Тема 2.

Основные элементы языка Паскаль

Содержание

1. Общая характеристика языка Паскаль

Языки программирования

Язык Паскаль был разработан Никласом Виртом первоначально для целей обучения программированию. В настоящее время он получил широкое распространение по ряду объективных причин:

1) По своей идеологии Паскаль наиболее близок к современной методике и технологии программирования. В частности, он достаточно полно отражает идеи структурного программирования, что довольно хорошо видно даже из основных управляющих структур языка.
2) Паскаль хорошо приспособлен для применения технологии разработки программ сверху-вниз (пошаговой детализации).
3) Паскаль содержит большое разнообразие различных структур данных, что обеспечивает простоту алгоритмов, а следовательно снижение трудоемкости при разработке программ.

Отличия алгоритмических языков от машинных

- алгоритмический язык обладает гораздо большими выразительными возможностями, т.е. его алфавит значительно шире алфавита машинного языка, что существенно повышает наглядность текста программы;
- набор операций, допустимых для использования, не зависит от набора машинных операций, а выбирается из соображений удобства формулирования алгоритмов решения задач определенного класса;
- формат предложений достаточно гибок и удобен для использования, что позволяет с помощью одного предложения задать достаточно содержательный этап обработки данных;

Отличия алгоритмических языков от машинных

- требуемые операции задаются в удобном для человека виде, например, с помощью общепринятых математических обозначений;
- для задания операндов операций, используемым в алгоритме данным присваиваются уникальные имена, выбираемые программистом, и ссылка на операнды производится, в основном, по именам;
- в языке может быть предусмотрен значительно более широкий набор типов данных по сравнению с набором машинных типов данных.

Языки программирования

Из вышеперечисленного следует, что алгоритмический язык в значительной мере является ***машинно-независимым***.

2. Основные понятия языка Паскаль

Алфавит языка Паскаль

Алфавит включает в себя буквы, цифры и специальные символы.

1. **Прописные и строчные буквы латинского алфавита:**

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

_ знак подчеркивания (используется в именах вместо пробела)

2. **Десятичные цифры**: **0 1 2 3 4 5 6 7 8 9**

Алфавит языка Паскаль

3. **Прописные и строчные буквы русского алфавита**
(для комментариев, для вывода сообщений на экран):

А Б В Г Д Е Ё Ж З И Й К Л М Н О П Р С Т У Ф Х Ц Ч Ш Щ Ъ Ы Ь Э Ю Я

а б в г д е ё ж з и й к л м н о п р с т у ф х ц ч ш щ ъ ы ь э ю я

Алфавит языка Паскаль

4. **Специальные символы**:

+ плюс **–** минус

***** звездочка **/** дробная черта (слэш)

> больше **<** меньше **=** равно

: двоеточие **;** точка с запятой

пробел

' апостроф **,** запятая **.** точка

^ крышка **@** коммерческое а (эт)

$ знак доллара **#** номер

[ ] квадратные скобки

{ } фигурные скобки

( ) круглые скобки

Алфавит языка Паскаль

5. **Составные символы, которые нельзя разделять пробелами**

<> не равно

<= меньше или равно

>= больше или равно

:= присваивание

.. промежуток значений

(* *) (. .) начало и конец комментариев

(замена фигурных скобок)

Зарезервированные слова

ABSOLUTE	EXTERNAL	MOD	SHL
AND	FAR	NEAR	SHR
ARRAY	FILE	NIL	STRING
ASM	FOR	NOT	THEN
ASSEMBLER	FORWARD	OBJECT	TO
BEGIN	FUNCTION	OF	TYPE
CASE	GOTO	OR	UNIT
CONST	IF	PACKED	UNTIL
CONSTRUCTOR	IMPLEMENTATION	PRIVATE	USES
DESTRUCTOR	IN	PROCEDURE	VAR
DIV	INHERITED	PROGRAM	VIRTUAL
DO	INLINE	PUBLIC	WHILE
DOWNTO	INTERFACE	RECORD	WITH
ELSE	INTERRUPT	REPEAT	XOR
END	LABEL	SET	

Структура программы

Для того чтобы компилятор правильно понял, какие именно действия от него ожидаются, *программа* должна быть оформлена в полном соответствии с *синтаксисом* (правилами построения программ) языка Паскаль.

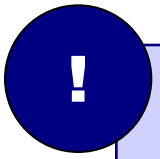
Структура программы

```
program <имя программы>;
Uses …; { подключаемые модули и библиотеки }
Label …;    { раздел объявления меток          }
Const …;    { раздел объявления констант       }
Type …; { раздел объявления типов              }
Var …;  { раздел объявления переменных     }

Procedure …;  { раздел описания процедур  }
Function …;   { раздел описания функций   }

begin   { начало основного блока программы  }
  …     { операторы основного блока программы   }
end.        { конец основного блока программы   }
```

комментарии в фигурных скобках не обрабатываются

Структура программы

Любой из перечисленных необязательных разделов может встречаться в тексте программы более одного раза, их общая последовательность также может меняться, но при этом всегда должно выполняться ***главное правило языка Паскаль***:



прежде чем объект будет использован, он должен быть объявлен и описан.

Оформление текста программы

Шапка – комментарий в начале процедур и функций.

```
{-----------------------------------------
  Max – максимальное из двух чисел
   Вход: a, b – исходные числа
   Выход: максимальное из a и b
-----------------------------------------}
function Max(a, b: integer): integer;
begin
  ...
end;
```

Оформление текста программы

Отступы – тело цикла, условного оператора, оператора выбора и т.п. сдвигается вправо на 2-3 символа.

```
for i:=1 to n do begin j := 0; while j < i
do begin j := j + 1; k := k mod N; end; k
:= k + 1; end;
```

лесенка

```
for i:=1 to n do
begin
  j := 0;
  while j < i do
  begin
    j := j + 1;
    k := k mod N;
  end;
  k := k + 1;
```

- легче читать текст программы
- видны блоки **begin-end** (где начинаются и заканчиваются)

Скорость работы программы не меняется!

Оформление текста программы

- «говорящие» имена функций, процедур, переменных: **Sum, ShowMenu, count, speed**.
- пробелы в операторах

```
if(a<b)then b:=c+d;
```

```
if ( a < b ) then
  b := c + d;
```

- выделение пустыми строками и комментариями важных блоков

```
   { ввод данных }
writeln( 'Введите число' );
read ( n );
   { вычисления }
n2 := n*n;
   { вывод результата }
writeln ( 'Его квадрат ', n2);
```

Порядок разработки программы

1. Программист должен знать алгоритм решения задачи
2. Нужно придумать имена константам, переменным
3. Нужно определить какого типа будут переменные
4. Перед вычислениями нужно задать или ввести исходные данные для решения задачи
5. Задать действия необходимые для получения результата
6. Полученный результат нужно вывести
7. Проверить работоспособность программы на нескольких исходных данных

Из чего состоит программа?

Константа – постоянная величина, имеющая имя.

Переменная – изменяющаяся величина, имеющая имя (ячейка памяти).

Выражение состоит из констант, переменных, указателей функций, знаков операций и скобок и служит для задания правила вычисления некоторого значения.

Комментарий – строка (или несколько строк) из произвольных символов, заключенная в фигурные скобки.

Оператор – неделимый элемент программы, который позволяет выполнять определенные алгоритмические действия.

Из чего состоит программа?

Процедура – вспомогательный алгоритм, описывающий некоторые действия (рисование окружности).

Функция – вспомогательный алгоритм для выполнения вычислений (вычисление квадратного корня, **sin**).

Идентификаторы

Имена, даваемые программным объектам (*константам*, типам, *переменным*, функциям и процедурам, да и всей *программе* целиком) называются ***идентификаторами***.

Каждый объект программы должен иметь уникальный идентификатор.

Идентификаторы могут иметь любую ***длину***, но если у двух имен первые 63 символа совпадают, то такие имена считаются идентичными. Максимальная длина - 127 символов.

Вы можете давать программным объектам любые имена, но необходимо, чтобы они отличались от *зарезервированных слов* языка Паскаль, потому что *компилятор* все равно не примет *переменные* с "чужими" именами.

Идентификаторы

Имена могут включать

- латинские буквы (A-Z)

заглавные и строчные буквы не различаются

- цифры

имя не может начинаться с цифры

- знак подчеркивания _

Имена НЕ могут включать

- русские буквы
- пробелы
- скобки, знаки +, =, !, ? и др.

Константы

Константа - это объект, значение которого известно еще до начала работы программы.

- необходимы для оформления наглядных *программ*,
- незаменимы при использовании в тексте *программы* многократно повторяемых значений,
- удобны в случае необходимости изменения этих значений сразу во всей программе.

Константы

```
const
     i2 = 45;  { целое число }
   pi = 3.14;  { вещественное  число }
   qq = 'Вася';  { строка символов }
   L  = True;  { логическая величина }
```

целая и дробная часть отделяются точкой

можно использовать русские буквы!

может принимать два значения:

- True (истина, «да»)
- False (ложь, «нет»)

Константы

В языке Паскаль существует три вида констант:

- *неименованные константы* (цифры и числа, символы и строки, множества);
- *именованные нетипизированные константы*;
- *именованные типизированные константы*.

Неименованные константы

Неименованные константы не имеют имен, и потому их не нужно описывать.

Тип неименованной константы определяется автоматически, по умолчанию:

- любая последовательность цифр (возможно, предваряемая знаком "**-**" или "**+**" или разбиваемая одной точкой) воспринимается компилятором как число (целое или вещественное);
- любая последовательность символов, заключенная в апострофы, воспринимается как строка;
- любая последовательность целых чисел либо символов через запятую, обрамленная квадратными скобками, воспринимается как множество.

Кроме того, существуют две специальные константы ***true*** и ***false***, относящиеся к логическому типу данных.

Неименованные константы

Примерами использования неименованных констант могут послужить следующие операторы:

```
int1 := -10;
real2 := 12.075 + x;
char3 := 'z';
string4 := 'abc' + string44;
set5 := [1,3,5] * set55;
boolean6 := true;
```

Нетипизированные константы

Именованные константы, как следует из их названия, должны иметь имя. Стало быть, эти имена необходимо сообщить компилятору, то есть описать в специальном разделе ***const***.

Если не указывать тип константы, то по ее внешнему виду компилятор сам определит, к какому типу ее отнести.

Любую уже описанную константу можно использовать при объявлении других констант, переменных и типов данных.

Нетипизированные константы

Примеры описания нетипизированных констант:

```
const n = -10;
   m = 1000000000;
   mmm = n*100;
   x = 2.5;
   c = 'z';
   s = 'компьютер';
   b = true;
```

Типизированные константы

Типизированные именованные константы представляют собой *переменные* с начальным значением, которое к моменту старта программы уже известно.

Следовательно,

- во-первых, типизированные константы нельзя использовать для определения других констант, типов данных и переменных,
- во-вторых, их значения можно изменять в процессе работы программы.

Типизированные константы

Описание *типизированных констант* производится по следующему шаблону:

```
const
  < имя константы > : < тип константы > =
     <начальное значение>;
```

Типизированные константы

Примеры описания типизированных констант:

```
const n: integer = -10;
   x: real = 2.5;
   c: char = 'z';
   b: boolean = true;
```

Переменные

Переменная – это величина, имеющая имя, тип данных и значение. Значение переменной можно изменять во время работы программы.

Тип данных - это характеристика диапазона значений, которые может принимать переменная, относящиеся к этому типу данных.

Наиболее часто применяемые типы переменных:

- integer { целая }
- real { вещественная }
- char { один символ }
- string { символьная строка }
- boolean { логическая }

Переменные

Все используемые в программе переменные должны быть описаны в специальном разделе ***var*** по следующему шаблону:

```
var <имя переменной 1> [, <имя переменной 2, ...>]     : <имя типа 1>;
    <имя переменной 3> [, <имя переменной 4, ...>]   : <имя типа 2>;
```

Пример объявления переменных (выделение памяти):

```
var  a, b: integer;
     Q: real;
     s1, s2: string;
```

Комментарии

Используют для пояснений, необходимых для лучшего понимания программы.

Комментарий представляет собой пояснительный текст, который можно записывать в любом месте программы, где разрешен пробел.

Текст комментария ограничен символами **{** и **}** или **(*** и ***)**. Может содержать любые комбинации латинских и русских букв, цифр и других символов алфавита языка Паскаль.

Примеры:

```
{ Комментарий к программе  Regress   }
{ Блок вычисления
      корней уравнения }
(* Переменная для вычисления суммы ряда *)
```

Комментарии

По месту положения в программе комментарии подразделяются на три класса:

1) объясняющие назначение программы;
2) поясняющие смысл идентификаторов констант и переменных;
3) объясняющие труднопонимаемые элементы алгоритма.

Комментарии

Внутри самого комментария символы `}` или `*)` встречаться не должны.

Во время компилирования программы комментарии игнорируются. Следовательно, их можно добавлять в любом месте программы.

Можно даже разорвать оператор вставкой комментария.

Кроме того, все, что находится после ключевого слова `end.`, завершающего текст программы, компилятор тоже воспринимает как комментарий.

3. Типы данных и операции

Компиляторы языка Паскаль требуют, чтобы сведения об объеме памяти, необходимой для работы программы, были предоставлены до начала ее работы.

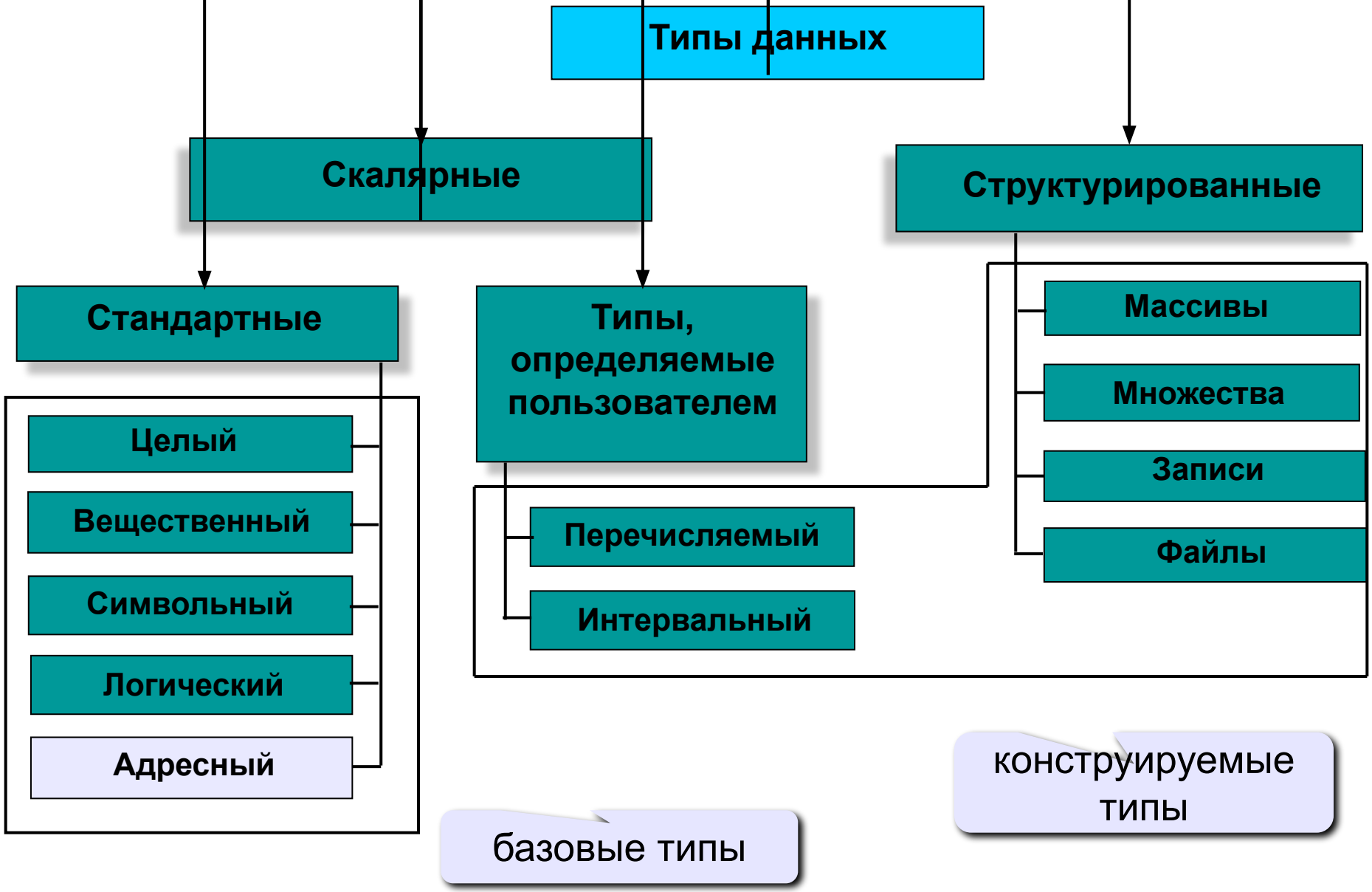
Для этого в разделе описания переменных (***var***) нужно перечислить все переменные, используемые в программе. Кроме того, необходимо также сообщить компилятору, сколько памяти каждая из этих переменных будет занимать. А еще было бы неплохо заранее условиться о различных операциях, применимых к тем или иным переменным.

Все это можно сообщить программе, просто указав тип будущей переменной. Имея информацию о типе переменной, компилятор "понимает", сколько байт необходимо отвести под нее, какие действия с ней можно производить и в каких конструкциях она может участвовать.

Тип данных 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.

Классификация типов данных



Целочисленные типы данных

Целочисленные типы определяют константы, переменные и функции, значения которых реализуются множеством целых чисел.

Целочисленный тип данных *BYTE*

Допустимые значения: **от 0 до 255**
Пример: **5 58**

Целочисленный тип данных *WORD*

Допустимые значения: **от 0 до 65535**
Пример: **5 58 43467**

Целочисленные типы данных

Целочисленный тип данных *SHORTINT*

Допустимые значения: **от –128 до 127**
Пример: **-5 0 58**

Целочисленный тип данных *INTEGER*

Допустимые значения: **от –32768 до 32767**
Пример: **5 –58 0 10000 –32768**

Целочисленные типы данных

Целочисленный тип данных *LONGINT*

Допустимые значения: **от –2147483648 до 2147473647**
Пример: **5 –3345550 3345550 0**

Целочисленные типы данных

Над целочисленными операндами выполняются арифметические операции, операции отношения.

Арифметические операции

+ сложение,

– вычитание,

***** умножение,

MOD и **DIV** целочисленное деление:

MOD – остаток от деления операндов

DIV – целая часть частного.

Результат выполнения операции является величиной целого типа.

```
21 Div 4 = 5    21 MOD 5 = 1
4 DIV 8 = 0   4 mod 10 = 4
(-2 mod 5)= -2
```

Целочисленные типы данных

Операции отношения (сравнения)

= равенство, **<>** неравенство,
< меньше, **>** больше,
<= меньше или равно, **>=** больше или равно

Результат выполнения операции является величиной **логического типа** (**True** или **False**).

Вещественные типы данных

Вещественные типы определяют константы, переменные и функции, значения которых реализуются множеством действительных (вещественных) чисел.

Вещественный тип данных *REAL*

Допустимые значения: **от 2.9e–39 до 1.7e+38**
Пример: **5.567 58e–3 1.76e+8 1.0**

Вещественный тип данных *SINGLE*

Допустимые значения: **от 1.5e–45 до 3.4e+38**

Вещественные типы данных

Вещественный тип данных *DOUBLE*

Допустимые значения: **от 5.0e–324 до 1.7e+308**

Вещественный тип данных *EXTENDED*

Допустимые значения: **от 3.4e–4932 до 1.1e+4932**

Вещественный тип данных *COMP*

Допустимые значения: **от –9.2e+18 до 9.2e+18**

Вещественные типы данных

Над вещественными операндами выполняются арифметические операции, операции отношения.

Арифметические операции

- **+** сложение,
- **–** вычитание,
- ***** умножение,
- **/** деление.

Результат выполнения операции является величиной вещественного типа.

```
3.0+7.51          6.2-10/33
5.23*(-10.1E2)        21.2/(11.21E-2)
```

Вещественные типы данных

Операции отношения (сравнения)

= равенство, **<>** неравенство,
< меньше, **>** больше,
<= меньше или равно, **>=** больше или равно

Результат выполнения операции является величиной **логического типа** (**True** или **False**).

Логический тип данных *BOOLEAN*

Данные, которые могут принимать логические значения ***True*** и ***False***.

Логические операции

Not – логическое отрицание
And – логическое И (конъюнкция)
Or – логическое ИЛИ (дизъюнкция)
Xor – логическое исключающее ИЛИ

```
If (a>b) and (b>c) then
……..……..
```

Логический тип данных *BOOLEAN*

Операции отношения (сравнения)

Логический тип определен таким образом, что

True < False.

Это позволяет применять к булевским операндам все операции сравнения:

= равенство, **<>** неравенство,

< меньше, **>** больше,

<= меньше или равно, **>=** больше или равно

Типы данных: символы

Символьный тип данных *CHAR*

Допустимые значения: **один символ из кодовой таблицы (256 символов кода ASCII)**

Пример: ***Y f 4 я Д ****

Применимы все операции отношения, функции преобразования типов ***Ord()*** и ***Chr()***, функции, которые определяют предыдущий и последующий символы ***Pred()*** и ***Succ()***.

Типы данных: символы

Строковый тип данных *STRING*

Строка типа ***String*** – это цепочка символов типа ***Char***.

String используется для хранения текстовых сообщений.

Допустимые значения: **любой текст длинной не более 255 символов**

Пример: **Всё, что вы хотите написать!**

Типы данных, определяемые программистом

Интервальный тип данных

Позволяет задавать две константы, определяющие границы диапазона значений для данной переменной.

Обе константы должны принадлежать одному из стандартных типов (тип ***real*** здесь недопустим).

Значение первой константы должно быть обязательно меньше второй.

Формат описания типа:

```
Type  <Имя типа> =
      <константа 1> .. <константа 2> ;
```

Например:

```
Type  Dni = 1 ..31;
```

Типы данных, определяемые программистом

Перечисляемый тип данных

Определение перечисляемого типа задает упорядоченное множество значений путем перечисления имен, обозначающих эти значения.

Формат описания типа:

```
Type  <Имя типа> = (<имя данного 1>,
       <имя данного 2>, …, <имя данного k>);
```

Например:

```
Type Weekday = (Monday, Tuesday, Wednesday,
      Thursday, Friday, Saturday, Sunday);
      Colour = (Red, Orange, Yellow, Green, Blue,
         Black);
      Operation = (Plus, Minus, Times, Divide);
```

Например:

```
Type
     Operation = (Plus, Minus, Times, Divide);
Var
     a :  Operation;

     Begin
     ……….
     a := Operation(b);

     if a = Plus then
     ……….
```

4. Стандартные функции

Арифметические функции

Функция	Назначение	Тип аргумента	Тип результата
ABS(X)	Абсолютное значение (модуль) аргумента	**целый вещественный**	**целый вещественный**
ARCTAN(X)	Арктангенс аргумента	**целый вещественный**	**вещественный**
COS(X)	Косинус аргумента	**целый вещественный**	**вещественный**
EXP(X)	Экспонента аргумента	**целый вещественный**	**вещественный**
FRAC(X)	Дробная часть числа	**вещественный**	**вещественный**
INT(X)	Целая часть числа	**вещественный**	**вещественный**
LN(X)	Натуральный логарифм	**целый вещественный**	**вещественный**
PI	Значение величины π=3.1415926535		**вещественный**
SIN(X)	Синус аргумента	**целый вещественный**	**вещественный**
SQR(X)	Квадрат аргумента	**вещественный**	**вещественный**
SQRT(X)	Квадратный корень аргумента	**целый вещественный**	**вещественный**

Функции преобразования типов

Эти функции предназначены для преобразования типов величин, например, символа в целое число, вещественного числа в целое и т.д.

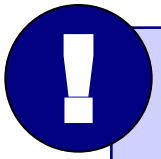
- ***Chr(X)*** – преобразование ASCII-кода в символ. Аргумент функции – целого типа от 0 до 255. Результатом – символ, соответствующий данному коду.
- ***High(X)*** – получение максимального значения величины. Аргумент функции – параметр или идентификатор порядкового типа (целые, логический, символьный, перечисляемый), типа-массива, типа-строки. Результат функции для величины порядкового типа – максимальное значение этой величины, типа-массива – максимальное значение индекса, типа-строки – объявленный размер строки.
- ***Low(X)*** – получение минимального значения величины. Аргумент функции и результат функции аналогичны ***High(X)***.

Функции преобразования типов

- ***Ord(X)*** – преобразование любого порядкового типа в целый тип. Аргумент функции – логический, символьный, перечисляемый тип. Результат – величина типа *Longint*.
- ***Round(X)*** – округление вещественного числа до ближайшего целого. Результат – округленная до ближайшего целого величина типа *Longint*.
- ***Trunc(X)*** – получение целой части вещественного числа. Результат – целая часть этого числа типа *Longint*.

Примеры арифметических функций

`sin(x)` `cos(x)` `arctan(x)`



Аргумент ***x*** для тригонометрических функций указывается в радианах.

```
a:=Pi/180*30;
s:=sin(a);   c:=cos(a);
t:=s/c;        ct:=c/s;
```

синус, косинус, тангенс и котангенс угла 30°

Примеры арифметических функций

```
exp(x)
```

e^x (экспонента числа, e≈2.7183)

```
ln(x)
```

ln x (натуральный логарифм)

```
Exp(b*Ln(a))
```

a^b

! Вычислять a^b таким образом можно только для положительных значений **a**.

```
exp(7*ln(x-3))
```

$(x-3)^7$

```
Power(x-3,7)
```

```
exp(x*ln(2))
```

2^x

```
Power(2,x)
```

Примеры арифметических функций

```
Round(x)
```

Перевод дробного числа в целое с округлением

```
Trunc(x)
```

Перевод дробного числа в целое с отбрасыванием дробной части

```
a1:=Round(2.34);
```

a1 = 2

```
a2:=Trunc(2.34);
```

a2 = 2

```
b1:=Round(8.51);
```

b1 = 9

```
b2:=Trunc(8.51);
```

b2 = 8

```
c1:=Round(-3.7);
```

c1 = -4

Примеры арифметических функций

Вычисляемая функция	Математическая запись	Запись на языке Паскаль
x^y	$e^{y \cdot \ln x}$	***Exp(y*ln(x))***
$\sqrt[y]{x}$	$x^{1/y} = e^{1/y \cdot \ln x}$	***Exp(1/y*ln(x))***
$\arcsin(x)$	$arctg\left(x \Big/ \sqrt{1-x^2}\right)$	***arctan(x/sqrt(1-x*x))***
$\arccos(x)$	$arctg\left(\sqrt{1-x^2} \Big/ x\right)$	***arctan(sqrt(1-x*x)/x)***
$\log_y(x)$	$\ln x / \ln y$	***Ln(x)/Ln(y)***
$tg(x)$	$\sin x / \cos x$	***Sin(x)/Cos(x)***

Генераторы случайных чисел

- ***Randomize*** – стандартная процедура установки датчика случайных чисел в исходное состояние.
- ***Random*** – стандартная функция формирования случайного дробного числа из диапазона от 0 до 1.
- ***Random(N)*** – стандартная функция формирования случайного целого числа из диапазона от 0 до N-1.

Примеры получения случайных чисел

`a:=Random;`	0<a<1
`x:=Random+10;`	10<x<11
`y:=5*Random;`	0<y<5
`c:=10*Random-5;`	-5<c<5
`a:=Random(3);`	0,1,2
`y:=Random(5)+3;`	3,4,5,…,7
`c:=Random(8)-5;`	-5,-4,…,2

Арифметические выражения

Все арифметические операции можно сочетать друг с другом - с учетом допустимых для их операндов типов данных.

В роли операндов любой операции могут выступать переменные, константы, вызовы функций или выражения, построенные на основе других операций. Все вместе и называется ***выражением***.

Примеры арифметических выражений

```
(x<0) and (y>0)
```

выражение, результат которого принадлежит к типу ***boolean***

```
z shl abs(k)
```

вторым операндом является вызов стандартной функции

```
(x mod k) + min(a,b) +
trunc(z)
```

сочетание арифметических операций и вызовов функций

```
odd(round(x/abs(y)))
```

"многоэтажное" выражение

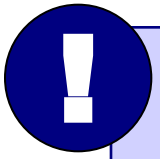
```
sin(-x*x-1/(1+x))*(koef[1]*koef[1]-4*koef[2])
;
```

"многоэтажное" выражение с использованием массива

5. Простейшие операторы

Оператор языка Паскаль – это неделимый элемент программы, который позволяет выполнять определенные алгоритмические действия.

Если говорить строго, то ***оператором*** называется (минимальная) структурно законченная единица программы.



Все операторы должны заканчиваться знаком ";" (точка с запятой), и ни один оператор не может разрываться этим знаком.

Единственная возможность не ставить после оператора ";" появляется в том случае, когда сразу за этим оператором следует ключевое слово ***end***.

Простейшие операторы языка

- ***a:= b;*** - ***присваивание*** переменной ***a*** значения переменной ***b***. В правой части присваивания может находиться переменная, константа, арифметическое выражение или вызов функции.
- ***;*** - ***пустой оператор***, который можно вставлять куда угодно, а также вычеркивать откуда угодно, поскольку на целостность программы это никак не влияет.
- ***Операторные скобки***, превращающие несколько операторов в один:

```
begin
   <несколько операторов>
end;
```

Простейшие операторы языка

- Оператор безусловного перехода (**GoTo**).
- Операторы вызова подпрограммы (например, **Abs**, **Write**, **ReadLn**).

Составные операторы языка

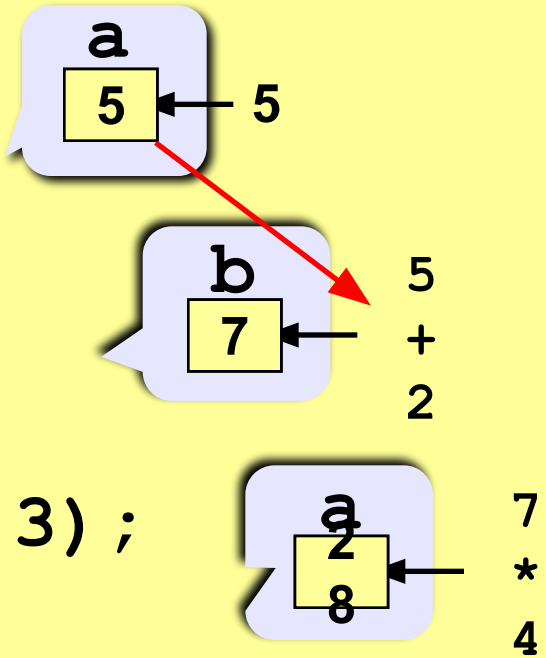
- Составной оператор – это последовательность операторов, заключенных в операторные скобки ***Begin*** и ***End***.
- Условные операторы (***If***, ***Case***).
- Операторы цикла (***Repeat***, ***While***, ***For***).
- Оператор присоединения (***With***).

Как изменить значение переменной?

Оператор присваивания служит для изменения значения переменной.

Пример:

```
program qq;
var a, b: integer;
begin
  a := 5;
  b := a + 2;
  a := (a + 2)*(b - 3);
end.
```



Оператор присваивания

```
<имя переменной> := <выражение>;
```

Арифметическое выражение может включать

- константы
- имена переменных
- знаки арифметических операций:

`+ -  *  /  div  mod`

умножение | деление | деление нацело | остаток от деления

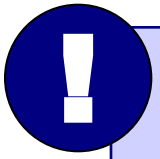
- вызовы функций
- круглые скобки ( )

Оператор присваивания

Имя слева от символа присваивания := является именем переменной, которой присваивается значение выражения, стоящего справа.

С помощью этого оператора осуществляется преобразование информации.

Он предназначен для вычисления нового значения некоторой переменной, а также для определения значения, возвращаемого функцией.



Тип выражения в правой части оператора присваивания должен совпадать или быть совместимым с типом переменной из левой части. Компилятор на этапе синтаксического анализа программы осуществляет эту проверку - так называемый контроль типов.

Примеры операторов присваивания

```
Root1 := Pi*(x - y);
Discriminant := Sqrt(b*b-4*a*c)/2/A;
Index := Index + 1;
F:= sin(-x*x-1/(1+x));
R:=(r1 + r2)/(r1*r2);
D:=((a = b) AND (c = d)) OR (a > d);
```

Какие операторы неправильные?

```
program qq;
var a, b: integer;
   x, y: real;
begin
  a := 5;
  10 := x;
  y := 7,8;
  b := 2.5;
  x := 2*(a + y);
  a := b + x;
end.
```

имя переменной должно быть слева от знака **:=**

целая и дробная часть отделяются **точкой**

нельзя записывать вещественное значение в целую переменную

6. Ввод и вывод данных

Ввод и вывод: консоль

Любой алгоритм должен быть результативным. В общем случае это означает, что он должен сообщать результат своей работы потребителю: пользователю-человеку или другой программе.

В программировании существует специальное понятие ***консоль***, которое обозначает клавиатуру при вводе и монитор при выводе.

Ввод с консоли

Для того чтобы получить данные, вводимые пользователем вручную (т.е. с консоли), применяются команды

read (<список_ввода>)

readln (<список_ввода>)

Первая из этих команд считывает все предложенные ей данные, оставляя курсор в конце последней строки ввода, а вторая - сразу после окончания ввода переводит курсор на начало следующей строки. В остальном же их действия полностью совпадают.

Ввод с консоли

Список ввода - это последовательность имен *переменных*, разделенных запятыми.

Например, при помощи команды

```
readln(k, x, c, s);
{k:integer; x:real; c:char; s:string}
```

программа может получить с клавиатуры данные сразу для четырех переменных, относящихся к различным типам данных.

Ввод с консоли

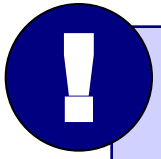
Вводимые значения необходимо разделять пробелами, а завершать ввод - нажатием клавиши Enter.

Ввод данных заканчивается в тот момент, когда последняя переменная из списка ввода получила свое значение.

Следовательно, вводя данные при помощи приведенной на слайде выше команды, вы можете нажать Enter четыре раза - после каждой из вводимых переменных, - либо же только один раз, предварительно введя все четыре переменные в одну строчку (обязательно нужно разделить их пробелами).

Вывод на консоль

Важное замечание: ожидая от человека ввода с клавиатуры, не нужно полагать, что он окажется ясновидящим и просто по мерцанию курсора на черном экране догадается, какого типа переменная нужна ожидающей программе.



Старайтесь всегда придерживаться правила: «пустой" ввод недопустим! Перед тем как считывать что-либо с консоли, необходимо сообщить пользователю, что именно он должен ввести: смысл вводимой информации, тип данных, максимальное и минимальное допустимые значения и т.п.

Вывод на консоль

Примером неплохого приглашения служит, скажем, такая строчка:

Введите два вещественных числа (0.1<x,y<1000000) - длины катетов.

Впрочем, и ее можно улучшить, сообщив пользователю не только допустимый диапазон ввода, но и ожидаемую точность (количество знаков после запятой).

Вывод на консоль

Для того чтобы вывести на экран какое-либо сообщение, воспользуйтесь процедурами

```
write( <список_вывода> )
```

```
writeln( <список_вывода> )
```

Первая из них, напечатав на экране все, о чем ее просили, оставит курсор в конце выведенной строки, а вторая переведет его в начало следующей строчки.

Вывод на консоль

Список вывода может состоять из нескольких переменных или констант, записанных через запятую; все эти переменные могут относиться к целому, вещественному, символьному или булевскому типам.

Например:

```
writeln(a, b, c);
```

В качестве элемента списка вывода кроме имен переменных могут использоваться выражения и строки.

Оператор ***Writeln*** без параметров реализует пропуск строки и переход к началу следующей строки.

Форматированный вывод

Если для вывода информации воспользоваться командой, приведенной в конце предыдущего слайда, то выводимые символы окажутся "слепленными".

Чтобы этого не случилось, нужно либо позаботиться о пробелах между выводимыми переменными:

```
writeln(a, ' ', b, ' ', c);
```

Но предпочтительнее задать для всех (или хотя бы для некоторых) переменных ***формат вывода***:

```
writeln(a:5, b, c:10:5);
```

количество позиций, выделяемых под всю переменную

количество позиций, выделяемых под дробную часть числа

Форматированный вывод

Например, если a = 25, b = 'x', a c = 10.5, то после выполнения команды

```
writeln(a:5, ' ',b, c:10:5);
```

на экране или в файле будет записано следующее:

```
_ _ _25_x_ _10.50000
```

Особенно важен формат при выводе *вещественных переменных*.

К примеру, если не указать формат, то число 10.5 будет выведено как

1.0500000000E+0001.

Такой формат называется записью с плавающей точкой.

Форматированный вывод

Если же задать только общую длину вещественного числа, не указывая длину дробной части, то оно будет занимать на экране заданное количество символов (в случае надобности, спереди будет добавлено соответствующее количество пробелов), но при этом останется в формате плавающей точки.

Примеры форматированного вывода

Для целочисленного выражения

Вывод десятичного представления величины **J**, начиная с позиции расположения курсора.

Оператор вывода	Значение переменной	Результат
`Write(J);`	**236**	**236**
`Write(J, J+1, J+2);`	**236**	**236237238**

Вывод десятичного представления величины **J** в крайние правые позиции поля заданной шириной.

Оператор вывода	Значение переменной	Результат
`Write(J:6);`	**236**	**___236**
`Write(J:10);`	**1**	**_____1**
`Write(J+J:7);`	**236**	**____472**

Примеры форматированного вывода

Для вещественного выражения

Вывод в поле шириной 18 символов (по умолчанию) десятичного представления величины **R** в формате с плавающей точкой.

Оператор вывода	Значение переменной	Результат
`Write (R);`	**715.432**	**__7.1543200000E+02**
`Write (R);`	**-1.919E+01**	**_-1.9190000000E+01**
`Write (R/2);`	**567.986**	**__2.8399300000E+02**

Примеры форматированного вывода

Для вещественного выражения

Вывод десятичного представления величины **R** в формате с фиксированной точкой в крайние правые позиции поля заданной шириной.

Причем, после десятичной точки выводится заданное количество цифр (не более 24-х), представляющих дробную часть числа; если их количество равно 0, ни дробная часть, ни десятичная точка не выводятся.

Оператор вывода	Значение переменной	Результат
`Write(R:8:4);`	**511.04**	**511.0400**
`Write(R:7:2);`	**-46.78**	**_-46.78**
`Write(R:9:4);`	**-46.78**	**_-46.7800**

Примеры форматированного вывода

Для выражения символьного типа

Вывод символа **Ch**, начиная с позиции расположения курсора.

Оператор вывода	Значение переменной	Результат
`Write(Ch);`	**'X'**	**X**
`Write(Ch, Ch, Ch);`	**'!'**	**!!!**

Вывод символа **Ch** в крайнюю правую позицию поля заданной ширины.

Оператор вывода	Значение переменной	Результат
`Write(Ch:5);`	**'X'**	**____X**
`Write(Ch:2, Ch:4);`	**'!'**	**_!___!**

Примеры форматированного вывода

Для выражения строкового типа

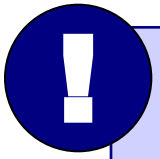
Вывод строки **S**, начиная с позиции расположения курсора.

Оператор вывода	Значение переменной	Результат
`Write(S);`	**‘Иванов‘**	**Иванов**
`Write(S);`	**‘Ведомость № 2‘**	**Ведомость № 2**
`Write(S, S);`	**‘ZZXXX‘**	**ZZXXXZZXXX**

Вывод символа **S** в крайние правые позиции поля заданной ширины.

Оператор вывода	Значение переменной	Результат
`Write(S:10);`	**‘Иванов‘**	**____Иванов**
`Write(S:14);`	**‘Ведомость № 2‘**	**_Ведомость № 2**
`Write(S:6, S:6);`	**‘ZZXXX‘**	**_ZZXXX_ZZXXX**

Форматированный вывод



В случае недостаточной длины вывода число будет автоматически округлено.

Например:

Оператор форматного вывода	Результат вывода на экран
`write (125.2367:10);`	_1.3E+0002
`write (125.2367:11);`	_1.25E+0002
`write (125.2367:12);`	_1.252E+0002
`write (125.2367:13);`	_1.2524E+0002
`write (125.2367:14);`	_1.25237E+0002
`write (125.2367:15);`	_1.252367E+0002
`write (125.2367:16);`	_1.2523670E+0002

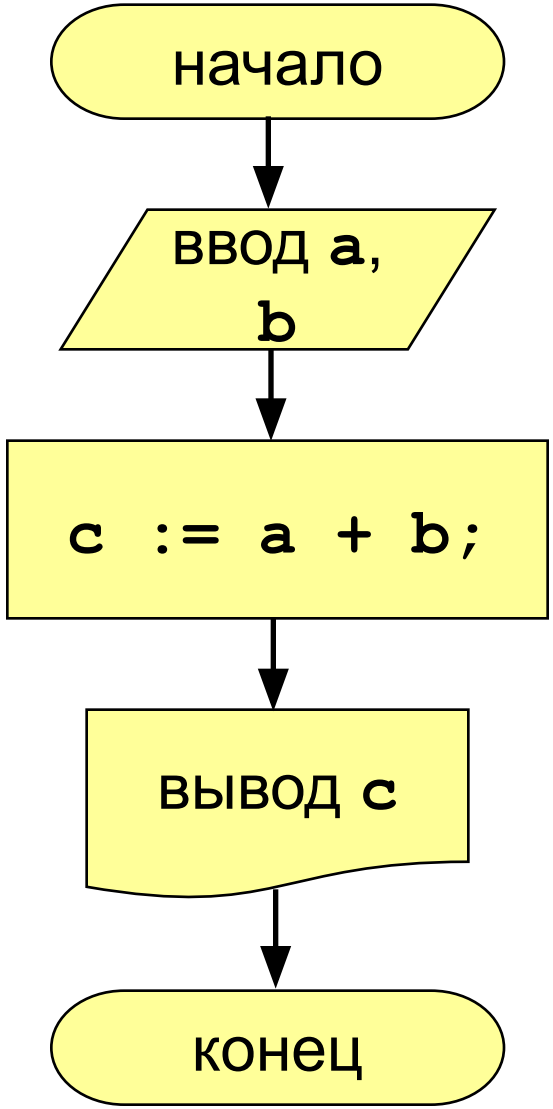
Пример 1. Сложение двух чисел

Задача. Ввести два целых числа и вывести на экран их сумму.

Простейшее решение:

```
program qq;
var a, b, c: integer;
begin
  read ( a, b );
  c := a + b;
  writeln ( c );
end.
```

Блок-схема линейного алгоритма



блок «начало»

блок «ввод»

блок «процесс»

блок «вывод»

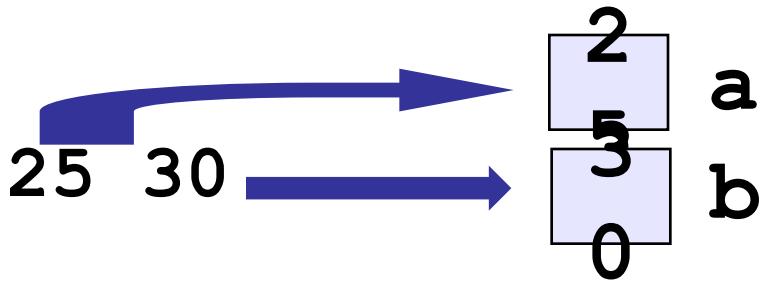
блок «конец»

Оператор ввода

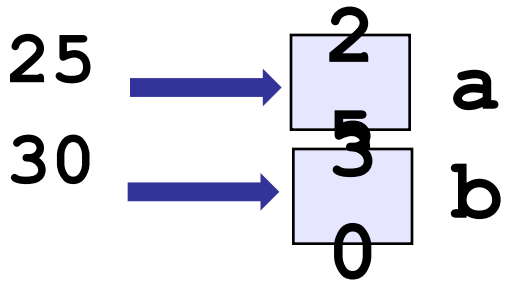
```
read ( a );      { ввод значения переменной a}

read ( a, b ); { ввод значений переменных a
                 и b}
```

Как вводить два числа?

через пробел:



через *Enter*:



Оператор вывода

```
write ( a );     { вывод значения
                   переменной a}

writeln ( a );   { вывод значения
                   переменной a и переход
                   на новую строчку}

writeln ('Введите a'); {вывод текста}

writeln ( 'Ответ: ', c );    { вывод
          текста и значения переменной c}

writeln ( a, '+', b, '=', c );
```

Полное решение

```
program qq;
var a, b, c: integer;
begin
  writeln('Введите два целых числа');
  read ( a, b );
  c := a + b;
  writeln ( a, '+', b, '=', c );
end.
```

компьютер

Протокол:

```
Введите два целых числа
25 30
25+30=55
```

пользователь

7. Метки и безусловный переход

Метки

Метка помечает какое-либо место в тексте *программы*.

Метками могут быть числа от 0 до 9999 или *идентификаторы*, которые в этом случае уже нельзя использовать для каких-либо иных нужд.

Все *метки* должны быть описаны в специальном разделе ***label***:

```
label <список_всех_меток_через_запятую>;
```

Меткой может быть помечен любой оператор программы:

```
<метка>: <оператор>;
```

Любая метка может встретиться в тексте программы только один раз.

Оператор безусловного перехода

Используются *метки* только *операторами* безусловного перехода goto:

```
goto <метка>;
```

Это означает, что сразу после *оператора* **goto** будет выполнен не следующий за ним оператор (как это происходит в обычном случае), а тот *оператор*, который помечен соответствующей *меткой*.

В принципе, передавать управление можно вперед и назад по тексту программы, внутрь составных операторов и наружу и т.п.

Исключением являются только процедуры и функции:

- внутрь них и наружу безусловные переходы невозможны.

Пример

```
. . .
Label  Metka1,  Metka2;
. . .
   Goto  Metka1;
. . .
Metka1: Writeln (Summa);
   Goto  Metka2;
. . .
Metka2:
End.
```