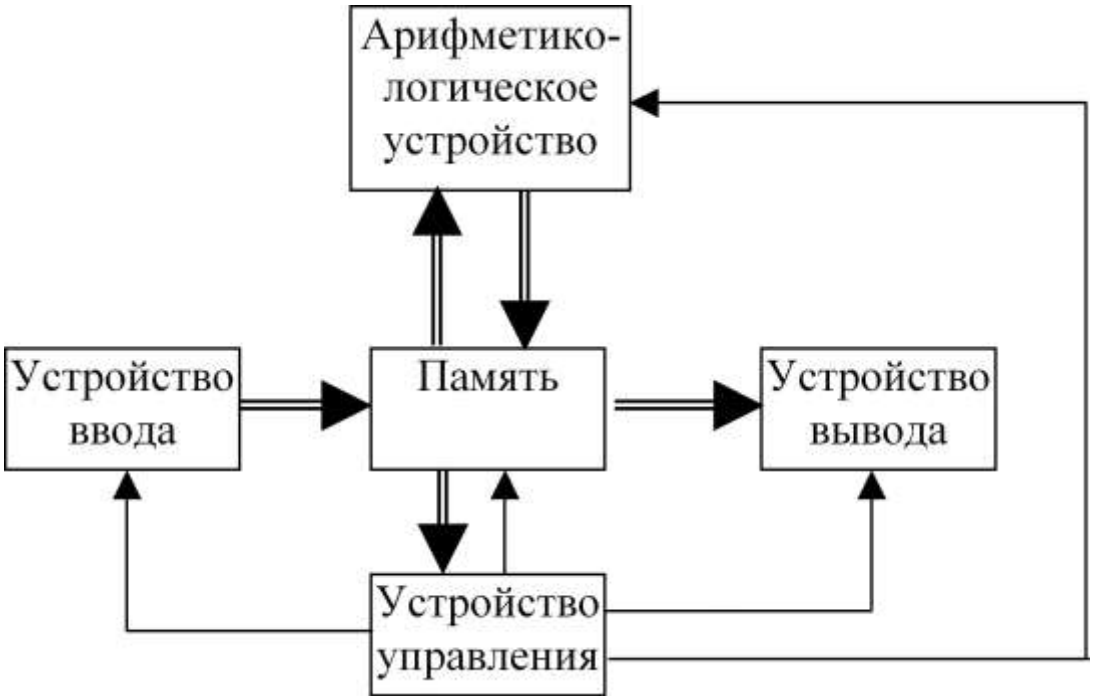
ЛЕКЦИИ ПО ДИСЦИПЛИНЕ «ИНФОРМАТИКА»

Стенгач Михаил Сергеевич

ЛЕКЦИЯ 1

Архитектура ЭВМ
Структура компьютера и принципы его функционирования.

В 1946 - 1948 годах в Принстонском университете (США) коллективом исследователей под руководством Джона фон Неймана был разработан проект ЭВМ, который никогда не был реализован, но идеи которого используются и по сей день. Этот проект получил название машины фон Неймана или Принстонской машины.
Структура Принстонской машины:



Несмотря на огромное разнообразие вычислительной техники и ее необычайно быстрое совершенствование, фундаментальные принципы устройства машин во многом остаются неизменными. В частности, начиная с самых первых поколений, любая ЭВМ состоит из следующих основных устройств: **процессор, память** (внутренняя и внешняя) и **устройства ввода и вывода** информации. Рассмотрим более подробно назначение каждого из них.

Процессор является главным устройством компьютера, в котором собственно и происходит обработка всех видов информации. Другой важной функцией процессора является обеспечение согласованного действия всех узлов, входящих в состав компьютера. Соответственно наиболее важными частями процессора являются **арифметико-логическое устройство АЛУ** и **устройство управления УУ**.

Каждый процессор способен выполнять вполне определенный набор универсальных инструкций, называемых чаще всего **машинными командами**. **Работа ЭВМ** состоит в выполнении последовательности таких команд, подготовленных в виде программы. Процессор способен организовать считывание очередной команды, ее анализ и выполнение, а также при необходимости принять данные или отправить результаты их обработки на требуемое устройство.

Хотя внутри процессора всегда имеются специальные ячейки (регистры) для оперативного хранения обрабатываемых данных и некоторой служебной информации, в нем сознательно не предусмотрено место для хранения программы. Для этой важной цели в компьютере служит другое устройство - **память.**

Память в целом **предназначена для** хранения, как данных, так и программ их обработки: согласно фундаментальному принципу фон Неймана, для обоих типов информации используется единое устройство.

Начиная с самых первых ЭВМ, память сразу стали делить на **внутреннюю** и **внешнюю.** Исторически это действительно было связано с размещением внутри или вне процессорного шкафа. Однако с уменьшением размеров машин внутрь основного процессорного корпуса удавалось поместить все большее количество устройств, и первоначальный непосредственный смысл данного деления постепенно утратился. Тем не менее, терминология сохранилась.

Под **внутренней памятью** современного компьютера принято понимать быстродействующую электронную память, расположенную на его системной плате. Наиболее существенная часть внутренней памяти называется **ОЗУ - оперативное запоминающее устройство**. Его главное назначение состоит в том, чтобы хранить данные и программы для решаемых в текущий момент задач. При выключении питания содержимое ОЗУ полностью теряется. В состав внутренней памяти современного компьютера помимо ОЗУ также входят и некоторые другие разновидности памяти, которые при первом знакомстве можно пропустить. Здесь упомянем только о **постоянном запоминающем устройстве (ПЗУ)**, в котором в частности хранится информация, необходимая для первоначальной загрузки компьютера в момент включения питания. Как очевидно из названия, информация в ПЗУ не зависит от состояния компьютера (для лучшего понимания можно указать на некоторую аналогию между информацией в ПЗУ и "врожденными" безусловными рефлексами у живых существ). Раньше содержимое ПЗУ раз и навсегда формировалось на заводе, теперь же современные технологии позволяют, в случае необходимости, обновлять его, даже не извлекая из компьютерной платы.

Внешняя память реализуется в виде довольно разнообразных устройств хранения информации и обычно конструктивно оформляется в виде самостоятельных блоков. Сюда, прежде всего, следует отнести накопители на магнитных дисках (винчестерами), а также оптические дисководы (устройства для работы с CD ROM). В конструкции устройств внешней памяти имеются механически движущиеся части, поэтому скорость их работы существенно ниже, чем у полностью электронной внутренней памяти. Тем не менее, внешняя память позволяет сохранить огромные объемы информации с целью последующего использования.

Если процессор дополнить памятью, то такая система уже может быть работоспособной. Ее существенным недостатком является невозможность узнать что-либо о происходящем внутри такой системы. Для получения информации о результатах, необходимо дополнить компьютер **устройствами вывода**, которые позволяют представить их в доступной человеческому восприятию форме. Наиболее распространенным устройством вывода является дисплей, способный быстро и оперативно отображать на своем экране как текстовую, так и графическую информацию. Для того чтобы получить копию результатов на бумаге, используют печатающее устройство, или принтер.

Наконец, поскольку пользователю часто требуется вводить в компьютерную систему новую информацию, необходимы еще и **устройства ввода**. Простейшим устройством ввода является клавиатура. Другие устройства ввода: манипулятор мышь, сканнер.

ЛЕКЦИЯ 2

Основные положения теории информации. Понятие информации.

Термин "информация" происходит от латинского слова "informatio", что означает сведения, разъяснения, изложение. Несмотря на широкое распространение этого термина, понятие информации является одним из самых дискуссионных в науке. В настоящее время наука пытается найти общие свойства и закономерности, присущие многогранному понятию информация, но пока это понятие во многом остается интуитивным и получает различные смысловые наполнения в различных отраслях человеческой деятельности:

в обиходе информацией называют любые данные или сведения, которые кого-либо интересуют. Например, сообщение о каких-либо событиях, о чьей-либо деятельности и т.п. "Информировать" в этом смысле означает "сообщить нечто, неизвестное раньше";

в технике под информацией понимают сообщения, передаваемые в форме знаков или сигналов;

в кибернетике под информацией понимает ту часть знаний, которая используется для ориентирования, активного действия, управления, т.е. в целях сохранения, совершенствования, развития системы (Н. Винер).

Клод Шеннон, американский учёный, заложивший основы теории информации — науки, изучающей процессы, связанные с передачей, приёмом, преобразованием и хранением информации, — рассматривает информацию как снятую неопределенность наших знаний о чем-то.

Современное научное представление об информации очень точно сформулировал Норберт Винер, "отец" кибернетики. А именно:

Информация — это обозначение содержания, полученного из внешнего мира в процессе нашего приспособления к нему и приспособления к нему наших чувств.

Люди обмениваются информацией в форме сообщений. Сообщение — это форма представления информации в виде речи, текстов, жестов, взглядов, изображений, цифровых данных, графиков, таблиц и т.п.

Одно и то же информационное сообщение (статья в газете, объявление, письмо, телеграмма, справка, рассказ, чертёж, радиопередача и т.п.) может содержать разное количество информации для разных людей — в зависимости от их предшествующих знаний, от уровня понимания этого сообщения и интереса к нему.

Так, сообщение, составленное на японском языке, не несёт никакой новой информации человеку, не знающему этого языка, но может быть высокоинформативным для человека, владеющего японским. Никакой новой информации не содержит и сообщение, изложенное на знакомом языке, если его содержание непонятно или уже известно.

Информация есть характеристика не сообщения, а соотношения между сообщением и его потребителем. Без наличия потребителя, хотя бы потенциального, говорить об информации бессмысленно.

В случаях, когда говорят об автоматизированной работе с информацией посредством каких-либо технических устройств, обычно в первую очередь интересуются не содержанием сообщения, а тем, сколько символов это сообщение содержит.

Применительно к компьютерной обработке данных под информацией понимают некоторую последовательность символических обозначений (букв, цифр, закодированных графических образов и звуков и т.п.), несущую смысловую нагрузку и представленную в понятном компьютеру виде. Каждый новый символ в такой последовательности символов увеличивает информационный объём сообщения.

Вид информации

Информация может существовать в виде:

- текстов, рисунков, чертежей, фотографий;
- световых или звуковых сигналов;
- радиоволн;
- электрических и нервных импульсов;
- магнитных записей;

Предметы, процессы, явления материального или нематериального свойства, рассматриваемые с точки зрения их информационных свойств, называются информационными объектами.

Передача информации

Информация передаётся в форме сообщений от некоторого источника информации к её приёмнику посредством канала связи между ними. Источник посылает передаваемое сообщение, которое кодируется в передаваемый сигнал. Этот сигнал посылается по каналу связи. В результате в приёмнике появляется принимаемый сигнал, который декодируется и становится принимаемым сообщением.

Пример:

Сообщение, содержащее информацию о прогнозе погоды, передаётся приёмнику (телезрителю) от источника — специалиста-метеоролога посредством канала связи — телевизионной передающей аппаратуры и телевизора.

Измерение информации

Какое количество информации содержится, к примеру, в тексте романа "Война и мир", во фресках Рафаэля или в генетическом коде человека? Ответа на эти вопросы наука не даёт и, по всей вероятности, даст не скоро. А возможно ли объективно измерить количество информации? Важнейшим результатом теории информации является следующий вывод:

«В определенных, весьма широких условиях можно пренебречь качественными особенностями информации, выразить её количество числом, а также сравнить количество информации, содержащейся в различных группах данных».

В настоящее время получили распространение подходы к определению понятия "количество информации", основанные на том, что информацию, содержащуюся в сообщении, можно нестрого трактовать в смысле её новизны или, иначе, уменьшения неопределённости наших знаний об объекте. Эти подходы используют математические понятия вероятности и логарифма.

В качестве единицы информации Клод Шеннон предложил принять один бит (англ. bit — binary digit — двоичная цифра).

Бит в теории информации — количество информации, необходимое для различения двух равновероятных сообщений (типа "орел"—"решка", "чет"—"нечет" и т.п.).
В вычислительной технике битом называют наименьшую "порцию" памяти компьютера, необходимую для хранения одного из двух знаков "0" и "1", используемых для внутримашинного представления данных и команд.

Бит — слишком мелкая единица измерения. На практике чаще применяется более крупная единица — байт, равная восьми битам. Именно восемь битов требуется для того, чтобы закодировать любой из 256 символов алфавита клавиатуры компьютера (256=28).

Широко используются также ещё более крупные производные единицы информации:

1 Килобайт (Кбайт) = 1024 байт = 210 байт,

1 Мегабайт (Мбайт) = 1024 Кбайт = 220 байт,

1 Гигабайт (Гбайт) = 1024 Мбайт = 230 байт.

В последнее время в связи с увеличением объёмов обрабатываемой информации входят в употребление такие производные единицы, как:

1 Терабайт (Тбайт) = 1024 Гбайт = 240 байт,

1 Петабайт (Пбайт) = 1024 Тбайт = 250 байт.

Обработка информации

Обработка информации — получение одних информационных объектов из других информационных объектов путем выполнения некоторых алгоритмов.

Обработка является одной из основных операций, выполняемых над информацией, и главным средством увеличения объёма и разнообразия информации.

Средства обработки информации — это всевозможные устройства и системы, созданные человечеством, и в первую очередь, компьютер — универсальная машина для обработки информации.

Компьютеры обрабатывают информацию путем выполнения некоторых алгоритмов.

Понятие алгоритма

Алгоритм - четкое описание последовательности действий, которые необходимо выполнить при решении задачи. Можно сказать, что алгоритм описывает процесс преобразования исходных данных в результаты, т.к. для решения любой задачи необходимо:

1. Ввести исходные данные.
2. Преобразовать исходные данные в результаты (выходные данные).
3. Вывести результаты.

Разработка алгоритма решения задачи - это разбиение задачи на последовательно выполняемые этапы, причем результаты выполнения предыдущих этапов могут использоваться при выполнении последующих. При этом должны быть четко указаны как содержание каждого этапа, так и порядок выполнения этапов. Отдельный этап алгоритма представляет собой либо другую, более простую задачу, алгоритм решения которой известен (разработан заранее), либо должен быть достаточно простым и понятным без пояснений. Разработанный алгоритм можно записать несколькими способами:

- на естественном языке;
- в виде блок-схемы;

Рассмотрим пример алгоритма на естественном языке:

1. Ввести в компьютер числовые значения переменных **a, b и c**.
2. Вычислить d по формуле $\mathbf{d = b^2 - 4ac}$.
3. Если $\mathbf{d < 0}$, то напечатать сообщение "Корней нет" и перейти к п.4. Иначе вычислить и напечатать значения $\mathbf{x_1}$ **и** $\mathbf{x_2}$.
4. Прекратить вычисления.

1.2. Изображение алгоритма в виде блок-схемы

Блок-схемой называется наглядное графическое изображение алгоритма, когда отдельные его этапы изображаются при помощи различных геометрических фигур - блоков, а связи между этапами (последовательность выполнения этапов) указываются при помощи стрелок, соединяющих эти фигуры. Блоки сопровождаются надписями. Типичные действия алгоритма изображаются следующими геометрическими фигурами:
Блок начала-конца алгоритма. Надпись на блоке: "начало" ("конец").
Блок ввода-вывода данных. Надпись на блоке: слово "ввод" ("вывод" или "печать") и список вводимых (выводимых) переменных.

Блок начала-конца алгоритма

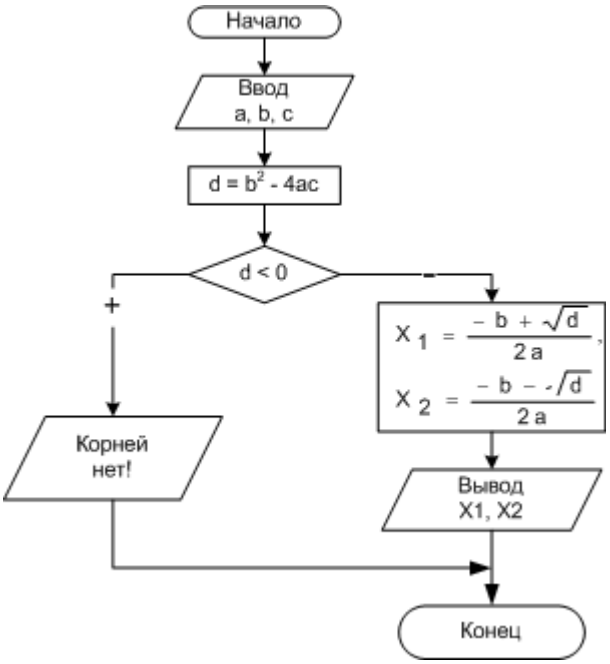
Блок ввода-вывода данных

Блок решения или **арифметический** . Надпись на блоке: операция или группа операций.
Условный блок.Надпись на блоке: условие. В результате проверки условия осуществляется выбор одного из возможных путей (ветвей) вычислительного процесса. Если условие выполняется, то следующим выполняется этап по ветви "+", если условие не выполняется, то выполняется этап по ветви "–".

Арифметический блок

Условный блок

В качестве примера рассмотрим блок-схему алгоритма решения квадратного уравнения



ЛЕКЦИЯ 3

* * *

В курсе лекций изучается программирование на языке Паскаль (с 2003 года – язык Delphi) и работа в визуальной среде программирования Delphi.

Каждая группа имеет на диске `D` в папке `KC` собственную папку, например `D:\KC\515`. Папки с проектами лабораторных работ должны храниться только в этой папке.

Алфавит языка Паскаль

Алфавит Паскаля содержит следующие символы:

- Заглавные и строчные латинские буквы и символ "подчеркивание":

`A…Z, a…z, _`

- Арабские цифры: `0…9`
- Двадцать два специальных символа:

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

Русские буквы и другие символы могут использоваться только для записи комментариев и в символьных константах.

Структура программы. Комментарии

В общем виде программа на Паскале состоит из заголовка (содержит слово `program` и имя программы), раздела описаний и выполняемой части.

Раздел описаний начинается сразу за заголовком программы. Выполняемая часть начинается со слова `BEGIN` и заканчивается словом `END`, после которого обязательно ставится точка (конец программы):

```
program <имя программы>;
   < РАЗДЕЛ ОПИСАНИЙ >
BEGIN
   < ВЫПОЛНЯЕМАЯ ЧАСТЬ >
END.
```

Операторы могут располагаться в программе произвольным образом, но обязательно должны заканчиваться `точкой с запятой.`

Для пояснения текста программы в нее могут включаться комментарии. Комментарий – это произвольная последовательность символов, находящаяся:

1. между фигурными скобками.
2. между круглыми скобками со звездочками: `(* … *).`
3. после двух символов `//.`

Пример простой программы:

```
program primer;                 (* заголовок программы *)
  var x,y,z:real;               // описание переменных
begin                         { начало выполняемой части }
  readln(x,y);                        // ввод чисел X и Y
  z:=x*y;                                // вычисление Z
  writeln(z);              // вывод на экран значения Z
end.                                    { конец программы }
```

Консольные приложения в среде программирования Delphi

После запуска Delphi на экране появляются несколько окон, из которых нас пока интересуют только два: Главное окно и Окно кода программы.

Главное окно Delphi осуществляет управление проектом. Здесь располагается главное меню Delphi.

Командным кнопкам соответствуют следующие горячие клавиши:

F9 – компилирует и выполняет программу,

Ctrl+F2 – завершение работы программы в случае зависания,

F8 – пошаговое выполнение программы,

F7 – пошаговое выполнение программы с отслеживанием работы вызываемых подпрограмм.

Окно кода предназначено для создания и редактирования текста программы. Для создания консольного приложения необходимо выполнить File=>New=>Other… и в появившемся диалоговом окне на странице New выбрать Console Application. Появится окно редактора кода. В окне находится шаблон кода проекта, которому присваивается по умолчанию имя Project2.dpr. Первоначально окно кода содержит минимальный исходный текст (версия Delphi 7):

```
program Project2;
{$APPTYPE CONSOLE}
uses
  SysUtils;
begin
  { TODO -oUser -cConsole Main : Insert code here }
end.
```

Идентификаторы

Идентификаторы в языке Паскаль – это имена констант, переменных, меток, типов, процедур и функций. Идентификатор может содержать буквы, цифры и знак подчеркивания и начинаться только с буквы или знака подчеркивания. Пробелы и специальные символы не могут входить в идентификатор.

Важно помнить, что соответствующие заглавные и строчные буквы в идентификаторах и служебных словах не различаются. Таким образом, следующие три идентификатора обозначают одну и ту же переменную: index, INDEX, IndEx.

В качестве идентификаторов нельзя использовать служебные слова. Служебные слова будут рассматриваться по мере изучения языка.

Пример идентификаторов:

правильно	не правильно
punkt_1	1_punkt
nomer1	#1
fort	for (служебное слово)

Оператор присваивания

Оператор присваивания имеет вид: <переменная>:=<выражение>;

где символ := означает присвоить.

Пример:

```
a:=2; b:=3;
c:=a+b;  {c=5}
```

Основные типы данных в Паскале

REAL - ВЕЩЕСТВЕННЫЕ ЧИСЛА. Это числа, содержащие целую и дробную части, разделяемые точкой. Для типа REAL используются следующие арифметические операции:

\+ сложение; - вычитание; * умножение; / деление.

Список операций приведен в порядке повышения приоритета. Так, например, в выражении a:=2+6/2*3; сначала выполнится деление 6/2=3, затем умножение 3*3=9, а затем сложение 2+9=11. Выражение в скобках имеет высший приоритет и выполняется первым.

Пример:

```
a:=2+6/(2*3); {рез.=3}
a:=(2+6)/2*3; {рез.=12}
```

INTEGER - ЦЕЛЫЕ ЧИСЛА. Для типа INTEGER вместо операции деления / определены две специальные операции:

DIV деление с отбрасыванием дробной части;

MOD взятие остатка от целочисленного деления.

Пример:

```
b:=5 div 2; {рез.=2}
b:=5 mod 2; {рез.=1}
```

BOOLEAN - ЛОГИЧЕСКИЙ ТИП. Объекты типа `BOOLEAN` могут принимать только два значения: `TRUE` - "истина", и `FALSE` - "ложь".

CHAR - СИМВОЛЫ. В переменную этого типа может быть помещен любой символ. Значение переменной `CHAR` задается в апострофах.

Пример: `a:='a';`

STRING - СТРОКИ. Массив символов.

Пример: `a:='абракадабра';`

Переменные

Каждая переменная должна быть описана в разделе описаний. Описание переменных начинается со слова `VAR`.

```
VAR <имя_переменной>: <тип_переменной>;
```

Например:

```
Var a: real;
    i,j: integer;
    b: Boolean;
```

Константы

Константы описываются в разделе описаний. Описание константы начинается со слова `CONST`. Значение константы нельзя изменить в теле программы.

Например:

```
Const  e=2.71828;    // Раздел
Var    x: real;      // описаний
       begin
         x:=e;       // Допустимое действие
         e:=3.14;    // Ошибка!
       end.
```

Основные математические функции

Стандартные математические функции языка Паскаль:

Идентификатор (имя) функции	Функция
`PI`	Число π
`Abs(x)`	Абсолютная величина |x|
`Exp(x)`	Экспонента e^x
`Sqr(x)`	x^2
`Sqrt(x)`	Квадратный корень x
`Ln(x)`	Натуральный логарифм *ln* x
`Sin(x)`	*Sin* x (угол в радианах)
`Cos(x)`	*Cos* x (угол в радианах)
`ArcTan(x)`	*Arctg* x (значение в радианах)
`Round(x)`	Округляет число до ближайшего целого
`Trunc(x)`	Отбрасывает дробную часть числа
`Int(x)`	Целая часть числа
`Frac(x)`	Дробная часть числа
`Randomize`	Инициация счётчика случайных чисел
`Random(x)`	Случайное число в диапазоне 0…(x-1) (число x – целое)

Очень много математических функций добавлено в Delphi. Все они находятся в модуле math, который необходимо подключить:

```
uses SysUtils, math;
```

Некоторые из функций модуля math:

`Power(x,y)`	x^y
`Log10(x)`	Десятичный логарифм *lg* x
`LogN(n,x)`	$log_n x$
`RandomRange(min, max)`	Случайное число в диапазоне min…max (числа max и min – целые)

Процедуры ввода и вывода

Операторы вывода:

`WRITE`. Процедура `Write` выводит на экран выражения без перевода строки. Символьные (текстовые) константы заключаются в апострофы. Например, в результате выполнения следующего фрагмента:

```
A := 7;
Write('A=', a);
Write('The end.');
```

на экране будет напечатано следующее:

```
A= 7The end.
```

`WRITELN`. Процедура `Writeln` отличается от `Write` только тем, что она после завершения вывода переводит текущую позицию в начало следующей строки. Например:

```
A := 7;
Writeln ('A=', a);
Write('The end.');
```

В результате выполнения этого фрагмента на экране будет напечатано следующее:

```
A= 7
The end.
```

Переменные типа `real` выводятся с помощью оператора `write` в экспоненциальной форме. Например:

```
c:=13.4;
write('C=',c);
```

На экран выведется:

```
C=1.34000000000000E+0001
```

Для вывода чисел в нормальной форме необходимо использовать форматный вывод: после имени переменной ставится ":" и указывается общее количество позиций под число, и, через еще одно ":" – количество позиций под дробную часть. Например:

```
write('C=',c:8:3);
```

На экран выведется:

```
C= 13.400
```

Операторы ввода:

`READ`. Процедура `Read` выполняет ввод с клавиатуры значений переменных.

Пример:

`Read(a);` - в этом месте выполнение программы приостанавливается, программа ожидает ввода с клавиатуры переменной `a`.

`READLN`. Процедура `Readln` отличается от `Read` только тем, что после завершения ввода она переводит текущую позицию в начало следующей строки.

ЛЕКЦИЯ 4

Логические выражения

В логических выражениях используются следующие операции сравнения:

< меньше; > больше; = равно;
<= меньше или равно; >= больше или равно; <> не равно.

Результатом операции сравнения является "истина" (TRUE) или "ложь" (FALSE).

Например:

(X+1)<>2 - имеет значение "ложь", если X=1, и значение "истина", если значение переменной X отлично от 1.

Логические операции:

NOT логическое НЕ;
AND логическое И;
OR логическое ИЛИ.

Запись таких логических выражений, как min < X < max осуществляется следующим образом: (min < X) and (X < max)

Результатом логической операции and является "истина", когда "истине" равны оба сравнения.

Условный оператор IF

Оператор IF предназначен для выбора одного из двух возможных действий в зависимости от некоторого условия.

Структура условного оператора:

```
IF   <условие>
THEN <оператор_1>
ELSE <оператор_2>;
```

где <условие> – логическое выражение.
Если <условие> истинно (TRUE) выполняется <оператор_1>, иначе (<условие>=FALSE) выполняется <оператор_2>.

Простой пример:

```
Program primer_1;
  Var  a: real;
  Begin
    readln(a);
    if a>=0
      then writeln('positive number')
      else writeln('negative number');
  End.
```

Условный оператор может быть также без ELSE-альтернативы:

```
IF   <условие>
THEN <оператор>;
```

Составной оператор

В условном операторе IF после THEN и ELSE можно выполнить только один оператор. Чтобы выполнить группу операторов нужно использовать *составной оператор,* т.е. заключить группу операторов между BEGIN и END;.

Пример. Вычислить и распечатать значение k только для случая a>b

```
if a>b then
          begin
            k:=a+b;
            writeln('k=',k);
          end;
```

Оператор безусловного перехода GOTO. Метки

Оператор GOTO передаёт управление оператору, которому предшествует соответствующая *метка*.

Метка – это идентификатор или целое число.

Каждая метка должна быть предварительно описана в разделе описаний:

```
LABEL <метка>;
```

Структура оператора:

```
GOTO  <метка>;
<метка>: <оператор>;
```

Один оператор может быть помечен несколькими метками, которые в этом случае отделяются друг от друга двоеточиями.

Пример:

```
Program KPSS;
Var  Vodka, Sozn: Real;
LABEL  1, Stop;
  Begin
 1: Vodka:=3.62; Sozn:=0;
    IF Sozn>Vodka then GOTO Stop
                  else GOTO 1;
Stop: writeln('победа коммунизма')
  End.
```

Видно, что приведенная программа при выполнении зацикливается. Sozn никогда не будет больше vodka и программа всегда будет возвращаться на метку 1.

Т.к. подобные ошибки субъективного характера могут возникать при использовании GOTO, применение на практике этого оператора ограничено.

Оператор выбора CASE

Оператор выбора позволяет выбрать одно из нескольких возможных продолжений программы.

Структура оператора:

```
CASE <ключ_выбора> OF
     <выбор_1>: <оператор_1>;
     <выбор_2>: <оператор_2>;
     . . .
     <выбор_k>: <оператор_k>;
 ELSE <оператор>
 END;
```

где

<ключ_выбора> - значение переменной, которая проверяется оператором CASE;

<выбор_1...k> - переменные того же типа, что и <ключ_выбора>.

В последовательности операторов <выбор_1...k> отыскивается такой, значение которого равно значению <ключ_выбора>. Соответствующий найденному выбору оператор выполняется, после чего оператор CASE завершает свою работу. Если переменная, равная значению <ключ_выбора>, не будет найдена, управление передается оператору, стоящему за словом ELSE.

Часть ELSE <оператор> может отсутствовать.

Любому из операторов от 1 до k может предшествовать не одна, а несколько значений <выбора>, разделенных запятыми.

Пример

```
Program SGAU;
  Var student: integer;
    BEGIN
         readln(student);
      CASE student of
```

```
        0:      WriteLn('политех');
        1,2,3:  WriteLn('Самолет');
        4:      WriteLn('чугун');
        5:      WriteLn('THE BEST');
      Else      WriteLn('...!');
    End;
END.
```

ОПЕРАТОРЫ ЦИКЛА

Цикл - конструкция языка, позволяющая несколько раз выполнять один оператор или группу операторов.

В Паскале имеется три оператора цикла: WHILE, REPEAT и FOR.

Для того чтобы прервать выполнение цикла используется оператор break.

Оператор цикла WHILE

Структура:

```
WHILE  <условие>  DO  <оператор>;
```

где <условие> – логическое выражение.

Если <условие> истинно (TRUE) выполняется <оператор>, который будет выполняться в цикле до тех пор, пока <условие> не станет ложным (FALSE).

Таким образом, цикл выполняется пока <условие> истинно.

Пример. Программа табулирования функции $y=ax^2$:

```
var x,xk,dx,y,a:real;
begin
write('xn=');readln(x);
write('xk=');readln(xk);
write('dx=');readln(dx);
write('a='); readln(a);
while x<=xk do
  begin
  y:=a*x*x;
  writeln(x:7:2,' ',y:7:2);
  x:=x+dx;
  end;
readln;
end.
```

Оператор цикла REPEAT

Структура:

```
REPEAT  <оператор_1>; <оператор_2>; ... <оператор_k>;
UNTIL   <условие>;
```

Отличия REPEAT от ранее рассмотренного оператора цикла WHILE:

1. Условие проверяется в конце (т.о. цикл выполняется хотя бы один раз).

2. Признаком окончания цикла является выполнение условия (имеет значение TRUE). Таким образом, цикл выполняется пока условие ложно.

3. В теле цикла может содержаться произвольное количество операторов (REPEAT и UNTIL - ограничители).

Если для организации цикла использовать оператор REPEAT…UNTIL, то приведенный выше пример табулирования функции $y=ax^2$ будет иметь вид:

```
repeat
  y:=a*x*x;
  writeln(x:7:2,' ',y:7:2);
  x:=x+dx;
until x>xk;
```

ЛЕКЦИЯ 5

Оператор цикла FOR

Структура:

```
FOR <параметр>:=<нач.знач.> TO <кон.знач.> DO <оператор>;
```

где <параметр> - переменная цикла типа INTEGER или CHAR.

Значение <параметра> изменяется в цикле от <нач.знач.> до <кон.знач.> с шагом равным 1 и это определяет количество выполнений <оператора>.

Пример.

```
for i:=1 to 3 do write(i:2);
```

Переменная i изменяется в цикле от 1 до 3 с шагом 1 и оператор write выполняется 3 раза.

Результат работы оператора for: 1 2 3.

Существует другая форма оператора цикла:

```
FOR <параметр>:=<нач.знач.> DOWNTO <кон.знач.> DO <оператор>;
```

Замена служебного слова TO на DOWNTO означает, что шаг изменения переменной <параметр> равен -1.

МАССИВЫ

Массив - это совокупность данных одного типа, занимающая непрерывную область оперативной памяти. Массив имеет имя и состоит из ячеек. Каждая ячейка имеет свой номер (индекс массива). Индексы у массива могут иметь тип integer или char.

Чтобы осуществить доступ к ячейке массива, нужно указать имя массива и номер ячейки в квадратных скобках.

Одномерный массив

Описание массива:

```
Var a: array [1..7] of Integer;
Const b: array [0..3] of Real=(1.1,1.2,-3.2,0.5);
```

	1	2	3	4	5	6	7	
a	-2	4	6	8	-5	12	17	a[6]:=1 2;

	0	1	2	3
b	1.1	1.2	-3.2	0.5

В качестве индекса массива можно использовать арифметические выражения, имеющие целый результат

```
i:=2;
a[2*i]:=8;          // a[4]:=8;
b[2*i-1]:=0.5;      // b[3]:=0.5;
```

Примеры

1. Ввод элементов массива

```
Var a: array [1..10] of Integer;
    i: Integer;
begin
    for i:=1 to 10 do
        begin
        Write('a[',i:2,']=');
        ReadLn(a[i]);
        end;
end.
```

2. Вывод элементов массива

```
for i:=1 to 10 do  WriteLn('a[',i,']=',a[i]);
```

3. Операция присваивания с массивами

Для массивов одного типа и одной размерности.

```
Var a,b: array [1..10] of Integer;
< Ввод массива a >
b:=a;
```

4. Сумма элементов массива

```
S:=0;
for i:=1 to 10 do  S:=S+a[i];
```

5. Произведение элементов массива

```
P:=1;
for i:=1 to 10 do  P:=P*a[i];
```

6. Подсчет отрицательных элементов массива

```
n:=0;
for i:=1 to 10 do
if a[i] < 0 then n:=n+1;
```

7. В массиве из 10 элементов найти самое большое число

```
max:=a[1];
for i:=2 to 10 do
if a[i] > max then max:=a[i];
```

8. Найти порядковый номер ячейки, содержащей самое большое число

```
k:=1;
for i:=2 to 10 do
if a[i] > a[k] then k:=i;
```

10. Сортировка массива - расположить элементы массива либо по возрастанию, либо по убыванию

1	2	3	4	5
-2	4	6	8	-5

```
for i:=1 to 5 do
for j:=1 to 5-i do
  begin
   if a[j] > a[j+1] then
      begin
        r:=a[j];
        a[j]:=a[j+1];
        a[j+1]:=r;
      end;
  end;
```

Двумерные массивы (матрицы)

Двумерный массив - это совокупность ячеек памяти, расположенных по строкам, доступ к каждой ячейке памяти осуществляется путем задания имени матрицы и двух индексов (1-ый - строка, 2-ой - столбец).

Описание:

```
Var a: array [1..3,1..5] of Real;
Const b: array [1..2,1..3] of Integer=((1,2,3),(4,5,6));
```

		1	2	3	4	5
a	1	-2	4	6	8	-5
	2	3	8	4	0	0.1
	3	4.22	3	9.9	6	1.3

При обработке значений, хранящихся в матрице, необходимо использовать двойной цикл.

Примеры

1. Ввод двумерного массива (матрицы) 3x5

```
Var a:   array [1..3, 1..5] of Real;
    i,j: Integer;
begin
    for i:=1 to 3 do
      for j:=1 to 5 do
        begin
          Write('Введите a[',i:2,j:2,']=');
          ReadLn(a[i,j]);
        end;
 end.
```

2. Вывод двумерного массива (матрицы) 3x4

```
    for i:=1 to 3 do
      for j:=1 to 4 do
        Writeln('a[',i:2,j:2,']=',a[i,j]);
```

3. Сумма элементов матрицы

```
S:=0;
for i:=1 to 3 do
for j:=1 to 5 do  S:=S+a[i,j];
```

4. Просуммировать элементы главной диагонали матрицы

```
S:=0;
for i:=1 to 3 do S:=S+a[i,i];
```

5. Транспонирование матрицы

```
for i:=1 to 3 do  // Матрица
for j:=1 to 3 do  // д.б. квадратной
     if i<j then
     begin
        r:=a[i,j];
        a[i,j]:=a[j,i];
        a[j,i]:=r;
     end;
```

Тип данных CHAR

Переменная типа `char` представляет собой любой символ и занимает один байт памяти.

Всего символов 256. Каждому символу соответствует цифровой код в интервале 0..255.

Соответствие между символом и его кодом установлено стандартом ANSI. Первая половина символов с кодами 0..127 стандартная, а вторая половина с кодами 128..255 меняется для различных шрифтов.

Например:

коды 48..57 - символы `'0'...'9'`

коды 97..122 - символы `'a'...'z'`

Для работы с кодами переменных типа `char` используются переменные типа `BYTE`, занимающие 1 байт памяти и предназначенные для хранения целых чисел из диапазона от 0 до 255.

Существуют функции преобразования символа в код и наоборот.

Функция `CHR` Число -> Символ Результат:

```
Var a: byte;                    z
begin
   a:=122;
   writeln(chr(a));
end.
```

Функция ORD Символ -> Число Результат:

```
Var a: byte;                    122
    b: char;
begin
   b:='z';
   a:=ord(b);
   writeln(a);
end.
```

Функция UPCASE преобразует маленькие латинские буквы в большие

```
upcase(a:char): char;
```

ЛЕКЦИЯ 6

СТРОКИ

Строка (`string`) - это последовательность элементов типа `char`.

Описание:

```
Var s: string;
```

Можно описать т.н. короткую строку произвольной длины в пределах 255 символов:

```
Var s: string[N];
```

Эта запись означает, что машина выделяет в памяти N+1 байт.

Например:

```
Var s: string[100];
```

Приведенное описание переменной `s` эквивалентно описанию:

`Var s: array[0..100] of char;` но не идентично.

Количество фактически введенных или существующих символов в переменной типа `string` постоянно находится в её нулевой ячейке. При любых операциях, приводящих к изменению длины строки, число в нулевой ячейке обновляется.

Пример:

```
Var a: string[9];
begin
  a:='самолёт';
end.
```

0	1	2	3	4	5	6	7	8	9	
7	с	а	м	о	л	ё	т			a

`writeln(a[3]);` выведет букву ”м”

`writeln(a[0]);` выведет символ, код которого = 7.

`writeln(ord(a[0]));` выведет цифру 7

В случае массива `CHAR` отслеживания количества символов в нулевой ячейке не происходит, поэтому операции и функции, применимые к данным типа `string`, не подходят к переменным типа `array of char`.

Основные операции для типа string

1. Вывод и ввод строки `writeln(s); readln(s);`

2. Две строки можно сравнивать с помощью условного оператора `if`

```
if a=s then ...
```

Можно применить операцию < или >, тогда строки сравниваются посимвольно. Более длинная строка, в случае совпадения первых символов с более короткой, считается больше.

3. Операция конкатенации - сцепления двух строк.

```
b:='само';
c:='лет';
a:=b+c;
d:=b+'вар';
```

Функции для работы со строками

1. Функция определения длины строки

`k:=lenght(a);` результат - число типа `integer`.

2. Функция `copy(st,from,count)`. Она копирует из строки `st count` символов, начиная с `from`.

```
a:='самолет'; //записать "оле" в другую строку
b:=copy(a,4,3); for i:=1 to 4 do write(b,'! ');
```

3. Процедура `delete(st,from,count)`. Из строки `st` удаляет `count` символов, начиная с символа `from`.

```
a:='арбат'; //надо, чтобы осталось "арба"
delete(a,5,1);
```

4. Процедура insert(subst,st,from) - вставка подстроки subst в строку st, начиная с символа from.

```
b:='шайтан';
insert(a,b,7);
```

5. Функция pos (от position)

```
p:=pos(subst,str);
```

2 входных параметра: строка subst и str. Функция возвращает целое число - позицию первого символа подстроки subst в строке str.

```
str:='гидроэлектростанция';
subst:='электро';
p:=pos(subst,str);
```

Результат: p=6

Если подстрока не обнаружена, то p=0.

Массивы строк

Пример:

```
var s: array [1..4] of string[5]; // массив из 4 строк
                                  // string[5]
one,two,first,second,i,j: integer;
begin
  s[1]:='наука';
  s[2]:='умеет';
  s[3]:='много';
  s[4]:='гитик';
 writeln('In what rows are your cards?');
  readln(one);
  readln(two);
 for i:=1 to 5 do
 for j:=1 to 5 do
     if (s[one,i]=s[two,j]) and (i<>j) then
 begin first:=i; second:=j; end;
 writeln(s[one,first],' ',s[two,second]);
end.
```

ПОДПРОГРАММЫ

Подпрограмма – самостоятельная часть программы, предназначенная для решения конкретной задачи.

В Паскале 2 вида подпрограмм:

1. Функции.
2. Процедуры.

Для того чтобы прервать выполнение подпрограммы используется оператор exit.

Функции

Предназначена для выдачи одного единственного результата. Возвращаемое значение передается из функции с помощью оператора Result или присваивается внутри функции ее имени. Описание функции начинается со слова function.

Структура программы с функцией:

```
<Раздел описаний>
 Function <имя функции>(<описание параметров>):<тип функции>;
 begin
   <тело функции>
   result:=… {или}  <имя функции >:=…
 end;
begin
<тело программы>
<переменная>:=<имя функции>(<список параметров>);
end.
```

Пример подпрограммы-функции:

Программа определения площади треугольника по формуле Герона.

```
Var m,n,k,s: real;
  Function str(a,b,c: real): real;
  Var p: real;
  begin
    p:=(a+b+c)/2;
    str:=sqrt(p*(p-a)*(p-b)*(p-c));
  end;
begin
  writeln('Введите стороны треугольника');
  readln(m,n,k);
  s:=str(m,n,k);    //вызов функции
end.
```

Функция `str` может иметь и такой вид:

```
  Function str(a,b,c: real): real;
  Var p: real;
  begin
    p:=(a+b+c)/2;
    result:=sqrt(p*(p-a)*(p-b)*(p-c));
  end;
```

Формальные и фактические параметры

Параметры, которыми манипулирует функция, называются формальными, а параметры, которые передаются функции при ее вызове из основной программы, называются фактическими. Содержимое фактических параметров при вызове функции копируется формальными параметрами.

В нашем примере `m,n,k` – фактические параметры, `a,b,c` – формальные параметры.

Глобальные и локальные параметры

Глобальные параметры – параметры, определенные в основной программе, доступны в любой точке программы.

Локальные параметры – параметры, определенные в теле подпрограммы, доступны только в самой подпрограмме.

В нашем примере `m,n,k,s` – глобальные параметры, `a,b,c,p` – локальные параметры.

Т.о. обмен информацией между основной программой и подпрограммой может быть реализован через глобальные параметры.

```
Var a,b: integer;         // Глобальные параметры
Function aib: integer;    // Локальных параметров нет
begin
  Result:=a+b;
end;
begin
  readln(a,b);
  writeln('a+b=',aib);
end.
```

Передача параметров в функцию по ссылке

Механизм передачи параметров в функцию по ссылке позволяют сделать доступными в теле функции параметры, объявленные в основной программе. Значения этих параметров могут меняться после окончания работы функции. Для этой цели в заголовок функции перед списком передаваемых ей параметров ставится ключевое слово `Var`.

```
Var a,b,str,hyp: real;
    Function s(var x,y: real): real;
    begin
         Result:=x*y/2;
         x:=sqr(x);
```

```
          y:=sqr(y);
    end;
begin
  readln(a,b);
  str:=s(a,b);    // Площадь прямоуг. треугольника.
  hyp:=sqrt(a+b); // Гипотенуза, т.к. из функции s
end.              // вернулись квадраты переменных a и b
```

ЛЕКЦИЯ 7

Процедуры

Отличия процедуры от функции:

1. Вместо ключевого слова `function` - слово `procedure`;
2. У процедуры не определяется тип;
3. Имеются входные и выходные параметры;
4. Для вызова процедуры в основной программе просто указывается ее имя со списком параметров.

Пример.

```
{Процедура, вычисляющая сумму и произведение 3-х чисел}
Var a,b,c,d,e: integer; // глобальные параметры
procedure sumpr(x,y,z: integer; var v,w: integer);
  begin                  // x,y,z - входные параметры
   v:=x+y+z; w:=x*y*z    // v,w - выходные параметры
  end;
begin
 readln(a,b,c);
 sumpr(a,b,c,d,e);      // вызов процедуры
 writeln('sum=',d,' mult=',e);
end.
```

ТИПЫ ДАННЫХ

Тип определяет множество допустимых значений переменной.

В Паскале имеются следующие типы данных: простые типы, структурированные типы, строки и указатели.

ПРОСТЫЕ ТИПЫ

К простым относятся порядковые, вещественные (real) типы и тип дата-время.

Порядковые типы данных

Порядковые типы: `integer`, `char`, `boolean`, а также перечисляемый тип и тип-диапазон.

К ним применимы следующие процедуры и функции:

Процедура `dec(x,y)` - уменьшает значение числа `x` на `y`.

Пример: `x:=5; y:='d'; dec(x,2); dec(y,3);` рез.: `x=3 y='a'`.

Процедура `inc(x,y)` - увеличивает значение числа `x` на `y`.

Процедуры `dec` и `inc` могут иметь только один параметр: `dec(x)`, `inc(x)`. В этом случае значение `x` уменьшается (увеличивается) на единицу. Часто вместо `i:=i+1;` используют `inc(i);`

Функция `pred(x)` - возвращает значение, предшествующее `x`.

Пример: `x:='b'; y:=pred(x);` рез.: `y='a'`

Функция `succ(x)` - возвращает значение, следующее за `x`.

Оператор TYPE

В Паскале существует ключевое слово `TYPE`, которое позволяет использовать существующие в Паскале типы данных, а также описывать свои.

Задание типа осуществляется перед описанием переменных:

```
type <тип>=<допустимые значения>;
var  <переменная>:<тип>;
```

Перечисляемый тип данных

Определяется набором идентификаторов, разделенных запятой и указываемых в круглых скобках.

Раздел описаний:

```
type week=(Monday,Tuesday,Wednesday,Thursday,Friday,
Saturday,Sunday);
var  day: week;
begin
  day:=Sunday;
  if day < Saturday  then  writeln('Week-day');
  if day > Friday  then  writeln('Week end');
end.
```

Значениями переменной `day` могут быть только идентификаторы, перечисленные в типе `week`.

Известный нам логический тип является частным случаем перечисляемого типа:

```
type boolean=(False,True);
```

Перечисляемый тип можно использовать в качестве индекса массива.

Переменные перечисляемого типа не могут вводиться оператором `read` и выводиться оператором `write`.

Тип-диапазон

Тип-диапазон есть подмножество своего базового типа. Задается min и max значениями, например:

```
type digit= '0'..'9'; //базовый тип char
```

Тип-диапазон можно не описывать в разделе `type`, а сразу указывать при описании переменной:

```
var month: 1..12; //базовый тип integer
```

Тип-диапазон имеет все свойства базового типа, но с ограничениями по min и max значениям.

Пример.

```
type WeekEnd=Saturday..Sunday; //базовый тип week
var: w: WeekEnd;
begin
  w:=Saturday;
end.
```

Код переменной типа диапазон равен её порядковому номеру в базовом типе (нумерация начинается с нуля). Так `Ord(w)` вернет значение 5.

Тип дата-время

`TDateTime` – тип дата-время, предназначен для хранения даты и времени. Вещественное число. В целой части храниться дата, в дробной – время. Дата - количество суток, прошедших с 30.12.1899 г., а время – как часть суток, прошедших с 0 часов. Над данными типа `TDateTime` определены те же операции, что и над вещественными числами.

Функции для типа дата-время:

`Date: TDateTime;` - возвращает текущую дату;

`Now:  TDateTime;` - возвращает текущую дату и время;

`DateToStr(D: TDateTime): String;` – преобразует дату в строку символов в формате короткой даты, который устанавливается в Windows (обычно `dd.mm.yyyy`);

`FormatDateTime(F: String; D: TdateTime): String;` – преобразует дату и время в строку символов в формате, указанном в F.

Пример:

```
var D:TDateTime; s1,s2,s3:string;
begin
D:=Now;
s1:=DateToStr(D);
s2:=FormatDateTime('dd.mm.yy hh:mm:ss',D);
s3:=FormatDateTime('dd mmmm yyyy',D);
end;
```

Результат работы программы:
s1=’12.03.2002’
s2=’12.03.02 16:45:07’
s3=’12 Март 2002’

Преобразование типов

В подпрограммы нельзя передавать в явном виде массивы. В этом случае тип массива нужно преобразовывать в простой тип.

Пример.

Неверной будет запись:

```
procedure sum(a:array[1..10] of real);
```

Мы должны переопределить тип переменной `a` в простой тип:

```
type massive=array[1..10] of real;
Var a: massive;  summa: real;
    i: integer;
      procedure sum(b:massive; var s:real);
      begin
       for i:=1 to 10 do s:=s+b[i];
      end;
    begin
       for i:=1 to 10 do readln(a[i]);
       sum(a,summa);
       writeln('summa=',summa:1:2);
    end.
```

СТРУКТУРИРОВАННЫЕ ТИПЫ

Четыре: массивы, записи, множества, файлы.

Записи

Записи позволяют хранить в одной переменной данные, имеющие различный тип.

Структура объявления типа записи:

```
type <запись> = RECORD
  <поля записи>
END;
```

После объявления типа можно описать переменную типа запись:

```
Var <переменная> : <запись>;
```

Пример:

```
type Info = record
  Name:
  string[25];
  Income: real;
  Gr: TdateTime;
end;
var person: Info;
    student: array [1..20] of Info;
```

Поля записи

Для обработки доступна как вся запись, так и отдельные ее поля.

При обращении к отдельным полям указывается имя всей записи и имя отдельного поля, разделенные точкой.

```
person.Gr:=StrToDate('11.05.85');
student[1].Name:='Куролесов';
for i:=1 to 20 do
    student[i].Income:=10000;
writeln(student[1].Income:1:2);
```

Оператор WITH...DO

Применяется при совместной обработке нескольких полей записи.

Идентификатор записи указывается однократно в начале фрагмента программы обработки записи. Это позволяет более компактно представлять переменные записей.

Например, вместо:

```
person.Name:='Одуванчиков';
person.Income:=1000;
```

можно записать:

```
WITH person DO
begin
  Name:='Одуванчиков';
  Income:=1000;
end;
```

ЛЕКЦИЯ 8

Множества

Множество – это набор логически связанных друг с другом объектов одного типа. Элементов в множестве может быть от 0 до 256.

Описание множества:

```
type <тип>=set of <диапазон>;
var <множество>:<тип>;
```

Пример:

```
type digit=set of 0..9;
var s1,s2,s3,s4:digit;
    AlphaBet: set of 'a'..'z'; // можно и так
```

Конкретные значения множественного типа задаются списком элементов множества, заключенных в квадратные скобки:

```
s1:=[1,2,3,4];
s2:=[3..5];
s3:=[]; // пустое множество
```

Операции над множествами

1. Объединение двух множеств, знак "+"

Результат: множество, содержащее элементы первого множества, дополненное недостающими элементами из 2-го множества.

```
s4:=s1+s2; // s4 = [1,2,3,4,5]
```

2. Разность двух множеств, знак "-"

Результат: множество, содержащее элементы первого множества, не принадлежащие 2-му.

```
s4:=s1-s2; // s4 = [1,2]
```

3. Пересечение двух множеств, знак "*"

Результат: множество, содержащее элементы, общие для 2-х множеств.

```
s4:=s1*s2; // s4 = [3,4]
```

4. Операции отношения: =, <> , >= ,<=

Два множества являются эквивалентными, когда все их элементы одинаковы, при этом порядок следования элементов в множестве несущественен.

```
  s2=[5,4,3];  - TRUE      s2=[1,3,5];  - FALSE
  s2 >= [];    - TRUE      s2 >= [4,5]; - TRUE
  s2 >= [1,4]; - FALSE //элементы множеств не совпадают
```

5. Оператор `in` – проверка элемента на принадлежность множеству.

```
  4 in s2;     - TRUE
  3*3 in s2;   - FALSE
```

Оператор `in` можно использовать для более компактной записи некоторых логических выражений:

```
in [множество]
```

Например, выражение $3 \le x \le 5$ вместо

```
If (3<=x) and (x<=5) then …
```

можно записать как

```
if x in [3..5] then …
```

Или ещё вариант: `if c in ['+','-','/','*'] then …`

Файлы

Файл – именованная область внешней памяти компьютера (диска, дискеты). Для доступа к конкретному файлу в программе используется переменная файлового типа.

Переменные файлового типа можно объявить тремя способами:

`<имя>: File of <тип>;` - типизированный файл, где `<тип>` – любой тип данных

`<имя>: TextFile;` - текстовый файл

`<имя>: File;` - нетипизированный файл

Процедуры обработки файлов:

1. `AssignFile(<переменная файлового типа >,<имя файла>);`

`AssignFile` связывает переменную файлового типа с именем файла.

Например

```
Var f: TextFile;   // переменная файлового типа
begin
AssignFile(f,'d:\kc\512a\1.txt');
```

`AssignFile(f,'1.txt');` - файл находится в текущей папке.

2. `Rewrite(f)` - создает новый файл или стирает содержимое в уже существующем файле.

3. `Reset(f)` - открывает для чтения файл, если он существует. Чтобы проверить, существует ли файл, необходимо использовать функцию `FileExists`. Пример:

```
if FileExists('1.txt') then - файл существует
                       else - файл не существует
```

4. `Read(f,…)` и `Write(f,…)` – чтение данных из файла и запись данных в файл. `Readln` и `Writeln` – только для текстовых файлов.

5. `Append(f)` – инициирует запись в существующий файл типа `TextFile`.

6. `EOF(f)` – конец файла. Логическая функция. `TRUE` - если достигнут конец файла.

7. `CloseFile(f)` - закрывает файл.

Пример:

```
Var f: TextFile;
    s: string;
    a:real;
begin
 AssignFile(f,'1.txt');
 if not FileExists('1.txt')
    then rewrite(f)
    else begin reset(f); append(f); end;
 readln(s,a);
// добавляем строку s и переменную a в файл 1.txt :
 writeln(f,s,' ',a:1:2);
end.
```

УКАЗАТЕЛИ

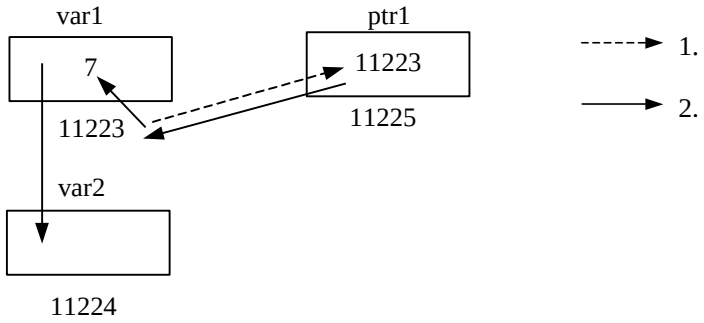
Указатели относятся к т.н. динамически размещаемым переменным. Оперативная память компьютера представляет собой совокупность ячеек для хранения информации – байтов. Каждый байт имеет собственный номер или адрес. Указатели содержат адреса байтов.

Если за указателем стоит значок ^, то имеется ввиду *значение*, на которое указывает указатель, а если значка ^ нет, то имеется ввиду *адрес*, по которому размещены данные. Для получения адреса переменной используется функция `addr`.

Имеется 2 типа указателей:

1. Типизированные. Указывают на переменные определенного типа. Для объявления перед типом ставится значок ^.

```
Var ptr1: ^integer; // указатель на переменные типа integer
    var1, var2: integer;
begin
  var1:=7;
var2:=var1; // присваиваем var2 значение var1 напрямую
// или с помощью указателей:
ptr1:=addr(var1);//1. ptr1 присваивается адрес var1 (11223)
var2:=ptr1^;//2. переменной var2 присваивается значение
            //   переменной var1 на которую указывает ptr1
end.
```



Две следующие записи идентичны:

```
var1:=10;
```

или

```
ptr1^:=10;
```

2. Нетипизированные, т.е. не указывающие на определенный тип. Имеют свой тип `Pointer`.

```
Var ptr2: Pointer; //указатель на переменные любого типа
```

МОДУЛИ (UNIT)

Модуль (`UNIT`) – набор процедур и функций, хранящихся в отдельном файле. Файлы, содержащие модули, имеют расширение `pas`.

Структура:

```
UNIT <имя модуля>;
INTERFACE //интерфейсная часть
< Подключение модулей, объявление меток и переменных,
  заголовки функций >
IMPLEMENTATION //раздел реализации
< Подключение модулей, объявление меток и переменных,
  тела функций >
END.
```

Подключение модуля к основной программе и к другим модулям осуществляется с помощью оператора `USES`, который помещается в разделе описаний перед описанием типов, переменных, констант и меток:

```
USES <имя модуля>;
```

Для создания в проекте нового модуля необходимо выполнить `File=>New=>Unit`. В Окне кода программы появиться шаблон кода модуля, которому присваивается по умолчанию имя `Unit1`:

```
unit Unit1;
interface
implementation
end.
```

Пример модуля, содержащего процедуру округления дробной части числа с указанной точностью:

```
unit Unit1;
interface
 procedure okrugl(s:integer;var a:real);
implementation
 procedure okrugl(s:integer;var a:real);
  //Округление дробной части числа a с точностью s
  begin
  a:=round(a*s)/s;
  end;
end.
```

Программа, использующая модуль Unit1:

```
uses Unit1;      // подключение модуля Unit1
var b:real;
begin
readln(b);
okrugl(100,b);  // вызов процедуры из модуля Unit1
writeln(b:1:6);
readln;
end.
```