Конспект лекций по дисциплине
ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЕ

Предназначен для бакалавров, обучающихся по направлению подготовки 13.03.02 «Электроэнергетика и электротехника» профилю подготовки Электрооборудование автомобилей и тракторов очной формы обучения

1. Описание языка Pascal
2. Среда разработчика Turbo Pascal v 7.0
    1. Запуск программы
    2. Окно среды разработчика
    3. Основные команды и горячие клавиши
3. Синтаксис и семантика
    1. Алфавит языка
    2. Элементарные конструкции
4. Типы данных
    1. Понятие типа данных
    2. Простые типы данных
5. Переменные и константы
6. Выражения
    1. Математические операции
    2. Логические операции
    3. Операции отношения
    4. Приоритет опраций
    5. Основные математические функции
7. Операторы ввода и вывода
8. Структура программы
9. Организация ветвлений
    1. Оператор условного перехода (if .. then .. else)
    2. Оператор выбора (case .. of .. end)
    3. Оператор безусловного перехода (goto)
10. Циклы
    1. Арифметические
    2. Итерационные с предусловием
    3. Итерационные с постусловием
    4. Операторы завершения цикла
11. Процедуры и функции
    1. Описание и вызов процедур и функций
    2. Передача параметров
    3. Локальные и глобальные идентификаторы
12. Структурированные типы данных
    1. Перечисляемый
    2. Интервальный
    3. Массивы
    4. Строки
        1. Строковый тип данных
        2. Процедуры и функции для работы со строками
    5. Множества

1.Описание языка Turbo Pascal

Язык Паскаль, названный в честь французского математика и философа Блеза Паскаля (1623-1662), был создан как учебный язык программирования в 1968-71 годах швейцарским ученым Никлаусом Виртом на кафедре информатики Стэнфордского университета (Цюрих). В настоящее время это язык имеет более широкую сферу применения, чем предусматривалось при его создании. Свое признание Паскаль получил с появлением пакета Турбо Паскаль (Turbo Pascal). Этот язык отличается простотой понимания, стройностью и структурностью алгоритмов, быстротой компилятора и удобными средствами создания и отладки программ.

Достоинствами языка Паскаль являются:

1. Простой синтаксис языка. Небольшое число базовых понятий. Программы на Паскале достаточно легко читаемы.
2. Достаточно низкие аппаратные и системные требования как самого компилятора, так и программ, написанных на Паскале.
3. Универсальность языка. Язык Паскаль применим для решения практически всех задач программирования.
4. Поддержка структурного програмирования, программирования "сверху-вниз", а также объектно-ориентированного программирования.

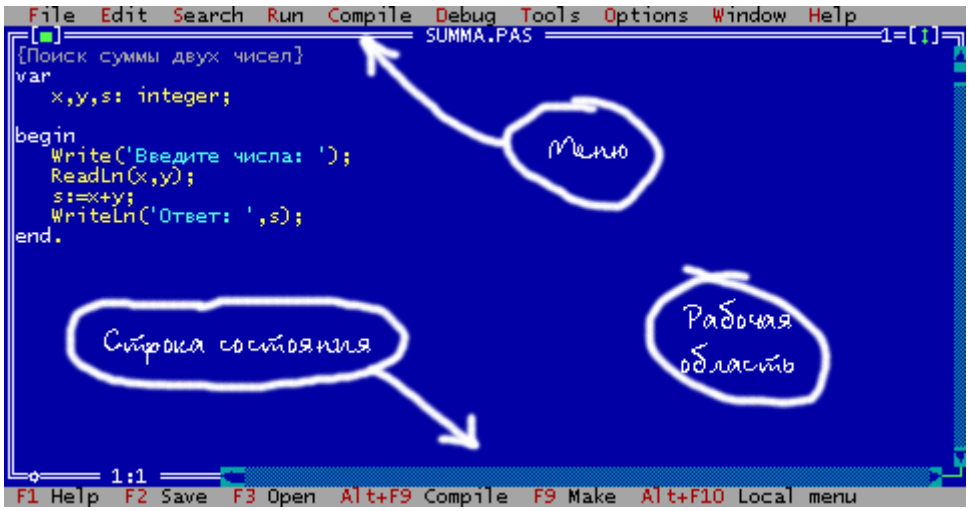
В настоящем пособии рассматривается Tirbo Pascal v7.0. Данная версия разработана фирмой Borland и является последней в линейке компиляторов Pascal для DOS. Дальнейшее развитие Паскаль получил в Delphi - системе разработки программ для Windows.

Среда разработчика Turbo Pascal 7.0

Базовыми компонентами система программирования Турбо Паскаль являются компилятор языка Паскаль, средства создания и редактирования исходных текстов программ и средства их отладки (поиска ошибок). Все эти компоненты объединены в единую **интегрированную среду разработчика**, с которой как раз и работает программист, создавая свои программы.

Окно среды разработчика

Основной экран интегрированной среды разработчика Turbo Pascal 7.0 выглядит следующим образом:



По функциональному назначению выделяется три области экрана:

- Строка меню
- Рабочая область
- Строка состояния

Строка меню активизируется нажатием клавиши F10. В меню содержатся следующие разделы:

- **File**. Позволяет выполнять все основные действия с файлами (создание, открытие, сохранение ..)
- **Edit**. Позволяет выполнять все основные операции редактирования текста (копирование, вставка, удаление фрагментов, отмена последних изменений ..)
- **Search**. Позволяет осуществлять поиск и замену фрагментов текста.
- **Run**. Позволяет запускать программу, в том числе в пошаговом режиме.
- **Compile**. Позволяет осуществлять компиляцию программы.
- **Debug**. Содержит команды, облегчающие процесс поиска ошибок в программе.
- **Tools**. Содержит некоторые дополнительные средства Турбо Паскаль.
- **Options**. Позволяет установить необходимые для работы параметры компилятора и среды разработчика.
- **Window**. Позволяет выполнять все основные операции с окнами (открывать, закрывать, перемещать, изменять размер).
- **Help**. Позволяет получить имеющуюся в системе справочную информацию.

Все пункты меню доступны через горячие клавиши. Для этого надо нажать клавишу Alt и ту букву, которая выделена красной в названии пункта меню. Меню также позволяет работать с мышью.

В рабочей области имеется возможность открывать различные окна программы - окна редактируемого текста, окна помощи, отладки и настройки. В вышеприведенном примере открыто только одно окно - окно текста программы. В заголовке окна написано имя файла - исходного текста программы.

Строка состояния демонстрирует некоторые доступные и важные в данный момент операции и соответствующие им комбинации клавиш.

Основные команды и горячие клавиши

Ниже приведены основные команды среды раработчика Турбо Паскаль и соответствующие им горячие клавиши. Более полный перечень горячих клавиш вы можете найти в приложении.

- **Ctrl+F9** - запуск программы
- **Alt+F5** - просмотр пользовательского экрана
- **F2** - сохранение программы
- **F3** - открытие сохраненной программы
- **Alt+F3** - закрытие активного окна
- **Alt+X** - выход из Турбо Паскаль
- **F1** - контекстная помощь
- **Ctrl+F1** - справка об операторе, на котором установлен курсор
- **Alt+Backspace** - отмена последнео изменения
- **Ctrl+Y** - удаление строки
- **Shift+стрелки** - выделение блока текста
- **Ctrl+Insert** - копирование выделенного блока в буфер
- **Shift+Insert** - вставка из буфера

2.Синтаксис и семантика

Описание каждого элемента языка задается его СИНТАКСИСОМ и СЕМАНТИКОЙ. Синтаксические определения устанавливают правила построения элементов языка. Семантика определяет смысл и правила использования тех элементов языка, для которых были даны синтаксические определения.

Алфавит языка

Алфавит - это совокупность допустимых в языке символов. Алфавит Турбо Паскаль включает следующий набор основных символов:

- строчные и прописные латинские буквы:
- A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
- a b c d e f g h i j k l m n o p q r s t u v w x y z
- пробел
- подчеркивание: _
- арабские цифры: 0 1 2 3 4 5 6 7 8 9
- знаки операций: + - * / = <> < > <= >= := @
- ограничители: . , ' ( ) [ ] (. .) { } (* *) .. : ;
- спецификаторы:^ # $
- служебные (зарезервированные) слова:
- ABSOLUTE EXPORTS LIBRARY SET
- ASSEMBLER EXTERNAL MOD SHL
- AND FAR NAME SHR
- ARRAY FILE NIL STRING
- ASM FOR NEAR THEN
- ASSEMBLER FORWARD NOT TO
- BEGIN FUNCTION OBJECT TYPE

- CASE GOTO OF UNIT
- CONST IF OR UNTIL
- CONSTRUCTOR IMPLEMENTATION PACKED USES
- DESTRUCTOR IN PRIVATE VAR
- DIV INDEX PROCEDURE VIRTUAL
- DO INHERITED PROGRAM WHILE
- DOWNTO INLINE PUBLIC WITH
- ELSE INTERFACE RECORD XOR
- END INTERRUPT REPEAT
- EXPORT LABEL RESIDENT

Элементарные конструкции

Элементарные конструкции языка Паскаль включают в себя имена, числа и строки.

Имена (идентификаторы) называют элементы языка - константы, метки, типы, переменные, процедуры, функции, модули, объекты. Идентификатор в Турбо Паскале может включать в себя:
- буквы латинского алфавита,
- цифры
- символ подчеркивания.

Строчные и прописные буквы не различаются (например, NAME, Name и name будет означать одно и то же). Цифра не может стоять на первом месте в идентификаторе, а символ подчеркивания может находиться в любой позиции (например, name1 и name2item являются допустимыми идентификаторами, а 5name - нет; _name, name_, name_item - тоже допустимые названия). Длина идентификатора может быть любой, но значимыми являются только первые 63 символа. В качестве имен не допускается использовать служебные слова.

Для отделения друг от друга идентификаторов, чисел, зарезервированных слов используются разделители. В качестве них можно использовать:
- пробел и табуляцию;
- перевод строки;
- комментарий.

В любом месте программы, где можно поместить один разделитель, их можно поместить любое количество и в любом сочетании, что позволяет наглядно представить структуру программы. Комментарии заключаются либо в скобки { ... }, либо в скобки вида (* ... *) и могут занимать любое число строк.

Числа в языке Паскаль обычно записываются в десятичной системе счисления. Они могут быть целыми и действительными. Положительный знак числа может быть опущен. Целые числа записываются в форме без десятичной точки, например:

217 -45 8954 +483

Действительные числа записываются в форме с десятичной точкой:

28.6 0.65 -0.018 4.0

Возможна также запись с использованием десятичного порядка, который изображается буквой E:

5E12 -1.72E9 73.1E-16

В "переводе" такую запись следует понимать соответственно как:

$5x10^{12}$ $-1.72x10^{9}$ $73.1x10^{-16}$

Паскаль допускает запись целых чисел и фрагментов действительных чисел в форме с порядком в шестнадцатиричной системе счисления:

$7F $40 $ABC0

Строки в языке Паскаль - это последовательность символов, записанная между апострофами. Если в строке в качестве содержательного символа необходимо употребить сам апостроф, то следует записать два апострофа. Примеры строк:

'СТРОКА' 'STRING' 'ПРОГРАММА' 'АД"ЮТАНТ'

3.Типы данных

Понятие типа данных в Турбо Паскаль

Для обработки ЭВМ данные представляются в виде величин и их совокупностей. С понятием величины связаны такая важная характеристика, как ее тип.

Тип определяет:

- возможные значения переменных, констант, функций, выражений, принадлежащих к данному типу;
- внутреннюю форму представления данных в ЭВМ;
- операции и функции, которые могут выполняться над величинами, принадлежащими к данному типу.

В языке Паскаль тип величины задают заранее. Все переменные, используемые в программе, должны быть объявлены в разделе описания с указанием их типа. Обязательное описание типа приводит к избыточности в тексте программ, но такая избыточность является важным вспомогательным средством разработки программ и рассматривается как необходимое свойство современных алгоритмических языков высокого уровня.

Иерархия типов в языке Паскаль такая:

- Простые
  - Порядковые
    - Целые
    - Логические
    - Символьные
    - Перечисляемые
    - Интервальные
  - Вещественные

- Структуированные
  - Массивы
  - Строки
  - Множества
  - Записи
  - Файлы
- Указатели

Простые типы данных

В таблице приведены простые типы данных Турбо Паскаль, объем памяти, необходимый для хранения одной переменной указанного типа, множество допустимых значений и применимые операции.

Идентификатор	Длина (байт)	Диапазон значений	Операции
Целые типы			
integer	2	-32768..32767	+, -, /, *, Div, Mod, >=, <=, =, <>, <, >
byte	1	0..255	+, -, /, *, Div, Mod, >=, <=, =, <>, <, >
word	2	0..65535	+, -, /, *, Div, Mod, >=, <=, =, <>, <, >
shortint	1	-128..127	+, -, /, *, Div, Mod, >=, <=, =, <>, <, >
longint	4	-2147483648..2147483647	+, -, /, *, Div, Mod, >=, <=, =, <>, <, >
Вещественные типы			
real	6	$2,9x10^{-39}$ - $1,7x10^{38}$	+, -, /, *, >=, <=, =, <>, <, >
single	4	$1,5x10^{-45}$ - $3,4x10^{38}$	+, -, /, *, >=, <=, =, <>, <, >
double	8	$5x10^{-324}$ - $1,7x10^{308}$	+, -, /, *, >=, <=, =, <>, <, >
extended	10	$3,4x10^{-4932}$ - $1,1x10^{4932}$	+, -, /, *, >=, <=, =, <>, <, >
Логический тип			
boolean	1	true, false	Not, And, Or, Xor, >=, <=, =, <>, <, >
Символьный тип			
char	1	все символы кода ASCII	+, >=, <=, =, <>, <, >

Перечисляемый и интервальный тип относятся к типам, определяемым пользователем и будут рассмотрены нами позже.

Подробности..

Порядковые типы, выделяемые из группы простых типов, характеризуются следующими свойствами:

- все возможные значения порядкового типа представляют собой ограниченное упорядоченное множество;
- к любому порядковому типу может быть применена стандартная функция Ord, которая в качестве результата возвращает порядковый номер конкретного значения в данном типе;
- к любому порядковому типу могут быть применены стандартные функции Pred и Succ, которые возвращают предыдущее и последующее значения соответственно;
- к любому порядковому типу могут быть применены стандартные функции Low и High, которые возвращают наименьшее и наибольшее значения величин данного типа.

В языке Паскаль введены понятия эквивалентности и совместимости типов. Два типа T1 и T2 являются эквивалентными (идентичными), если выполняется одно из двух условий:

- T1 и T2 представляют собой одно и то же имя типа;
- тип T2 описан с использованием типа T1 с помощью равенства или последовательности равенств. Например:
- type
- T1 = Integer;
- T2 = T1;
- T3 = T2;

Менее строгие ограничения накладываются на совместимость типов. Так, типы являются совместимыми, если:

- они эквивалентны;
- являются оба либо целыми, либо действительными;
- один тип - интервальный, другой - его базовый;
- оба интервальные с общим базовым;
- один тип - строковый, другой - символьный.

В Турбо Паскаль ограничения на совместимость типов можно обойти с помощью приведения типов. Приведение типов позволяет рассматривать одну и ту же величину в памяти ЭВМ как принадлежащую разным типам. Для этого используется конструкция

Имя_Типа(переменная или значение)

Напрмер, Integer('Z') представляет собой значение кода символа 'Z' в двухбайтном представлении целого числа, а Byte(534) даст значение 22, поскольку целое число 534 имеет тип Word и занимает два байта, а тип Byte занимает один байт, и в процессе приведения старший байт будет отброшен.

4.Переменные и константы

Переменной называют элемент программы, который предназначен для хранения, коррекции и передачи данных внутри программы. Все переменные программы в Турбо Паскаль должны быть объявлены в разделе описания переменных (см. далее).

Наряду с переменными в пограммах используются и **константы**. Константа - это идентификатор, обозначающий некоторую неизменную величину определенного типа. Константы, как и переменные, должны объявляться в соответствующем разделе программы.

В Турбо Паскаль применяется несколько стандартных видов констант:

- **Целочисленные** константы. Могут быть определены посредством чисел, записанных в десятичном или шестнадцатиричном формате данных. Это число не должно содержать десятичной точки.
- **Вещественные** константы. Могут быть определены числами, записанными в десятичном формате данных с использованием десятичной точки.
- **Символьные** константы. Могут быть определены посредством некоторого символа (заключенного в апострофы).
- **Строковые** константы. Могут быть определены последовательностью произвольных символов (заключенных в апострофы).
- **Типизированные** константы. Представляют собой инициализированные переменные, которые могут использоваться в программах наравне с обычными переменными. Каждой типизированной константе ставится в соответствие имя, тип и начальное значение. Например:
- year: integer = 2001;
- symb: char = '?';
- money: real = 57.23;

5.Выражения

Выражение задает правило вычисления некоторого значения. Выражение состоит из констант, переменных, указателей функций, знаков операций и скобок.

Математические операции

В таблице приведены основные математические операции Турбо Паскаль.

Символ операции	Название операции	Пример
*	умножение	2*3 (результат: 6)
/	деление	30/2 (результат: 1.5E+01)
+	сложение	2+3 (результат: 5)
-	вычитание	5-3 (результат: 2)
div	целочисленное деление	5 div 2 (результат: 2)
mod	остаток от деления	5 mod 2 (результат: 1)

Логические операции

Над логическими аргументами в Турбо Паскаль определены следующие операции:

- NOT - логическое отрицание ("НЕ")
- AND - логическое умножение ("И")
- OR - логическое сложение ("ИЛИ")
- XOR - логическое "Исключающее ИЛИ"

Результаты выполнения этих операций над переменными A и B логического типа приведены в таблице истинности.

A	B	not A	A and B	A or B	A xor B
true	true	false	true	true	false
true	false		false	true	true
false	true	true	false	true	true
false	false		false	false	false

Операции отношения

К операциям отношения в Турбо Паскаль относятся такие операции, как:

- > - больше
- < - меньше
- = - равно
- <> - не равно
- >= - больше или равно
- <= - меньше или равно

В операциях отношения могут принимать участие не только числа, но и символы, строки, множества и указатели.

Приоритет операций

Порядок вычисления выражения определяется старшинством (приоритетом) содержащихся в нем операций. В языке Паскаль принят следующий приоритет операций:

- унарная операция not, унарный минус -, взятие адреса @
- операции типа умножения: * / div mod and
- операции типа сложения: + - or xor
- операции отношения: = <> < > <= >= in

Порядок выполнения операций переопределить можно с помощью скобок. Например 2*5+10 равно 20, но 2*(5+10) равно 30.

Основные математические функции

В этом разделе приведены основные математические функции, встроенные в системную библиотеку Турбо Паскаль.

Abs(X)
Возвращает абсолютное значение числа X.
Cos(X), Sin(X)
Возвращает косинус (синус) числа X, где X - угол в радианах.

Функций тангенс и котангенс в Турбо Паскале нет. Для их вычисления используйте выражение sin(x)/cos(x) (или cos(x)/sin(x) для котангенса).

ArcTan(X)
Возвращает арктангенс числа X.
Exp(X)

Возвращает число, равное e в степени X.
Ln(x)
Возвращает число, равное натуральному логарифму от числа X.
Pi
Число Пи.
Sqr(X)
Возвращает число, равное квадрату числа X.

Функции возведения в произвольную степень в Турбо Паскале нет. Используйте многократное умножение для возведения в целочисленную степень, либо функции Exp и Ln для возведения в вещественную степень.

Sqrt(X)
Возвращает число, равное квадратному корню из числа X.
Trunc(X)
Возвращает число, равное целой части числа X. (Происходит отбрасывание дробной части числа X. Результат выполнения имеет тип Longint).
Frac(X)
Возвращает число, равное дробной части числа X.
Int(X)
Возвращает число, равное целой части числа X. Результат выполнения функции - real.
Round(X)
Функция округляет число X. Возвращаемое значение имеет тип Longint.
Random(X)
Возвращает случайное целое число в диапазоне 0..X. Если аргумент опущен (Random), то возвращается случайное вещественное число от 0 до 1.

Перед использованием random в программах рекомендуется сначала инициализировать генератор псевдослучайных чисел процедурой Randomize. В противном случае при каждом запуске программы будет генерироваться одна и та же последовательность случайных чисел.

Пример. Вывод на экран 5 случайных чисел в диапазоне -10..10.

```
var i: integer;
begin
  randomize;
  for i:=1 to 5 do writeln(random(21)-10);
end.
```

Inc(X,Y)
Увеличивает значение числа X на Y. Если число Y не указано, то увеличение происходит на 1.
Dec(X,Y)
Уменьшает значение числа X на Y. Если число Y не указано, то уменьшение происходит на 1.

6.Структура программы на Турбо Паскаль

Программа на языке Паскаль состоит из заголовка, разделов описаний и раздела операторов. Заголовок программы содержит имя программы, например:

Program PRIM;

Описания могут включать в себя:

- раздел подключаемых библиотек (модулей);
- раздел описания меток;
- раздел описания констант;
- раздел описания типов;
- раздел описания переменных;
- раздел описания процедур и функций.

Раздел описания модулей определяется служебным словом USES и содержит имена подключаемых модулей (библиотек) как входящих в состав системы Turbo Pascal, так и написанных пользователем. Раздел описания модулей должен быть первым среди разделов описаний. Имена модулей отделяются друг от друга запятыми:

```
uses CRT, Graph;
```

Любой оператор в программе может быть помечен меткой. Имя метки задается по правилам образования идентификаторов Турбо Паскаль. В качестве метки также могут использоваться произвольные целые числа без знака, содержащие не более четырех цифр. Метка ставится перед оператором и отделяется от него двоеточием. Все метки, используемые в программе, должны быть перечислены в **разделе описания меток**, например:

```
label 3, 471, 29, Quit;
```

Описание констант позволяет использовать имена как синонимы констант, их необходимо определить в **разделе описания констант**:

```
const K= 1024; MAX= 16384;
```

В **разделе описания переменных** необходимо указать все переменные, используемые в программе, и определить их тип:

```
var P,Q,R: Integer;
   A,B:   Char;
   F1,F2: Boolean;
```

Описание типов, процедур и функций будет рассмотрено ниже. Отдельные разделы описаний могут отсутствовать, но следует помнить, что в Паскаль - программе должны быть обязательно описаны все компоненты программы.

Раздел операторов представляет собой составной оператор, который содержит между служебными словами

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

последовательность операторов. Операторы отделяются друг от друга символом ;. Текст программы заканчивается символом точка.

Кроме описаний и операторов Паскаль - программа может содержать комментарии, которые представляют собой произвольную последовательность символов,

расположенную между открывающей скобкой комментариев { и закрывающей скобкой комментариев }.

Пример 1

```
program Primer; {вычисление суммы двух чисел}
var
   x,y,s: integer;
begin
  WriteLn('Введите через пробел два числа ');
  ReadLn(x,y);
  s := x + y;
  WriteLn('Сумма чисел равна ',s);
end.
```

Данная программа запрашивает с клавиатуры два числа, находит их сумму и выводит ответ. Теперь сделаем так, чтобы программа сначала очищала экран, выполняла свои действия, а в конце работы позволяла пользователю посмотреть результат, ожидая его нажатия клавиши.

Пример 2

```
program Primer; {вычисление суммы двух чисел}
uses Crt; {подключение модуля, необходимого для процедур
          очистки экрана и задержки}
var
  x,y,s: integer;
begin
  ClrScr; {очистка экрана}
  WriteLn('Введите через пробел два числа ');
  ReadLn(x,y);
  s := x + y;
  WriteLn('Сумма чисел равна ',s);
  ReadKey; {ожидание нажатия клавиши}
end.
```

Подробности..

Текст Паскаль - программы может содержать ключи компиляции, которые позволяют управлять режимом компиляции. Синтаксически ключи компиляции записываются как комментарии. Ключ компиляции содержит символ $ и букву-ключ с последующим знаком + (включить режим) или - (выключить режим). Например:

{$E+} - эмулировать математический сопроцессор;
{$F+} - формировать дальний тип вызова процедур и функций;
{$N+} - использовать математический сопроцессор;
{$R+} - проверять выход за границы диапазонов.

Некоторые ключи компиляции могут содержать параметр, например:

{$I имя файла} - включить в текст компилируемой программы названный файл.

7.Операторы ввода и вывода

В данном разделе рассмотрим организацию ввода и вывода данных с терминального устройства. Терминальное устройство - это устройство, с которым работает пользователь, обычно это клавиатура и экран (дисплей).

Ввод данных

Для ввода исходных данных чаще всего используется процедура ReadLn:

ReadLn(A1,A2,...AK);

Процедура производит чтение К значений исходных данных и присваивает эти значения переменным А1, А2, ..., АК.

При вводе исходных данных происходит преобразование из внешней формы представления во внутреннюю, определяемую типом переменных. Переменные, образующие список ввода, могут принадлежать либо к целому, либо к действительному, либо к символьному типам. Чтение исходных данных логического типа в языке Паскаль недопустимо.

Значения исходных данных могут отделяться друг от друга пробелами и нажатием клавиш табуляции и Enter.

Не допускается разделение вводимых чисел запятыми!

Вывод данных

Для вывода результатов работы программы на экран используются процедуры:

Write(A1,A2,...AK);
WriteLn(A1,A2,...AK);

Первый из этих операторов производит вывод значений переменных А1, А2,...,АК в строку экрана. Второй оператор, в отличие от первого, не только производит вывод данных на экран, но и производит переход к началу следующей экранной строки. Если процедура writeln используется без параметров, то она просто производит пропуск строки и переход к началу следующей строки.

Переменные, составляющие список вывода, могут относиться к целому, действительному, символьному или булевскому типам. В качестве элемента списка вывода кроме имен переменных могут использоваться выражения и строки.

Форма представления значений в поле вывода соответствует типу переменных и выражений: величины целого типа выводятся как целые десятичные числа, действительного типа - как действительные десятичные числа с десятичным порядком, символьного типа и строки - в виде символов, логического типа - в виде логических констант TRUE и FALSE.

Оператор вывода позволяет задать ширину поля вывода для каждого элемента списка вывода. В этом случае элемент списка вывода имеет вид А:К, где А - выражение или строка, К - выражение либо константа целого типа. Если выводимое значение занимает в

поле вывода меньше позиций, чем К, то перед этим значением располагаются пробелы. Если выводимое значение не помещается в ширину поля К, то для этого значения будет отведено необходимое количество позиций.

Для величин действительного типа элемент списка вывода может иметь вид А:К:М, где А - переменная или выражение действительного типа, К - ширина поля вывода, М - число цифр дробной части выводимого значения. К и М - выражения или константы целого типа. В этом случае действительные значения выводятся в форме десятичного числа с фиксированной точкой.

Пример записи операторов вывода:

```
var rA, rB: Real;
   iP,iQ:Integer;
   bR, bS: Boolean;
   chT, chV, chU, chW: Char;
begin
  . . .
  WriteLn(rA, rB:10:2);
  WriteLn(iP, iQ:8);
  WriteLn(bR, bS:8);
  WriteLn(chT, chV, chU, chW);
end.
```

8.Организация ветвлений в программе.

В языке Паскаль используется два оператора для реализации условных переходов - IF и CASE, а также оператор безусловного перехода GOTO. Они позволяют нарушить последовательный порядок выполнения инструкций программы.

Оператор условного перехода

Оператор условного перехода в Турбо Паскаль имеет вид:

if *условие* then *оператор 1* else *оператор 2*;

условие - это логическое выражение, в зависимости от которого выбирается одна из двух альтернативных ветвей алгоритма. Если значение условия истинно (TRUE), то будет выполняться *оператор 1*, записанный после ключевого слова then. В противном случае будет выполнен *оператор 2*, следующий за словом else, при этом *оператор 1* пропускается. После выполнения указанных операторов программа переходит к выполеннию команды, стоящей непосредственно после оператора if.

Необходимо помнить, что перед ключевым словом else точка с запятой никогда не ставится!

else - часть в операторе if может отсутствовать:

if *условие* then *оператор 1*;

Тогда в случае невыполнения логического условия управление сразу передается оператору, стоящему в программе после конструкции if.

Следует помнить, что синтаксис языка допускает запись только одного оператора после ключевых слов then и else, поэтому группу инструкций обязательно надо объединять в составной оператор (окаймлять операторными скобками begin ... end). В противном случае возникает чаще всего логическая ошибка программы, когда компилятор языка ошибок не выдает, но программа тем не менее работает неправильно.

Примеры.

```
if x > 0 then modul := x else modul := -x;

if k > 0 then WriteLn('k - число положительное');

if min > max then begin
    t := min;
    min := max;
    max := t;
  end;
```

Оператор выбора

Часто возникают ситуации, когда приходится осуществлять выбор одного из нескольких альтернативных путей выполнения программы. Несмотря на то, что такой выбор можно организовать с помощью оператора if .. then, удобнее воспользоваться специальным оператором выбора. Его формат:

case *выражение* of
  вариант : *оператор*;
  ...
  вариант : *оператор*;
end;

или

case *выражение* of
  вариант : *оператор*;
  ...
  вариант : *оператор*;
  else *оператор*
end;

выражение, которое записывается после ключевого слова case, называется ***селектором***, оно может быть любого перечисляемого типа. *вариант* состоит из одной или большего количества констант или диапазонов, разделенных запятыми. Они должны принадлежать к тому же типу, что и селектор, причем недопустимо более одного упоминания *варианта*в записи инструкции case. Из перечисленного множества *операторов* будет выбран только тот, перед которым записан *вариант*, совпадающий со значением селектора. Если такого *варианта* нет, выполняется *оператор*, следующий за словом else (если он есть).

Пример

```
case ch of
   'A'..'Z', 'a'..'z' : WriteLn('Буква');
   '0'..'9'           : WriteLn('Цифра');
   '+', '-', '*', '/' : WriteLn('Оператор');
   else WriteLn('Специальный символ')
end;
```

Оператор безусловного перехода

Помимо операторов условного перехода существует также оператор безусловного перехода goto. Формат:

goto *метка*

Оператор goto переходит при выполнении программы к определенному оператору программы, перед которым находится *метка*. *Метка* должна быть описана в разделе описания меток той программы (процедуры или функции), в которой она используется. Нельзя перейти из одной процедуры или функции в другую.

Необходимо, чтобы в программе существовал оператор, отмеченный указанной меткой. Она записывается перед оператором и отделяется от него двоеточием.

Пример

```
label 1;
begin
  ...
  goto 1;
  ...
  1: WriteLn('Переход к метке 1');
end.
```

Учтите! Само понятие структурного программирования и общепринятый стиль программирования на структурных языках **НЕ ПРИВЕТСТВУЕТ** применение меток и операторов перехода в программах. Это затрудняет понимание программы как автором, так и потребителями, кроме того, применение меток отрицательно сказывается на эффективности генерируемого кода.

9.Циклы

В большинстве задач, встречающихся на практике, необходимо производить многократное выполнение некоторого действия. Такой многократно повторяющийся участок вычислительного процесса называется ***циклом***.

Если заранее известно количество необходимых повторений, то цикл называется ***арифметическим***. Если же количество повторений заранее неизвестно, то говорят об ***итерационном*** цикле.

В итерационных циклах производится проверка некоторого условия, и в зависимости от результата этой проверки происходит либо выход из цикла, либо повторение выполнения тела цикла. Если проверка условия производится перед выполнением блока операторов, то такой итерационный цикл называется циклом ***с предусловием*** (цикл "пока"), а если проверка производится после выполнения тела цикла, то это цикл ***с постусловием*** (цикл "до").

Особенность этих циклов заключается в том, что тело цикла с постусловием всегда выполняется хотя бы один раз, а тело цикла с предусловием может ни разу не выполниться. В зависимости от решаемой задачи необходимо использовать тот или иной вид итерационных циклов.

Арифметические циклы

Синтаксис:

for *переменная* := *значение 1* to *значение 2* do *оператор*

или

for *переменная* := *значение 1* downto *значение 2* do *оператор*

Оператор for вызывает *оператор*, находящийся после слова do, по одному разу для каждого значения в диапазоне от *значения 1* до *значения 2*.

Переменная цикла, начальное и конечное значения должны иметь порядковый тип. Со словом to, значение переменной цикла **увеличивается** на 1 при каждой итерации цикла. Со словом downto, значение переменной цикла **уменьшается** на 1 при каждой итерации цикла. Не следует самостоятельно изменять значение управляющей переменной внутри цикла.

Как и в случае использования оператора условного прехода, следует помнить, что синтаксис языка допускает запись только одного оператора после ключевого слова do, поэтому, если вы хотите в цикле выполнить группу операторов, обязательно надо объединить их в составной оператор (окаймить операторными скобками begin ... end). В противном случае будет сделана логическая ошибка программы.

Пример 1. Квадраты чисел от 2-х до 10-и.

```
for x:=2 to 10 do WriteLn(x*x);
```

Пример 2. Латинский алфавит.

```
for ch:='A' to 'Z' do Writeln(ch);
```

Пример 3. Использование цикла с downto.

```
for i:=10 downto 1 do WriteLn(i);
```

Пример 4. Использование составного оператора.

```
for x:=1 to 10 do begin
    y:=2*x+3;
    WriteLn('f(',x,')=',y);
  end;
```

Итерационные циклы с предусловием

Синтаксис:

while *выражение* do *оператор*

Оператор после do будет выполняться до тех пор, пока логическое *выражение* принимает истинное значение (True). Логическое *выражение* является условием возобновления цикла. Его истинность проверяется каждый раз перед очередным повторением *оператора* цикла, который будет выполняться лишь до тех пор, пока логическое *выражение* истинно. Как только логическое *выражение* принимает значение ложь (False), осуществляется переход к оператору, следующему за while.

Выражение оценивается до выполнения *оператора*, так что если оно с самого начала было ложным (False), то *оператор* не будет выполнен ни разу.

Здесь также следует помнить, что позволяется использовать только один оператор после ключевого слова do. Если необходимо выполнить группу операторов, то стоит использовать составной оператор.

Пример.

```
eps:=0.001;
while x > eps do x:=x/2;
```

Итерационные циклы с постусловием

Синтаксис:

repeat
  оператор;
  оператор;
  ...
  оператор
until *выражение*

Операторы между словами repeat и until повторяются, пока логическое *выражение* является ложным (False). Как только логическое *выражение* становится истинным (True), происходит выход из цикла.

Так как *выражение* оценивается после выполнения *операторов*, то в любом случае *операторы* выполнятся хотя бы один раз.

Пример.

```
repeat
  WriteLn('Введите положительное число');
  ReadLn(x);
until x>0;
```

Операторы завершения цикла

Для всех операторов цикла выход из цикла осуществляется как вследствие естественного окончания оператора цикла, так и с помощью операторов перехода и выхода.

В версии Турбо Паскаль 7.0 определены стандартные процедуры:

Break
Continue
Процедура Break выполняет безусловный выход из цикла. Процедура Continue обеспечивает переход к началу новой итерации цикла.
Заметим, что хотя и существует возможность выхода из цикла с помощью оператора безусловного перехода goto, делать этого не желательно. Во всех случаях можно воспользоваться специально предназначенными для этого процедурами Break и Continue.

10.Процедуры и функции

В языке Паскаль, как и в большинстве языков программирования, предусмотрены средства, позволяющие оформлять вспомогательный алгоритм как подпрограмму. Это бывает необходимо тогда, когда какой-либо подалгоритм неоднократно повторяется в программе или имеется возможность использовать некоторые фрагменты уже разработанных ранее алгоритмов. Кроме того, подпрограммы применяются для разбиения крупных программ на отдельные смысловые части в соответствии с модульным принципом в программировании.

Для использования подалгоритма в качестве подпрограммы ему необходимо присвоить имя и описать алгоритм по правилам языка Паскаль. В дальнейшем, при необходимости вызвать его в программе, делают вызов подпрограммы упоминанием в нужном месте имени соответствующего подалгоритма со списком входных и выходных данных. Такое упоминание приводит к выполнению входящих в подпрограмму операторов, работающих с указанными данными. После выполнения подпрограммы работа продолжается с той команды, которая непосредственно следует за вызовом подпрограммы.

В языке Паскаль имеется два вида подпрограмм - **процедуры** и **функции**.

Процедуры и функции помещаются в раздел описаний программы. Для обмена информацией между процедурами и функциями и другими блоками программы существует механизм **входных** и **выходных параметров**. Входными параметрами называют величины, передающиеся из вызывающего блока в подпрограмму (исходные

данные для подпрограммы), а выходными - передающиеся из подрограммы в вызывающий блок (результаты работы подпрограммы).

Одна и та же подпрограмма может вызываться неоднократно, выполняя одни и те же действия с разными наборами входных данных. Параметры, использующиеся при записи текста подпрограммы в разделе описаний, называют **формальными**, а те, что используются при ее вызове - **фактическими**.

Описание и вызов процедур и функций

Структура описания процедур и функций до некоторой степени похожа на структуру Паскаль-программы: у них также имеются заголовок, раздел описаний и исполняемая часть. Раздел описаний содержит те же подразделы, что и раздел описаний программы: описания констант, типов, меток, процедур, функций, перменных. Исполняемая часть содержит собственно операторы процедур.

Формат описания процедуры имеет вид:

procedure *имя процедуры* (*формальные параметры*);
раздел описаний процедуры
begin

исполняемая часть процедуры

end;

Формат описания функции:

function *имя функции* (*формальные параметры*):*тип результата*;
раздел описаний функции
begin

исполняемая часть функции

end;

Формальные параметры в заголовке процедур и функций записываются в виде:

var *имя праметра*: *имя типа*

и отделяются друг от друга точкой с запятой. Ключевое слово var может отсутствовать (об этом далее). Если параметры однотипны, то их имена можно перечислять через запятую, указывая общее для них имя типа. При описании параметров можно использовать только стандартные имена типов, либо имена типов, определенные с помощью команды type.Список формальных параметров может отсутствовать.

Вызов процедуры производится оператором, имеющим следующий формат:

имя процедуры(*список фактических параметров*);

Список фактических параметров - это их перечисление через запятую. При вызове фактические параметры как бы подставляются вместо формальных, стоящих на тех же местах в заголовке. Таким образом происходит передача входных параметров, затем выполняются операторы исполняемой части процедуры, после чего происходит возврат в вызывающий блок. Передача выходных параметров происходит непосредственно во время работы исполняемой части.

Вызов функции в Турбо Паскаль может производиться аналогичным способом, кроме того имеется возможность осуществить вызов внутри какого-либо выражения. В частности имя функции может стоять в правой части оператора присваивания, в разделе условий оператора if и т.д.

Для передачи в вызывающий блок выходного значения функции в исполняемой части функции перед возвратом в вызывающий блок необходимо поместить следующую команду:

имя функции := *результат*;

При вызове процедур и функций необходимо соблюдать следущие правила:

- количество фактических параметров должно совпадать с количеством формальных;
- соответствующие фактические и формальные параметры должны совпадать по порядку следования и по типу.

Заметим, что имена формальных и фактических параметров могут совпадать. Это не приводит к проблемам, так как соответствующие им переменные все равно будут различны из-за того, что хранятся в разных областях памяти. Кроме того, все формальные параметры являются временными переменными - они создаются в момент вызова подпрограммы и уничтожаются в момент выхода из нее.

Рассмотрим использование процедуры на примере программы поиска максимума из двух целых чисел.

```
var x,y,m,n: integer;

procedure MaxNumber(a,b: integer; var max: integer);
begin
  if a>b then max:=a else max:=b;
end;

begin
  write('Введите x,y ');
  readln(x,y);
  MaxNumber(x,y,m);
  MaxNumber(2,x+y,n);
  writeln('m=',m,'n=',n);
end.
```

Аналогичную задачу, но уже с использованием функций, можно решить так:

```
var x,y,m,n: integer;
```

```
function MaxNumber(a,b: integer): integer;
  var max: integer;
begin
  if a>b then max:=a else max:=b;
  MaxNumber := max;
end;

begin
  write('Введите x,y ');
  readln(x,y);
  m := MaxNumber(x,y);
  n := MaxNumber(2,x+y);
  writeln('m=',m,'n=',n);
end.
```

Передача параметров

В стандарте языка Паскаль передача параметров может производиться двумя способами - по значению и по ссылке. Параметры, передаваемые по значению, называют **параметрами-значениями**, передаваемые по ссылке - **параметрами-переменными**. Последние отличаются тем, что в заголовке процедуры (функции) перед ними ставится служебное слово var.

При первом способе (передача по значению) значения фактических параметров копируются в соответствующие формальные параметры. При изменении этих значений в ходе выполнения процедуры (функции) исходные данные (фактические параметры) измениться не могут. Поэтому таким способом передают данные только из вызывающего блока в подпрограмму (т.е. входные параметры). При этом в качестве фактических параметров можно использовать и константы, и переменные, и выражения. При втором способе (передача по ссылке) все изменения, происходящие в теле процедуры (функции) с формальными параметрами, приводят к немедленным аналогичным изменениям соответствующих им фактических параметров. Изменения происходят с переменными вызывающего блока, поэтому по ссылке передаются выходные параметры. При вызове соответствующие им фактические параметры могут быть только переменными. Выбор способа передачи параметров при создании процедуры (функции) происходит в соответствии со сказанным выше: входные параметры нужно передавать по значению, а выходные - по ссылке. Практически это сводится к расстановке в заголовке процедуры (функции) описателя var при всех параметрах, которые обозначают результат работы подпрограммы. Однако, в связи с тем, что функция возвращает только один результат, в ее заголовке использовать параметры-переменные не рекомендуется.

Локальные и глобальные идентификаторы

Использование процедур и функций в Паскале тесно связано с некоторыми особенностями работы с идентификаторами (именами) в программе. В часности, не все имена всегда доступны для использования. Доступ к идентификатору в конкретный момент времени определяется тем, в каком блоке он описан.

Имена, описанные в заголовке или разделе описаний процедуры или функции называют **локальными** для этого блока. Имена, описанные в блоке, соответствующем всей программе, называют **глобальными**. Следует помнить, что формальные параметры прцедур и функций всегда являются локальными переменными для соответствующих блоков.

Основные правила работы с глобальными и локальными именами можно сформулировать так:

- Локальные имена доступны (считаются известными, "видимыми") только внутри того блока, где они описаны. Сам этот блок, и все другие, вложенные в него, называют **областью видимости** для этих локальных имен.
- Имена, описанные в одном блоке, могут совпадать с именами из других, как содержащих данный блок, так и вложенных в него. Это объясняется тем, что переменные, описанные в разных блоках (даже если они имеют одинаковые имена), хранятся в разных областях оперативной памяти.

Глобальные имена хранятся в области памяти, называемой **сегментом данных** (статическим сегментом) программы. Они создаются на этапе компиляции и действительны на все время работы программы.

В отличие от них, локальные переменные хранятся в специальной области памяти, которая называется **стек**. Они являются временными, так как создаются в момент входа в подпрограмму и уничтожаются при выходе из нее.

Имя, описанное в блоке, "закрывает" совпадающие с ним имена из блоков, содержащие данный. Это означает, что если в двух блоках, один из которых содержится внутри другого, есть переменные с одинаковыми именами, то после входа во вложенный блок работа будет идти с локальной для данного блока переменной. Пременная с тем же имнем, описанная в объемлющем блоке, становится временно недоступной и это продолжается до момента выхода из вложенного блока.

Рекомендуется все имена, которые имеют в подпрограммах чисто внутреннее, вспомогательное назначение, делать локальными. Это предохраняет от изменений глобальные объекты с такими же именами.

11.Структурированные типы данных

Любой из структурированных типов данных характеризуется множественностью образующих этот тип элементов. Переменная или константа структурированного типа всегда имеет несколько компонент. Каждая из этих компонент, в свою очередь, может принадлежать структурированному типу, что позволяет говорить о возможной вложенности типов.

В Турбо Паскале пять структурированных типов:

- массивы;
- строки;
- множества;
- записи;
- файлы;

Однако, прежде чем приступить к их изучению, нам надо рассмотреть еще два типа данных - **перечисляемый** и **интервальный**, которые относятся к порядковым типам, нами ранее не рассматривались, но потребуются при изучении нового материала.

Перечисляемый тип данных

Перечисляемый тип представляет собой ограниченную упорядоченную последовательность скалярных констант, составляющих данный тип. Значение каждой константы задается ее именем. Имена отдельных констант отделяются друг от друга запятыми, а вся совокупность констант, составляющих данный перечисляемый тип, заключается в круглые скобки.

Программист объединяет в одну группу в соответствии с каким-либо признаком всю совокупность значений, составляющих перечисляемый тип. Например, перечисляемый тип Rainbow (РАДУГА) объединяет скалярные значения RED, ORANGE, YELLOW, GREEN, LIGHT_BLUE, BLUE, VIOLET (КРАСНЫЙ, ОРАНЖЕВЫЙ, ЖЕЛТЫЙ, ЗЕЛЕНЫЙ, ГОЛУБОЙ, СИНИЙ, ФИОЛЕТОВЫЙ). Перечисляемый тип Traffic_Light (СВЕТОФОР) объединяет скалярные значения RED, YELLOW, GREEN (КРАСНЫЙ, ЖЕЛТЫЙ, ЗЕЛЕНЫЙ).

Перечисляемый тип описывается в разделе описания типов, например:

```
type

  Rainbow = (RED, ORANGE, YELLOW,
          GREEN, LIGHT_BLUE, BLUE, VIOLET);
```

Каждое значение является константой своего типа и может принадлежать только одному из перечисляемых типов, заданных в программе. Например, перечисляемый тип Traffic_Light не может быть определен в одной программе с типом Rainbow, так как оба типа содержат одинаковые константы.

Описание переменных, принадлежащих к скалярным типам, которые объявлены в разделе описания типов, производится с помощью имен типов. Например:

```
type  Traffic_Light= (RED, YELLOW, GREEN);
var   Section: Traffic_Light;
```

Это означает, что переменная Section может принимать значения RED, YELLOW или GREEN.

Переменные перечисляемого типа могут быть описаны в разделе описания переменных, например:

```
var Section: (RED, YELLOW, GREEN);
```

При этом имена типов отсутствуют, а переменные определяются совокупностью значений, составляющих данный перечисляемый тип.

К переменным перечисляемого типа может быть применим оператор присваивания:

```
Section:= YELLOW;
```

Упорядоченная последовательность значений, составляющих перечисляемый тип, автоматически нумеруется, начиная с нуля и далее через единицу. Отсюда следует, что к перечисляемым переменным и константам могут быть применены операции отношения и стандартные функции Pred, Succ, Ord.

Интервальный тип данных

Отрезок (диапазон значений) любого порядкового типа может быть определен как интервальный (ограниченный) тип. Отрезок задается диапазоном от минимального до максимального значения констант, разделенных двумя точками. В качестве констант могут быть использованы константы, принадлежащие к целому, символьному, логическому или перечисляемому типам. Скалярный тип, на котором строится отрезок, называется базовым типом. Примеры отрезков:

```
1..10
-15..25
'a'..'z'
```

Минимальное и максимальное значения констант называются нижней и верхней границами отрезка, определяющего интервальный тип. Нижняя граница должна быть меньше верхней.

Над переменными, относящимися к интервальному типу, могут выполняться все операции и применяться все стандартные функции, которые допустимы для соответствующего базового типа.

Массивы

Массивы - это совокупности однотипных элементов. Характеризуются они следующим:

- каждый компонент массива может быть явно обозначен и к нему имеется прямой доступ;
- число компонент массива определяется при его описании и в дальнейшем не меняется.

Для обозначения компонент массива используется имя переменной-массива и так называемые индексы, которые обычно указывают желаемый элемент. Тип индекса может быть только порядковым (кроме longint). Чаще всего используется интервальный тип (диапазон).

Описание типа массива задается следующим образом:

type
  имя типа = array[*список индексов*] of *тип*

Здесь *имя типа* - правильный идентификатор; *список индексов* - список одного или нескольких индексных типов, разделенных запятыми; *тип* - любой тип данных.

Вводить и выводить массивы можно только поэлементно.

Пример 1. Ввод и вывод одномерного массива.

```
const
  n = 5;
type
  mas = array[1..n] of integer;
var
  a: mas;
  i: byte;
begin
  writeln('введите элементы массива');
  for i:=1 to n do readln(a[i]);
  writeln('вывод элементов массива:');
  for i:=1 to n do write(a[i]:5);
end.
```

Определить переменную как массив можно и непосредственно при ее описании, без предварительного описания типа массива, например:

```
var a,b,c: array[1..10] of integer;
```

Если массивы a и b описаны как:

```
var
  a = array[1..5] of integer;
  b = array[1..5] of integer;
```

то переменные a и b считаются разных типов. Для обеспечения совместимости применяйте описание переменных через предварительное описание типа.

Если типы массивов идентичны, то в программе один массив может быть присвоен другому. В этом случае значения всех переменных одного массива будет присвоены соответствующим элементам второго массива.

Вместе с тем, над массивами не определены операции отношения. Сравнивать два массива можно только поэлементно.

Так как *mun*, идущий за ключевым словом of в описании массива, - любой тип Турбо Паскаль, то он может быть и другим массивом. Например:

```
type
  mas = array[1..5] of array[1..10] of integer;
```

Такую запись можно заменить более компактной:

```
type
  mas = array[1..5, 1..10] of integer;
```

Таким образом возникает понятие **многомерного** массива. Глубина вложенности массивов произвольная, поэтому количество элементов в списке индексных типов (размерность массива) не ограничена, однако не может быть более 65520 байт.

Работа с многомерными массивами почти всегда связана с организацией вложенных циклов. Так, чтобы заполнить двумерный массив (матрицу) случайными числами, используют конструкцию вида:

```
for i:=1 to m do
for j:=1 to n do a[i,j]:=random(10);
```

Для "красивого" вывода матрицы на экран используйте такой цикл:

```
for i:=1 to m do begin
  for j:=1 to n do write(a[i,j]:5);
  writeln;
end;
```

Строки

Строковый тип данных

Для обработки строковой информации в Турбо Паскаль введен строковый тип данных. Строкой в Паскале называется последовательность из определенного количества символов. Количество символов последовательности называется длиной строки. Синтаксис:

```
var s: string[n];
var s: string;
```

n - максимально возможная длина строки - целое число в диапазоне 1..255. Если этот параметр опущен, то по умолчанию он принимается равным 255.

Строковые константы записываются как последовательности символов, ограниченные апострофами. Допускается формирование строк с использованием записи символов по десятичному коду (в виде комбинации # и кода символа) и управляющих символов (комбинации ^ и некоторых заглавных латинских букв).

Пример:

```
'Текстовая строка'
#54#32#61
'abcde'^A^M
```

Пустой символ обозначается двумя подряд стоящими апострофами. Если апостроф входит в строку как литера, то при записи он удваивается.

Переменные, описанные как строковые с разными максимальными длинами, можно присваивать друг другу, хотя при попытке присвоить короткой переменной длинную лишние символы будут отброшены.

Выражения типа char можно присваивать любым строковым переменным.

В Турбо Паскаль имеется простой доступ к отдельным символам строковой переменной: i-й символ переменной st записывается как st[i]. Например, если st - это 'Строка', то st[1] - это 'С', st[2] - это 'т', st[3] - 'р' и так далее.

Над строковыми данными определена операция слияния (конкантенации), обозначаемая знаком +. Например:

```
a := 'Turbo';
b := 'Pascal';
c := a + b;
```

В этом примере переменная с приобретет значение 'TurboPascal'.

Кроме слияния над строками определены операции сравнения <,>,=,<>,<=,>=. Две строки сравниваются посимвольно, слева направо, по кодам символов. Если одна строка меньше другой по длине, недостающие символы короткой строки заменяются символом с кодом 0.

Процедуры и функции для работы со строками

В системе Turbo Pascal имеется несколько полезных стандартных процедур и функций, ориентированных на работу со строками. Ниже приводится список этих процедур и функций с краткими пояснениями.

Length(s:string):integer
Функция возвращает в качестве результата значение текущей длины строки-параметра

Пример.

```
n := length('Pascal'); {n будет равно 6}
```

Concat(s1,[s2,...,sn]:string):string
Функция выполняет слияние строк-параметров, которых может быть произвольное количество. Каждый параметр является выражением строкового типа. Если длина строки-результата превышает 255 символов, то она усекается до 255 символов. Данная функция эквивалентна операции конкатенации "+" и работает немного менее эффективно, чем эта операция.

Copy(s:string; index:integer; count:integer):string
Функция возвращает подстроку, выделенную из исходной строки s, длиной count символов, начиная с символа под номером index.

Пример.

```
s := 'Система Turbo Pascal';
s2 := copy(s, 1, 7); {s2 будет равно 'Система'}
s3 := copy(s, 9, 5); {s3 будет равно 'Turbo'}
s4 := copy(s, 15, 6); {s4 будет равно 'Pascal'}
```

Delete(var s:string; index,count:integer)
Процедура удаляет из строки-параметра s подстроку длиной count символов, начиная с символа под номером index.

Пример.

```
s := 'Система Turbo Pascal';
delete(s,8,6); {s будет равно 'Система Pascal'}
```

Insert(source:string; var s:string;index:integer)
Процедура предназначена для вставки строки source в строку s, начиная с символа index этой строки.

Пример.

```
s := 'Система Pascal';
insert('Turbo ',s,9); {s будет равно 'Система Turbo Pascal'}
```

Pos(substr,s:string):byte
Функция производит поиск в строке s подстроки substr. Результатом функции является номер первой позиции подстроки в исходной строке. Если подстрока не найдена, то функция возвращает 0.

Пример.

```
s := 'Система Turbo Pascal';
x1 := pos('Pascal', s); {x1 будет равно 15}
x2 := pos('Basic', s); {x2 будет равно 0}
```

Str(X: *арифметическое выражение*; var st: string)
Процедура преобразует численное выражение X в его строковое представление и помещает результат в st.

Val(st: string; x: *числовая переменная*; var code: integer)
Процедура преобразует строковую запись числа, содержащуюся в st, в числовое представление, помещая результат в x. x - может быть как целой, так и действительной переменной. Если в st встречается недопустимый (с точки зрения правил записи чисел) символ, то преобразование не происходит, а в code записывается позиция первого недопустимого символа. Выполнение программы при этом не прерывается, диагностика не выдается. Если после выполнения процедуры code равно 0, то это свидетельствует об успешно произошедшем преобразовании.

В дополнение приведем некоторые функции, связанные с типом char, но которые тем не менее часто используются при работе со строками.
Chr(n: byte): char

Функция возвращает символ по коду, равному значению выражения n. Если n можно представить как числовую константу, то можно также пользоваться записью #n.
Ord(ch: char): byte;
В данном случае функция возвращает код символа ch.
UpCase(c: char): char;
Если c - строчная латинская буква, то функция возвращает соответствующую прописную латинскую букву, в противном случае символ c возвращается без изменения.

Множества

Понятие множества в языке Паскаль основывается на математическом представлении о конечных множествах: это ограниченная совокупность различных элементов. Для построения конкретного множественного типа используется перечисляемый или интервальный тип данных. Тип элементов, составляющих множество, называется базовым типом.

Множественный тип описывается с помощью служебных слов Set of, например:

type M = Set of B;

Здесь M - множественный тип, B - базовый тип.

Пример описания переменной множественного типа:

```
type
  M = Set of 'A'..'D';
var
  MS: M;
```

Принадлежность переменных к множественному типу может быть определена прямо в разделе описания переменных:

var C: Set of 0..7;

Константы множественного типа записываются в виде заключенной в квадратные скобки последовательности элементов или интервалов базового типа, разделенных запятыми, например:

['A', 'C'] [0, 2, 7] [3, 7, 11..14]

Константа вида [ ] означает пустое подмножество. Количество базовых элементов не должно превышать 256. Инициализация величин множественного типа может производиться с помощью типизированных констант:

const seLit: Set of 'A'..'D'= [];

Порядок перечисления элементов базового типа в константах безразличен.

Значение переменной множественного типа может быть задано конструкцией вида [T], где T - переменная базового типа. Например, вполне допустима конструкция:

type T = set of char;

Множество включает в себя набор элементов базового типа, все подмножества данного множества, а также пустое подмножество. Так, переменная T множественного типа

var T: Set of 1..3;

может принимать восемь различных значений:

[ ] [1] [2] [3] [1,2] [1,3] [2,3] [1,2,3]

К переменным и константам множественного типа применимы операции присваивания(:=), объединения(+), пересечения(*) и вычитания(-):

['A','B'] + ['A','D'] даст ['A','B','D']
['A','D'] * ['A','B','C'] даст ['A']
['A','B','C'] - ['A','B'] даст ['C'].

Результат выполнения этих операций есть величина множественного типа.

К множественным величинам применимы операции: тождественность (=), нетождественность (<>), содержится в (<=), содержит (>=). Результат выполнения этих операций имеет логический тип, например:

['A','B'] = ['A','C'] даст FALSE
['A','B'] <> ['A','C'] даст TRUE
['B'] <= ['B','C'] даст TRUE
['C','D'] >= ['A'] даст FALSE.

Кроме этих операций для работы с величинами множественного типа в языке ПАСКАЛЬ используется операция in, проверяющая принадлежность элемента базового типа, стоящего слева от знака операции, множеству, стоящему справа от знака операции. Результат выполнения этой операции - булевский. Операция проверки принадлежности элемента множеству часто используется вместо операций отношения, например:

'A' in ['A', 'B'] даст TRUE,
2 in [1, 3, 6] даст FALSE.

Записи

Запись представляет собой совокупность ограниченного числа логически связанных компонент, принадлежащих к разным типам. Компоненты записи называются полями, каждое из которых определяется именем. Поле записи содержит имя поля, вслед за которым через двоеточие указывается тип этого поля. Поля записи могут относиться к любому типу, допустимому в языке Паскаль, за исключением файлового типа.

Описание записи в языке Паскаль осуществляется с помощью служебного слова record, вслед за которым описываются компоненты записи. Завершается описание записи служебным словом end.

Например, телефонный справочник содержит фамилии и номера телефонов, поэтому отдельную строку в таком справочнике удобно представить в виде следующей записи:

```
type TRec = Record
      FIO: String[20];
      TEL: String[7]
    end;
var rec: TRec;
```

Описание записей возможно и без использования имени типа, например:

```
var rec: Record
          FIO: String[20];
          TEL: String[7]
        end;
```

Обращение к записи в целом допускается только в операторах присваивания, где слева и справа от знака присваивания используются имена записей одинакового типа. Во всех остальных случаях оперируют отдельными полями записей. Чтобы обратиться к

отдельной компоненте записи, необходимо задать имя записи и через точку указать имя нужного поля, например:

rec.FIO, rec.TEL

Такое имя называется **составным**. Компонентой записи может быть также запись, в таком случае составное имя будет содержать не два, а большее количество имен.

Обращение к компонентам записей можно упростить, если воспользоваться оператором присоединения with.

Он позволяет заменить составные имена, характеризующие каждое поле, просто на имена полей, а имя записи определить в операторе присоединения:

with rec do *оператор*;

Здесь rec - имя записи, *оператор* - оператор, простой или составной. *Оператор* представляет собой область действия оператора присоединения, в пределах которой можно не использовать составные имена. Например для нашего случая:

```
with rec do begin
            FIO:='Иванов А.А.';
            TEL:='2223322';
          end;
```

Такая алгоритмическая конструкция полностью идентична следующей:

```
rec.FIO:='Иванов А.А.';
rec.TEL:='2223322';
```

Инициализация записей может производиться с помощью типизированных констант:

```
type
  RecType = Record
      x,y: Word;
       ch: Char;
      dim: Array[1..3] of Byte
    end;

const
  Rec: RecType = ( x: 127;
                   y: 255;
                   ch: 'A';
                   dim: (2, 4, 8) );
```

Подробнее..

Особой разновидностью записей являются записи с вариантами, которые объявляются с использованием зарезервированного слова case. С помощью записей с вариантами вы можете одновременно сохранять различные структуры данных, которые имеют большую общую часть, одинаковую во все структурах, и некоторые небольшие отличающиеся части.

Например, сконструируем запись, в которой мы будем хранить данные о некоторой геометрической фигуре (отрезок, треугольник, окружность).

```
type
TFigure = record
  type_of_figure: string[10];
  color_of_figure: byte;
  ...
  case integer of
    1: (x1,y1,x2,y2: integer);
    2: (a1,a2,b1,b2,c1,c2: integer);
    3: (x,y: integer; radius: word);
end;
var figure: TFigure;
```

Таким образом, в переменной figure мы можем хранить данные как об отрезке, так и о треугольнике или окружности. Надо лишь в зависимости от типа фигуры обращаться к соответствующим полям записи.

Заметим, что индивидуальные поля для каждого из типов фигур занимают тем не менее одно адресное пространство памяти, а это означает, что одновременное их использование невозможно.

В любой записи может быть только одна вариантная часть. После окончания вариантной части в записи не могут появляться никакие другие поля. Имена полей должны быть уникальными в пределах той записи, где они объявлены.

12.Файлы

Введение файлового типа в язык Паскаль вызвано необходимостью обеспечить возможность работы с периферийными (внешними) устройствами ЭВМ, предназначенными для ввода, вывода и хранения данных.

Файловый тип данных или файл определяет упорядоченную совокупность произвольного числа однотипных компонент.

Понятие файла достаточно широко. Это может быть обычный файл на диске, коммуникационный порт ЭВМ, устройство печати, клавиатура или другие устройства.

При работе с файлами выполняются операции ввода - вывода. Операция ввода означает перепись данных с внешнего устройства (из входного файла) в основную память ЭВМ, операция вывода - это пересылка данных из основной памяти на внешнее устройство (в выходной файл).

Файлы на внешних устройствах часто называют физическими файлами. Их имена определяются операционной системой. В программах на языке Паскаль имена файлов задаются с помощью строк. Например, имя файла на диске может иметь вид:

```
'LAB1.DAT'
'c:\ABC150\pr.txt'
'my_files'
```

Типы файлов Турбо Паскаль

Турбо Паскаль поддерживает три файловых типа:

- текстовые файлы;
- типизированные файлы;
- нетипизированные файлы.

Доступ к файлу в программе происходит с помощью переменных файлового типа. Переменную файлового типа описывают одним из трех способов:

file of *mun* - типизированный файл (указан тип компоненты);
text - текстовый файл;
file - нетипизированный файл.

Примеры описания файловых переменных:

```
var
  f1: file of char;
  f2: file of integer;
  f3: file;
  t: text;
```

Стандартные процедуры и функции

Любые дисковые файлы становятся доступными программе после связывания их с файловой переменной, объявленной в программе. Все операции в программе производятся только с помощью связанной с ним файловой переменной.

Assign(f, FileName)
связывает файловую переменную f с физическим файлом, полное имя которого задано в строке FileName. Установленная связь будет действовать до конца работы программы, или до тех пор, пока не будет сделано переназначение.

После связи файловой переменной с дисковым именем файла в программе нужно указать направление передачи данных (открыть файл). В зависимости от этого направления говорят о чтении из файла или записи в файл.

Reset(f)
открывает для чтения файл, с которым связана файловая переменная f. После успешного выполнения процедуры Reset файл готов к чтению из него первого элемента. Процедура завершается с сообщением об ошибке, если указанный файл не найден.

Если f - типизированный файл, то процедурой reset он открывается для чтения и записи одновременно.

Rewrite(f)
открывает для записи файл, с которым связана файловая переменная f. После успешного выполнения этой процедуры файл готов к записи в него первого элемента. Если указанный файл уже существовал, то все данные из него уничтожаются.
Close(f)

закрывает открытый до этого файл с файловой переменной f. Вызов процедуры Close необходим при завершении работы с файлом. Если по какой-то причине процедура Close не будет выполнена, файл все-же будет создан на внешнем устройстве, но содержимое последнего буфера в него не будет перенесено.
EOF(f): boolean
возвращает значение TRUE, когда при чтении достигнут конец файла. Это означает, что уже прочитан последний элемент в файле или файл после открытия оказался пуст.
Rename(f, NewName)
позволяет переименовать физический файл на диске, связанный с файловой переменной f. Переименование возможно после закрытия файла.
Erase(f)
уничтожает физический файл на диске, который был связан с файловой переменной f. Файл к моменту вызова процедуры Erase должен быть закрыт.
IOResult
возвращает целое число, соответствующее коду последней ошибки ввода - вывода. При нормальном завершении операции функция вернет значение 0. Значение функции IOResult необходимо присваивать какой-либо переменной, так как при каждом вызове функция обнуляет свое значение. Функция IOResult работает только при выключенном режиме проверок ошибок ввода - вывода или с ключом компиляции {$I-}.

Работа с типизированными файлами

Типизированный файл - это последовательность компонент любого заданного типа (кроме типа "файл"). Доступ к компонентам файла осуществляется по их порядковым номерам. Компоненты нумеруются, начиная с 0. После открытия файла указатель (номер текущей компоненты) стоит в его начале на нулевом компоненте. После каждого чтения или записи указатель сдвигается к следующему компоненту.

Запись в файл:

Write(f, *список переменных*);

Процедура записывает в файл f всю информацию из списка переменных.

Чтение из файла:

Read(f, *список переменных*);

Процедура читает из файла f компоненты в указанные переменные. Тип файловых компонент и переменных должны совпадать. Если будет сделана попытка чтения несуществующих компонент, то произойдет ошибочное завершение программы. Необходимо либо точно рассчитывать количество компонент, либо перед каждым чтением данных делать проверку их существования (функция eof, см. выше)

Смещение указателя файла:

Seek(f, n);

Процедура смещает указатель файла f на n-ную позицию. Нумерация в файле начинается с 0.

Определение количества компонент:

FileSize(f): longint;

Функция возвращает количество компонент в файле f.

Определение позиции указателя:

FilePos(f): longint;

Функция возвращает порядковый номер текущего компонента файла f.

Отсечение последних компонент файла:

Truncate(f);

Процедура отсекает конец файла, начиная с текущей позиции включительно.

Работа с текстовыми файлами

Текстовый файл - это совокупность строк, разделенных метками конца строки. Сам файл заканчивается меткой конца файла. Доступ к каждой строке возможен лишь последовательно, начиная с первой. Одновременная запись и чтение запрещены.

Чтение из текстового файла:

Read(f, *список переменных*);
ReadLn(f, *список переменных*);

Процедуры читают информацию из файла f в переменные. Способ чтения зависит от типа переменных, стоящих в списке. В переменную char помещаются символы из файла. В числовую переменную: пропускаются символы-разделители, начальные пробелы и считывается значение числа до появления следующего разделителя. В переменную типа string помещается количество символов, равное длине строки, но только в том случае, если раньше не встретились символы конца строки или конца файла. Отличие ReadLn от Read в том, что в нем после прочтения данных пропускаются все оставшиеся символы в данной строке, включая метку конца строки. Если список переменных отсутствует, то процедура ReadLn(f) пропускает строку при чтении текстового файла.

Запись в текстовый файл:

Write(f, *список переменных*);
WriteLn(f, *список переменных*);

Процедуры записывают информацию в текстовый файл. Способ записи зависит от типа переменных в списке (как и при выводе на экран). Учитывается формат вывода. WriteLn от Write отличается тем, что после записи всех значений из переменных записывает еще и метку конца строки (формируется законченная строка файла).

Добавление информации к концу файла:

Append(f)

Процедура открывает текстовый файл для добавления информации к его концу. Используйте эту процедуру вместо Rewrite.

Работа с нетипизированными файлами

Нетипизированные файлы - это последовательность компонент произвольного типа.

Открытие нетипизированного файла:

Reset(f, BufSize)
Rewrite(f, BufSize)

Параметр BufSize задает число байтов, считываемых из файла или записываемых в него за одно обращение. Минимальное значение BufSize - 1 байт, максимальное - 64 К байт. Если BufSize не указан, то по умолчанию он принимается равным 128.

Чтение данных из нетипизированного файла:

BlockRead(f, X, Count, QuantBlock);

Эта процедура осуществляет за одно обращение чтение в переменную X количества блоков, заданное параметром Count, при этом длина блока равна длине буфера. Значение Count не может быть меньше 1. За одно обращение нельзя прочесть больше, чем 64 К байтов.

Необязательный параметр QuantBlock возвращает число блоков, прочитанных текущей операцией BlockRead. В случае успешного завершения операции чтения QuantBlock = Count, в случае аварийной ситуации параметр QuantBlock будет содержать число удачно прочитанных блоков. Отсюда следует, что с помощью параметра QuantBlock можно контролировать правильность выполнения операции чтения.

Запись данных в нетипизированный файл:

BlockWrite(f, X, Count, QuantBlock);

Эта процедура осуществляет за одно обращение запись из переменной X количества блоков, заданное параметром Count, при этом длина блока равна длине буфера.

Необязательный параметр QuantBlock возвращает число блоков, записанных успешно текущей операцией BlockWrite.

Для нетипизированных файлов можно использовать процедуры Seek, FIlePos и FileSize, аналогично соответствующим процедурам типизированных файлов.