in a then pl:=pl+[c];   
      if [c] in b then sl:=sl+[c];   
   until [c]='0';   
   for i:='A' to 'Z' do   
      if [i] in pl then write(i:3);   
   for i:='a' to 'z' do   
      if [i] in sl then write(i:3);   
End.   
  
  
Записи.   
  
Пример:   
  
Для реализации объединения данных разного типа в языке Pascal существует специальная структура - запись. Объявление записи начинается с зарезервированного слова record, за которым перечисляются имена и типы всех составляющих записей ее полей. Заканчивается объявление скобкой end.   
Пример:   
type   
   karta = record   
      family: string[20];   
      name: string[15];   
      age: integer;   
   end;   
При обращении к записи в программе указывается имя записи и через точку имя поля.   
Пример:   
   karta.family:='Иванов';   
   karta.name:='Иван';   
   karta.age:=20;   
Для упрощения обращения к записи может быть использован оператор работы со структурой with.   
Пример:   
with karta do   
   begin   
      family:='Иванов';   
      name:='Иван';   
      age:=20;   
   end;   
Полями записи наряду с простыми типами могут быть и данные структурированных типов, например, массивы или записи.   
Пример 1:   
var z: record   
      pole1: string;   
      pole2: array [1..10] of byte;   
   end;   
Begin   
   for i:=1 to 10 do   
   read (z.pole2[i]);   
End.   
  
Пример 2: объявите запись, содержащую сведения о фамилии, дате рождения и адресе студента.   
var student: record   
      fam: string[15];   
      data: record   
            day: 1..31;   
            mes: 1..12;   
            year: integer;   
      end;   
      adres: record   
            street: string[15];   
            dom: byte;   
            kvart: byte;   
      end;   
   end;   
Begin   
   with student do   
      begin   
         fam:= 'Иванов';   
         with data do   
            begin   
               day:= 30;   
               mes:= 4;   
               year:= 1987;   
            end;   
         with adres do   
            begin   
               street:= 'Туполева';   
               dom:= 22;   
               kvart:= 154;   
            end;   
      end;   
End.   
Для использования в программе набора с одинаковыми полями используются массивы записей.   
Пример: объявить массив из десяти записей.   
1 вариант решения:   
var A: array [1..10] of record   
      fam: string;   
      name: string;   
   end;   
2 вариант решения:   
type student = record   
      fam: string;   
      name: string;   
   end;   
var A: array [1..10] of student;



**Лекция 11 Объектно - ориентированное программирование**

Тип объект.   
  
Объект можно рассматривать как усовершенствование типа запись, в которой описание свойств и параметры моделируемой сущности дополняются методами - описаниями действий с объектом. В отличие от записи объект объявляется словом object.   
Пример: создадим простейший объект: позицию на экране в графическом режиме:   
program oop;   
uses graph;   
type pozicia = object   
      x, y: integer;   
      procedure init (xn, yn: integer);   
      procedure locate (var xl, yl: integer);   
   end;   
procedure pozicia.init;   
begin   
   x:=xn;   
   y:=yn;   
end;   
procedure pozicia.locate;   
begin   
   xl:=x;   
   yl:=y;   
end;   
var d, r, xx, yy: integer;   
   p: pozicia;   
begin   
   d:=detect;   
   randomize;   
   initgraph (d, r, 'c:\tp\bgi');   
   p.init (random(GetMaxX), random(GetMaxY));   
   closegraph;   
   p.locate (xx, yy);   
   write (xx, yy);   
end.   
  
  
 Инкапсуляция.   
  
Одним из главных свойств ООП является инкапсуляция - замыкание в общей оболочке (Object…end) всех составляющих описания. При этом поля оказываются глобальными для методов данного объекта, т.к. у полей и методов общая область действия, то совпадение имен полей и формальных параметров методов не допустимо. Блоки-методы вынесены за описание типа объект. Имена блоков-методов, принадлежащих разным типам могут совпадать. Даже при совпадении имен заголовки методов будут различны, т.к. состоят из префикса (имени типа) и имени метода.   
Доступ к полям объектов из вне можно принудительно ограничивать. Для этого группа полей в описании объекта заключается в скобки Private Public. После этого поля окажутся доступными лишь методам данного модуля.   
  
  
 Наследование.   
  
Примитивные объекты не используются как програмные модули, а используются в качестве носителей общих свойств и методов. Такие объекты называют родительскими. Объекты основанные на родительских называют дочерними. Родительский тип не используемый для описания переменных называется абстрактным. Тип потомок наследует все поля типа отца. В их числе все поля унаследованные отцом, если у него есть предки. Увеличение числа полей у потомка необязательно. Наследоваться также могут и методы, но выборочно. Описание типа потомка имеют отличительную деталь - имя типа отца:   
   <имя типа потомка>=object(<имя типа отца>)   
С увеличением сложности объектов увеличивается число действий, которое можно заменить построением нового метода, причем имена методов создаются так, как если бы объекты не имели между собой родственной связи. Одинаковое обозначение функционально-подобных методов упрощает не только восприятие системы объектов, но и программирование.   
Важной деталью использования наследования в программах является применение присваивания объектам значений объектов. Присваивание A:=B допустимо, если A и B - однотипны, A - предок B или для каждого поля A есть соответствующее поле в B.   
  
  
 Полиморфизм.   
  
Полиморфизм предполагает определение класса или нескольких классов для родственных объектных типов так, что каждому классу отводится своя функциональная роль. Методы одного класса обычно наделяются общим именем. В ситуации когда необходимо сложный метод использовать в нескольких объектах и различия в поведении объектов минимальны, возможно создание смежного сложного метода с вынесением различий в сменные подчиненные методы. Такой метод называется конструктивным полиморфизмом. Осуществляется эта идея созданием виртуальных сменных методов. В заголовке такого метода присутствует слово virtual, а для их подключения к общему методу - обращение к конструктору - блоку со специальным заголовком.   
   constructor <имя блока> (<список формальных параметров>)   
К конструктору надо обращаться для каждого объекта использующего виртуальные методы.   
Задача: тип kom - сын типа pozicia представляет закрашенные квадраты с длиной стороны raz (в пикселах). Наследуемые поля x, y являются координатами центра квадрата. Процедура kom.zoom увеличивает (уменьшает) объект если аргумент znak>0 (znak<=0). Длина raz изменяется на 2\*delt, где delt - еще один аргумент.   
  
uses graph, crt;   
type pozicia = object   
      x, y: integer;   
      procedure init (xn, yn: integer);   
      procedure locat (var xl, yl: integer);   
   end;   
   kom=object (pozicia)   
      cvet, raz: word;   
      procedure init (xn, yn: integer; color: word);   
      procedure zoom (delt, znak: integer);   
   end;   
procedure pozicia.init;   
begin   
   x:=xn;   
   y:=yn;   
end;   
procedure pozicia.locat;   
begin   
   xl:=x;   
   yl:=y;   
end;   
procedure kom.init;   
begin   
   pozicia.init (xn, yn);   
   cvet:=color;   
   raz:=1;   
end;   
procedure kom.zoom;   
var j, d: integer;   
begin   
   if znak>0 then setcolor (cvet)   
      else setcolor (getBkcolor);   
   for j:=1 to delt do   
      begin   
         if znak>0 then raz:=raz+2;   
         d:=raz div 2;   
         moveto (x-d, y-d);   
         linerel (d+d, 0);   
         linerel (0, d+d);   
         linerel (-d-d, 0);   
         linerel (0, -d-d);   
         if (znak<=0) and (raz>1) then raz:=raz-2;   
      end;   
end;   
const n=50;   
var j, d, r, xx, yy: integer;   
      kvad: array [1..n] of kom;   
begin   
      d:=detect;   
      randomize;   
      initgraph (d, r, 'c:\tp\bgi');   
      for j:=1 to n do   
         kvad[j].init (random(GetMaxX), random(GetMaxY), random(GetMaxColor);   
      repeat   
         delay (100);   
         j:=random (n)+1;   
         kvad[j].zoom(random(8)+1, random(3)-1);   
      until kepressed;   
      closegraph;   
      kvad[1].locat (xx, yy);   
      write (xx, ' ', yy);   
      readln;