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, ваша *программа* должна быть оформлена в полном соответствии с *синтаксисом* (правилами построения программ) языка Паскаль.

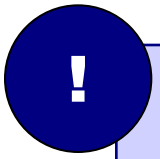
Структура программы

```
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

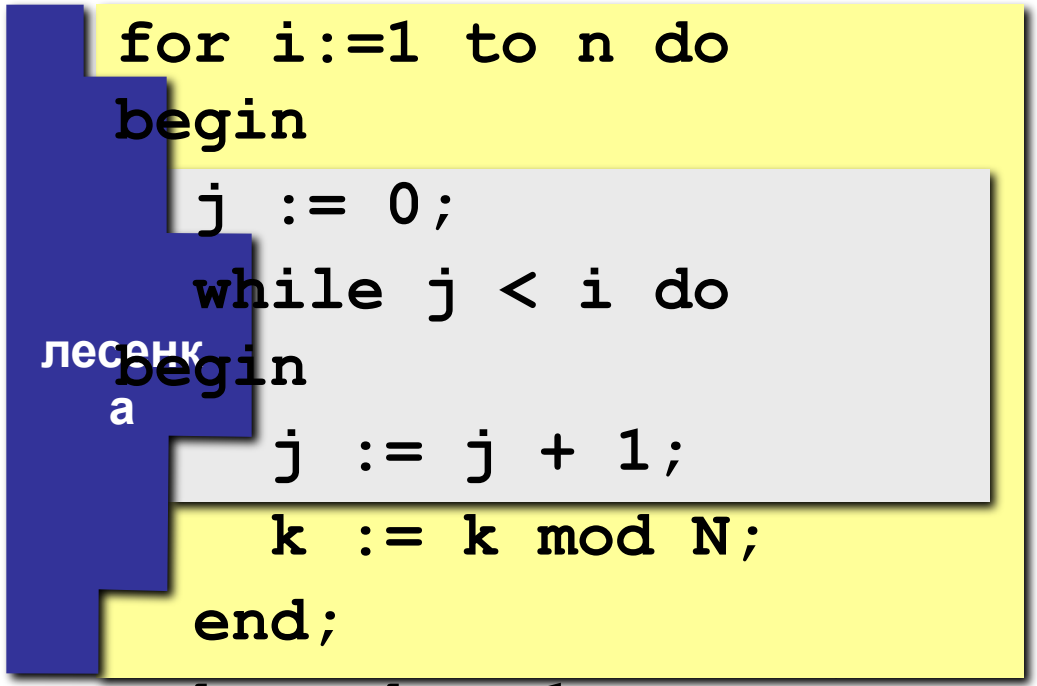
Шапка – комментарий в начале процедур и функций.

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

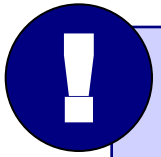
Оформление текста программы

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;
```



- 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.

Идентификаторы

Какие имена правильные?

AXby	R&B
4Wheel	Вася
“PesBarbos”	TU154
[QuQu]	_ABBA
A+B	Koren uravneniya
X1_X2	Koren

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 (см. сл. тему);
- любая последовательность целых чисел либо символов через запятую, обрамленная квадратными скобками, воспринимается как множество (см. сл. тему).

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 (базовому) 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

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

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;
```

Примеры типизированных констант других типов будем приводить по мере изучения соответствующих типов данных.

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 четыре класса:

1) объясняющие назначение программы;
2) поясняющие смысл идентификаторов констант и переменных;
3) поясняющие смысл идентификаторов констант и переменных;
4) 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.

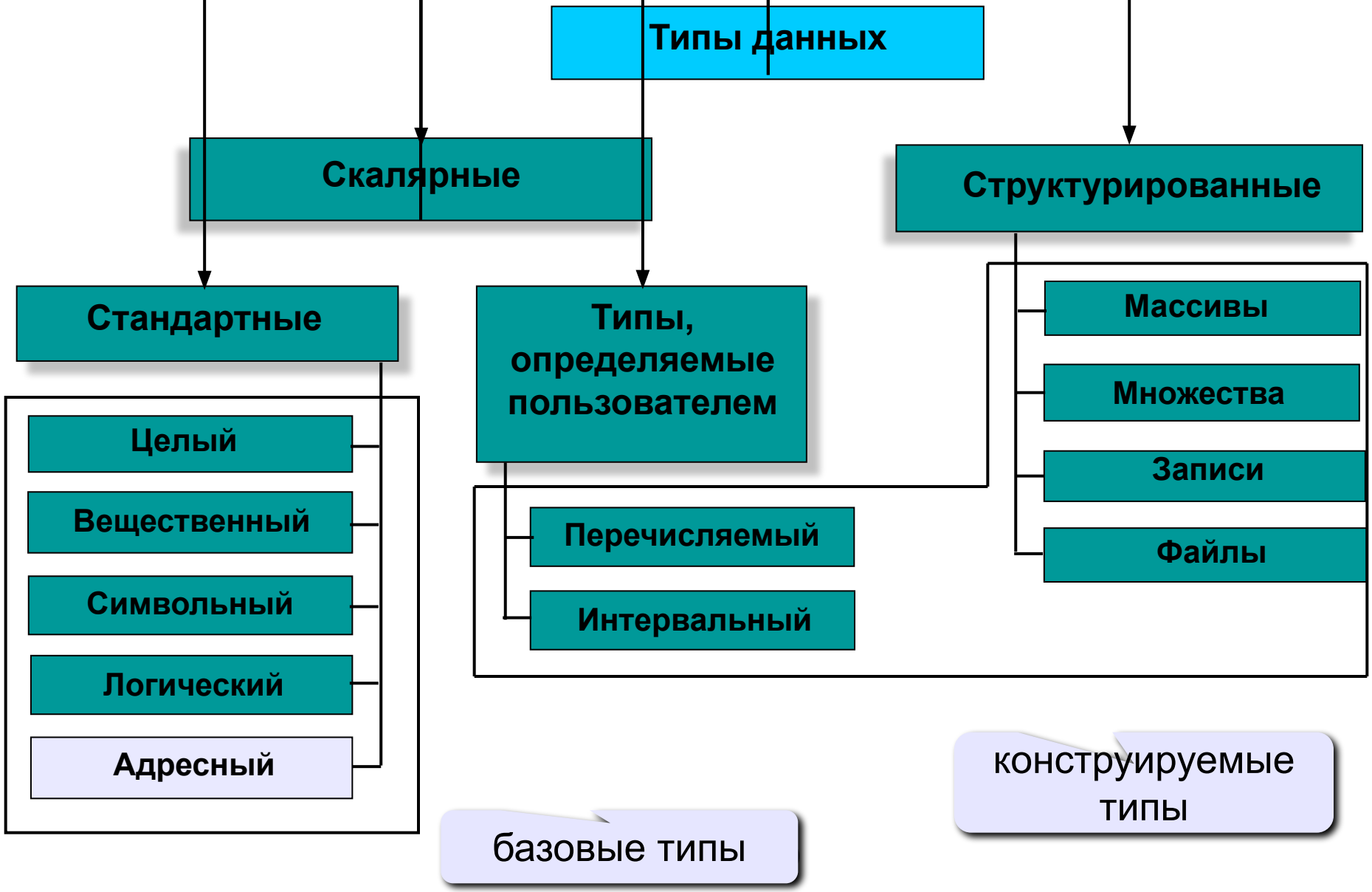
Классификация типов данных

Для удобства программистов существует множество стандартных типов данных и плюс к тому возможность создавать новые типы.

Конструируя новые типы данных на основе уже имеющихся (стандартных или опять-таки определенных самим программистом), нужно помнить, что любое здание должно строиться на хорошем фундаменте. Поэтому сейчас мы и поговорим об этом "фундаменте".

На основании ***базовых типов данных*** строятся все остальные типы языка Паскаль, которые так и называются: ***конструируемые***.

Классификация типов данных



Формат описания типа

Типы данных, конструируемые программистом, описываются в разделе ***type*** по следующему шаблону:

```
Type  <Имя типа> = <описание типа>;
```

Например:

```
type lat_bukvy = 'a'..'z','A'..'Z';
```

Базовые типы данных являются стандартными, поэтому нет нужды описывать их в разделе ***type***. Однако при желании это тоже можно сделать, например, дав длинным определениям короткие имена.

Скажем, введя новый тип данных

```
type int = integer;
```

можно немного сократить текст программы.

Стандартные конструируемые типы также можно не описывать в разделе ***type***.

Однако в некоторых случаях это все равно приходится делать из-за требований синтаксиса. Например, в списке параметров процедур или функций конструкторы типов использовать нельзя (см. лекции далее).

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**
Формат хранения в оперативной памяти: **1 байт без знака**

Целочисленный тип данных *WORD*

Допустимые значения: **от 0 до 65535**
Пример: **5 58 43467**
Формат хранения в оперативной памяти: **2 байта без знака**

Целые беззнаковые числа

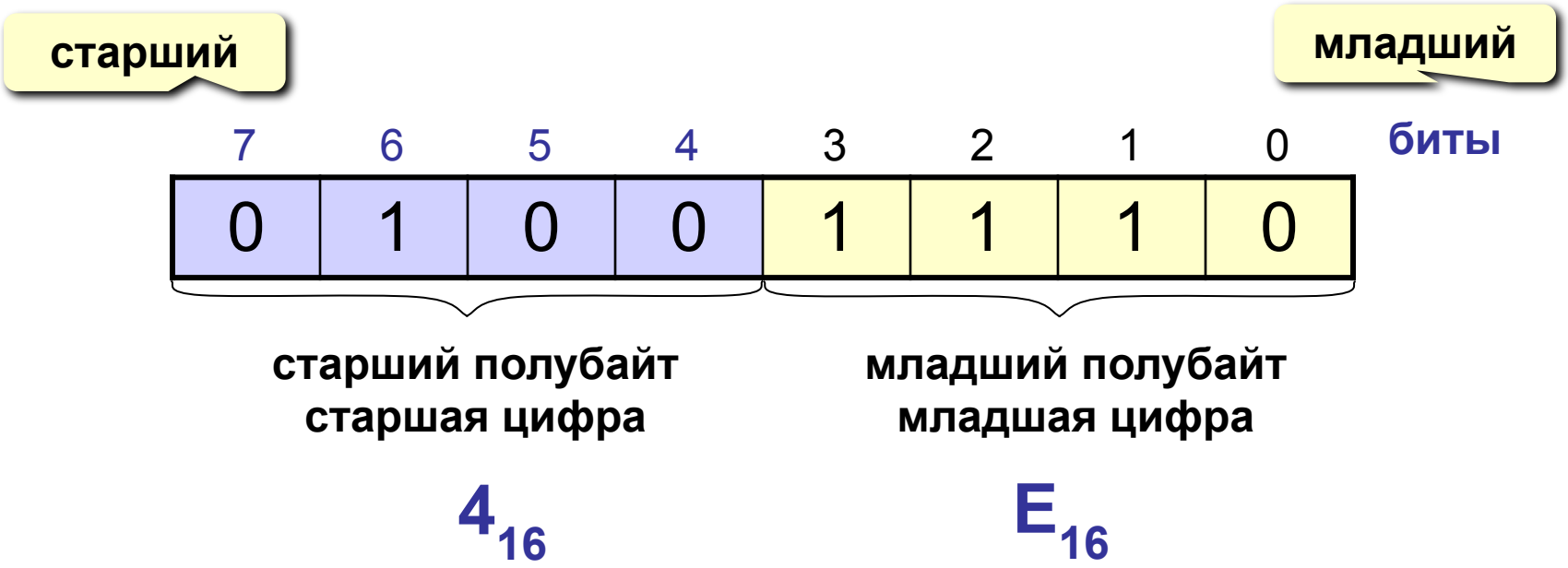
Беззнаковые данные – не могут быть отрицательными.

Байт (символ)

память: **1 байт = 8 бит**

диапазон значений **0…255,** **$0…FF_{16} = 2^8 - 1$**

Паскаль: *byte*



$$1001110_2 = 4E_{16}$$

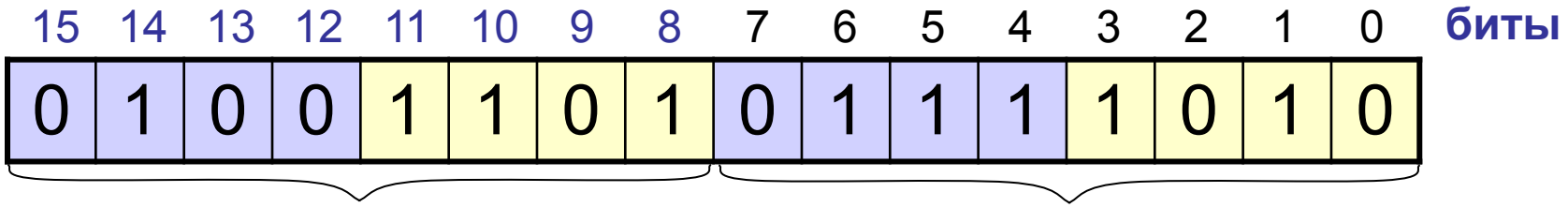
Целые беззнаковые числа

Целое без знака

память: **2 байта = 16 бит**

диапазон значений **0…65535,** **$0\ldots FFFF_{16} = 2^{16}-1$**

Паскаль: *word*



$4D_{16}$ **$7A_{16}$**

$1001101011110 10_2 = 4D7A_{16}$

Целочисленные типы данных

Целочисленный тип данных *SHORTINT*

Допустимые значения: **от –128 до 127**

Пример: **-5 0 58**

Формат хранения в оперативной памяти: **1 байт со знаком**

Целочисленный тип данных *INTEGER*

Допустимые значения: **от –32768 до 32767**

Пример: **5 –58 0 10000 –32768**

Формат хранения в оперативной памяти: **2 байта со знаком**

Целочисленные типы данных

Целочисленный тип данных *LONGINT*

Допустимые значения: **от –2147483648 до 2147473647**
Пример: **5 –3345550 3345550 0**
Формат хранения в оперативной памяти: 4 **байта со знаком**

Целые числа со знаком

Сколько места требуется для хранения знака?

Старший (знаковый) бит числа определяет его знак. Если он равен 0, число положительное, если 1, то отрицательное.

«-1» – это такое число, которое при сложении с 1 даст 0.

не помещается в 1 байт!

1 байт:

$FF_{16} + 1 = 1\,0\,0_{16}$

2 байта: $FFFF_{16} + 1 = 1\,0\,0\,0\,0_{16}$

4 байта: $FFFFFFFF_{16} + 1 = 1\,0\,0\,0\,0\,0\,0\,0\,0_{16}$

Двоичный дополнительный код

Задача: представить отрицательное число **(–a)** в двоичном дополнительном коде.

Решение:

1. Перевести число **a–1** в **двоичную систему**.
2. Записать результат **в разрядную сетку** с нужным числом разрядов.
3. Заменить все «0» на «1» и наоборот (***инверсия***).

Пример: **(– a) = – 78**, сетка 8 бит

4. **a – 1 = 77 = 1001101_2**
5.
6.

0	1	0	0	1	1	0	1

1	0	1	1	0	0	1	0

= – 78

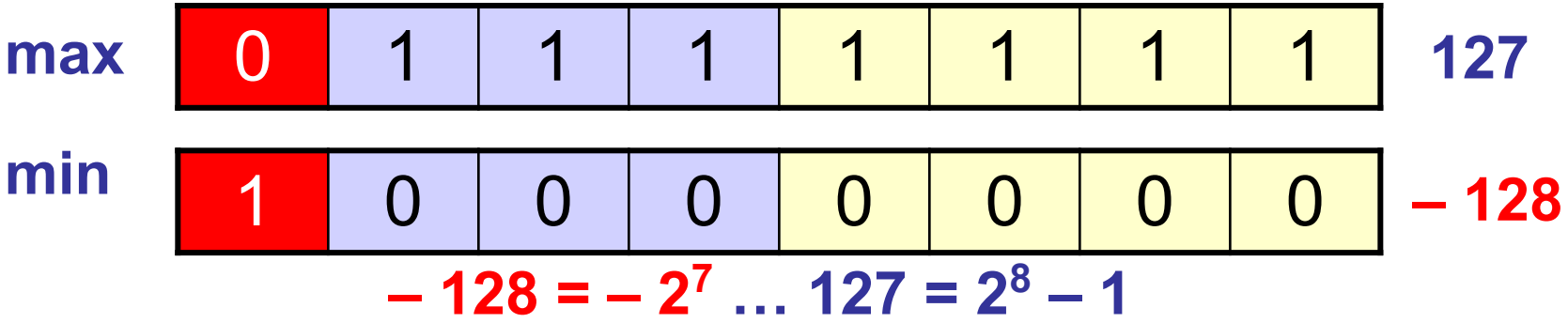
знаковый бит

Целые числа со знаком

Байт (символ) со знаком

память: **1 байт = 8 бит**

диапазон значений:



– 128 = – 2^7 … 127 = 2^8 – 1

Паскаль: *shortint*

можно работать с отрицательными числами

уменьшился диапазон положительных чисел

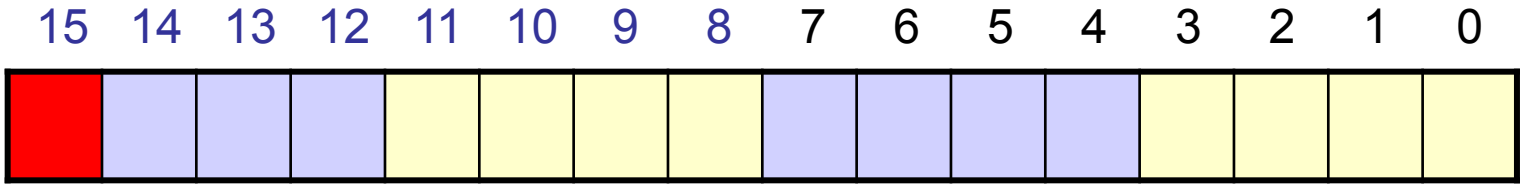
Целые числа со знаком

Слово со знаком

память: **2 байта = 16 бит**

диапазон значений

– 32768 … 32767



Паскаль: *integer*

Двойное слово со знаком

память – **4 байта**

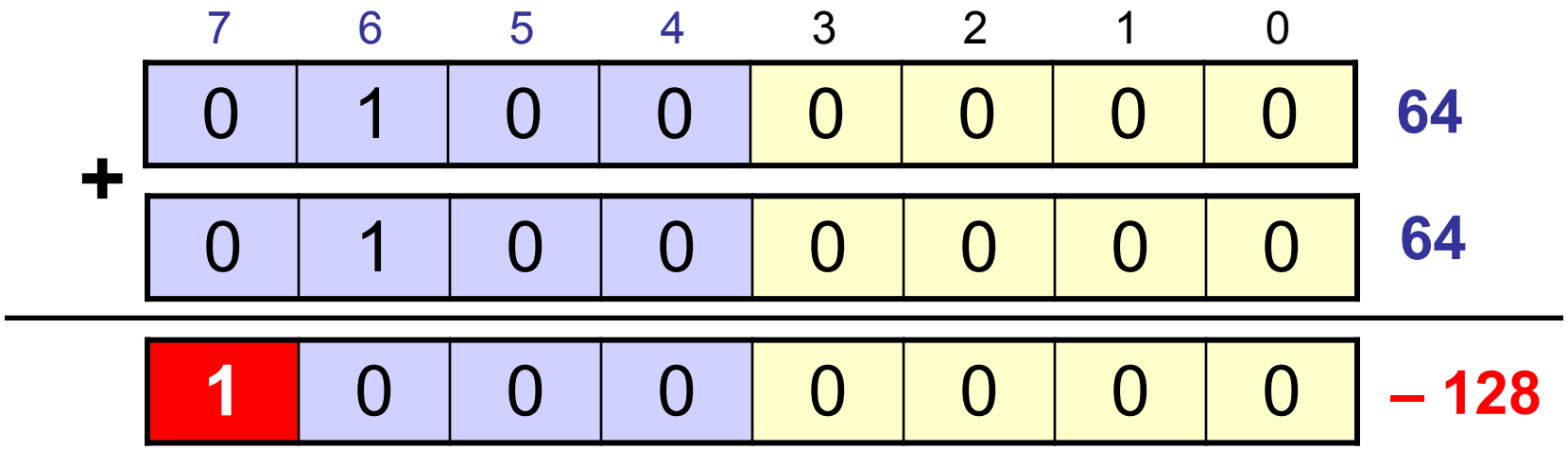
диапазон значений

$-2^{31} \ldots 2^{31}-1$

Паскаль: *longint*

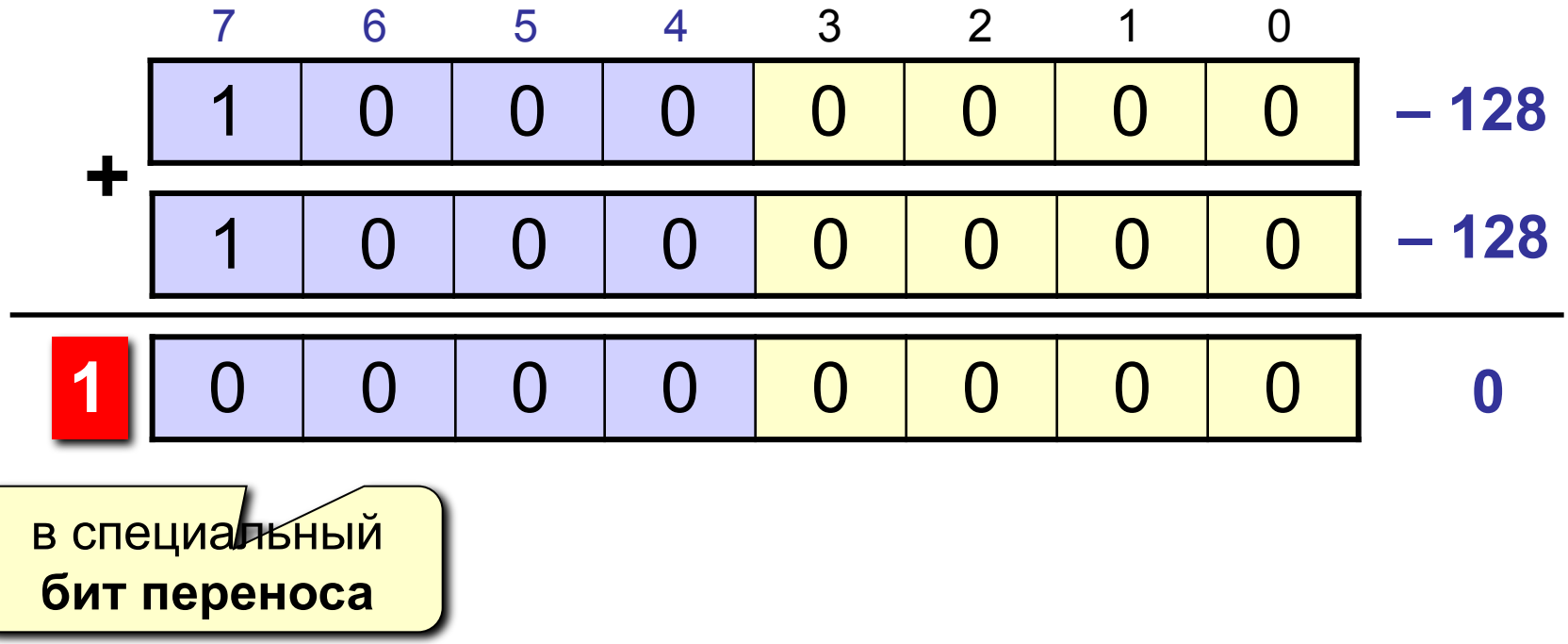
Ошибки

Переполнение разрядной сетки: в результате сложения больших положительных чисел получается отрицательное (**перенос в знаковый бит**).



Ошибки

Перенос: при сложении больших (по модулю) отрицательных чисел получается положительное (перенос **за границы разрядной сетки**).



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 8 = 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**
Формат хранения в оперативной памяти: **6 байт, в том числе 11 цифр для мантиссы**

Вещественный тип данных *SINGLE*

Допустимые значения: **от 1.5e–45 до 3.4e+38**
Формат хранения в оперативной памяти: **4 байта, в том числе 7 цифр для мантиссы**

Вещественные типы данных

Вещественный тип данных *DOUBLE*

Допустимые значения: **от 5.0e–324 до 1.7e+308**
Формат хранения в оперативной памяти: **8 байт, в том числе 15 цифр для мантиссы**

Вещественный тип данных *EXTENDED*

Допустимые значения: **от 3.4e–4932 до 1.1e+4932**
Формат хранения в оперативной памяти: **10 байт, в том числе 19 цифр для мантиссы**

Вещественный тип данных *COMP*

Допустимые значения: **от –9.2e+18 до 9.2e+18**
Формат хранения в оперативной памяти: **8 байт, в том числе 19 цифр для мантиссы**

Нормализация двоичных чисел

$$X = s \cdot M \cdot 2^e$$

s – **знак** (1 или -1)

M – **мантисса**, $M = 0$ **или** $1 \le M < 2$

e – **порядок**

Пример:

знак

мантисса

порядок

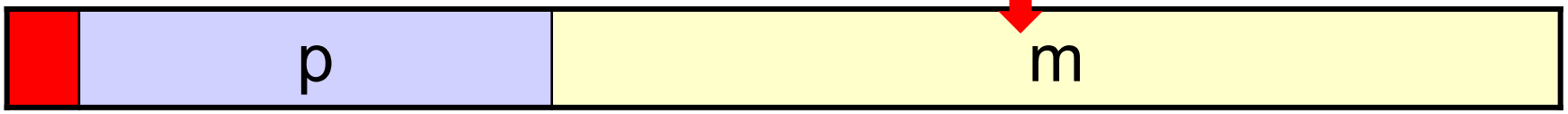
$$15{,}625 = 1111{,}101_2 = 1 \cdot 1{,}111101_2 \cdot 2^3$$

Нормализованные числа в памяти

IEEE Standard for Binary Floating-Point Arithmetic (IEEE 754)

$$15{,}625 = 1 \cdot 1{,}111101_2 \cdot 2^3$$

$s = 1$ $e = 3$ $M = 1{,}111101_2$



Порядок со сдвигом:
$p = e + E$ (сдвиг)

Дробная часть мантиссы:
$m = M - 1$

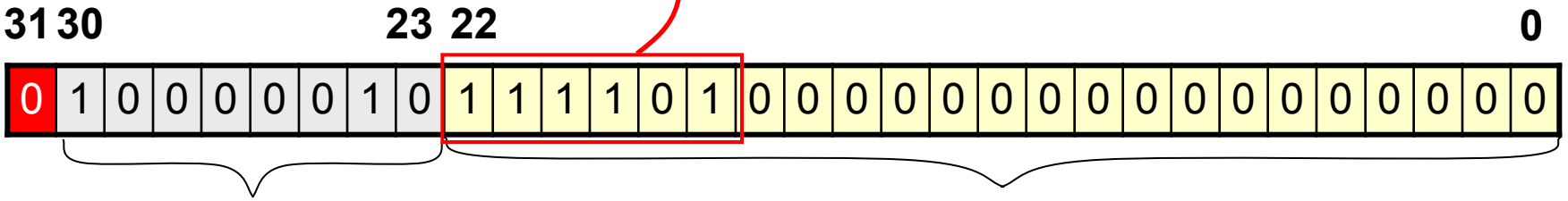
Знаковый бит:
0, если s = 1
1, если s = – 1

Целая часть M всегда 1, поэтому не хранится в памяти!

Вещественные числа в памяти

$15{,}625 = 1{,}111101_2 \cdot 2^3$

4 байта = 32 бита



$p = e + 127 = 130 = 10000010_2$

$m = M - 1 = 0{,}111101_2$

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

Таблицы истинности для операций:

A	*B*	*Not A*	*A And B*	*A Or B*	*A Xor B*
False	*False*	*True*	*False*	*False*	*False*
False	*True*		*False*	*True*	*True*
True	*False*	*False*	*False*	*True*	*True*
True	*True*		*True*	*True*	*False*

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 ****

Формат хранения в оперативной памяти: **1 байт**

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!**

Формат хранения в оперативной памяти: **1 байт на каждый символ строки + 1 байт под длину строки**

Кодирование символов

Текстовый файл

- на экране (символы)
- в памяти – двоичные коды

1000001_2	1000010_2	1000011_2	1000100_2
65	66	67	68

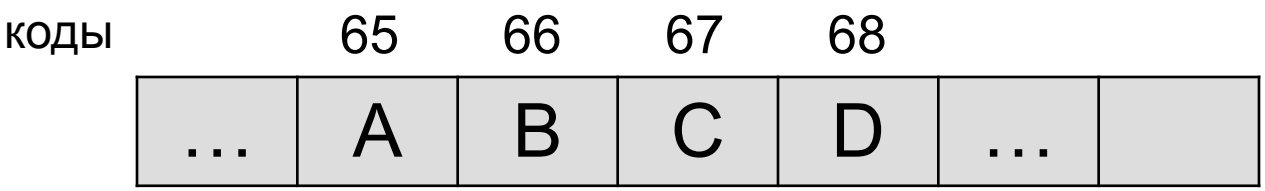
В файле хранятся не изображения символов, а их числовые коды в двоичной системе!

Кодирование символов

1. **Сколько символов** надо использовать одновременно? **256** или 65536 (UNICODE)
2. **Сколько места** надо выделить **на символ**:

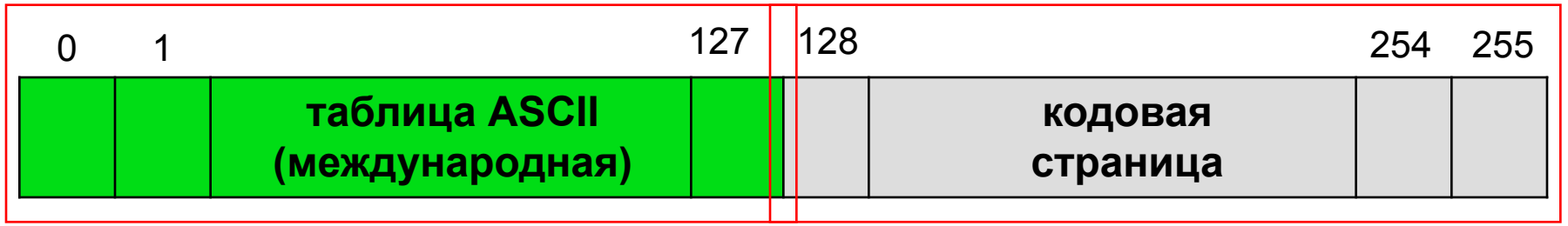
256 = 2^8 ⇒ **8 бит на символ**

1. Выбрать **256 любых символов** (или 65536) - **алфавит**.
2. Каждому символу – **уникальный код 0..255** (или 0..65535). Таблица символов:



3. Коды – в **двоичную систему**.

Кодировка 1 байт на символ



ASCII = *American Standard Code for Information Interchange*

0-31 управляющие символы:
7 – звонок, 10 – новая строка, 13 – возврат каретки, 27 – Esc.
32 пробел
знаки препинания: `. , : ; ! ?`
специальные знаки: `+ - * / () {} []`
48-57 цифры **0..9**
65-90 заглавные латинские буквы **A-Z**
97-122 строчные латинские буквы **a-z**

Кодовая страница (расширенная таблица ASCII)

для русского языка:

CP-866 для системы *MS DOS*
CP-1251 для системы *Windows* (Интернет)
КОИ8-R для системы *UNIX* (Интернет)

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);
```

Порядковые типы данных

Среди базовых типов данных особо выделяются ***порядковые типы***. Такое название можно обосновать двояко:

1) Каждому элементу порядкового типа может быть сопоставлен уникальный (порядковый) номер. Нумерация значений начинается с нуля. Исключение - типы данных **shortint**, **integer** и **longint**. Их нумерация совпадает со значениями элементов.
2) Кроме того, на элементах любого порядкового типа определен порядок (в математическом смысле этого слова), который напрямую зависит от нумерации. Таким образом, для любых двух элементов порядкового типа можно точно сказать, который из них меньше, а который - больше.

Порядковые типы данных

Только для величин порядковых типов определены следующие стандартные функции и процедуры:

1) Функция ***ord(x)*** возвращает порядковый номер значения переменной ***x*** (относительно того типа, к которому принадлежит переменная ***x***).
2) Функция ***pred(x)*** возвращает значение, предшествующее ***x*** (к первому элементу типа неприменима).
3) Функция ***succ(x)*** возвращает значение, следующее за ***x*** (к последнему элементу типа неприменима).
4) Процедура ***inc(x)*** возвращает значение, следующее за ***x*** (для арифметических типов данных это эквивалентно оператору ***x:=x+1***).

Порядковые типы данных

5) Процедура ***inc(x, k)*** возвращает ***k***-е значение, следующее за ***x*** (для арифметических типов данных это эквивалентно оператору ***x:=x+k***).
6) Процедура ***dec(x)*** возвращает значение, предшествующее ***x*** (для арифметических типов данных это эквивалентно оператору ***x:=x-1***).
7) Процедура ***dec(x, k)*** возвращает ***k***-е значение, предшествующее ***x*** (для арифметических типов данных это эквивалентно оператору ***x:=x-k***).

Порядковые типы данных

На первый взгляд кажется, будто результат применения процедуры ***inc(x)*** полностью совпадает с результатом применения функции ***succ(x)***. Однако разница между ними проявляется на границах допустимого диапазона.

Функция ***succ(x)*** неприменима к максимальному элементу типа, а вот процедура ***inc(x)*** не выдаст никакой ошибки, но, действуя по правилам машинного сложения, прибавит очередную единицу к номеру элемента.

Номер, конечно же, выйдет за пределы диапазона и за счет усечения превратится в номер минимального значения диапазона.

Получается, что процедуры inc() и ***dec()*** воспринимают любой порядковый тип словно бы "замкнутым в кольцо": сразу после последнего вновь идет первое значение.

Порядковые типы данных

Поясним все сказанное на примере. Для типа данных

```
type sixteen = 0..15;
```

попытка прибавить 1 к числу 15 приведет к следующему результату:

$$\begin{array}{r} 1111 \\ +\quad 1 \\ \hline 10000 \end{array}$$

Начальная единица будет отсечена, и потому получится, что ***inc(15)=0***.

Аналогичная ситуация на нижней границе допустимого диапазона произвольного порядкового типа данных наблюдается для процедуры ***dec(x)*** и функции ***pred(x)***:

```
dec(min_element)= max_element
```

Типы данных, относящиеся к порядковым

Опишем теперь порядковые типы данных более подробно.

1) Логический тип ***boolean*** имеет два значения: ***false*** и ***true***, и для них выполняются следующие равенства:
   ord(false)=0, ***ord(true)=1***,
   false<true,
   pred(true)=false, ***succ(false)=true***,
   inc(true)=false, ***inc(false)=true***,
   dec(true)=false, ***dec(false)=true***.
2) В символьный тип ***char*** входит 256 символов расширенной таблицы ASCII (например, 'a', 'b', 'я', '7', '#'). Номер символа, возвращаемый функцией ***ord()***, совпадает с номером этого символа в таблице ASCII.
3) Целочисленные типы данных.

Типы данных, относящиеся к порядковым

4) Перечисляемые типы данных задаются в разделе ***type*** явным перечислением их элементов.

   Например:

```
type week =(sun,mon,tue,wed,thu,fri,sat)
           0 1  2    3    4    5  6
```

   Для этого типа данных:

   inc(sat) = sun, ***dec(sun) = sat***.

5) Интервальные типы данных задаются только границами своего диапазона.

   Например:

```
type month = 1..12;
     budni = mon..fri;
```

Типы данных, относящиеся к порядковым

6) Программист может создавать и собственные типы данных, являющиеся комбинацией нескольких стандартных типов.

Например:

```
type valid_for_identifiers=
  'a'..'z','A'..'Z','_','0'..'9';
```

Этот тип состоит из объединения нескольких интервалов, причем в данном случае изменен порядок латинских букв: если в стандартном типе ***char*** 'A' < 'a', то здесь наоборот - 'a' < 'A'.

Для величин этого типа выполняются следующие равенства:

```
inc('z')='A'; dec('0')='_',
pred('9')='8';   ord('b')= 2.
```

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.

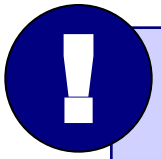
- ***Chr(X)*** – преобразование ASCII-кода в символ. Аргумент функции – целого типа от 0 до 255. Результатом – символ, соответствующий данному коду.
- ***High(X)*** – получение максимального значения величины. Аргумент функции – параметр или идентификатор порядкового типа, типа-массива, типа-строки. 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**. Алгоритм вычисления a^b для любых значений **a** будет разобран позднее.

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

$(x-3)^7$

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

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 числе

`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

5. 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

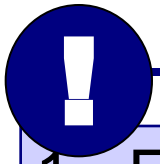
```
odd(round(x/abs(x)))
```

"многоэтажное" выражение

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

"многоэтажное" выражение с использованием массива

Порядок вычислений



1. Если в выражении *расставлены скобки*, то чем меньше глубина вложенности скобок, тем позже вычисляется заключенная в них операция.
2. Если же *скобок нет*, то сначала вычисляются значения операций с более высоким приоритетом, затем – с менее высоким.
3. Несколько подряд идущих *операций одного* ***приоритета*** вычисляются в последовательности "слева направо".
4. *Вызов любой функции* имеет более высокий приоритет, чем все внешние относительно этого вызова операции. Выражения, являющиеся аргументами вызываемой функции, вычисляются в момент вызова.

Приоритеты всех операций языка Паскаль

	Операции	Приоритет
Унарные операции	**+**, **-**, ***not***, **@**, **^**, **#**	Первый (высший)
Операции, эквивалентные умножению	*, /, ***div***, ***mod***, ***and***, ***shl***, ***shr***	Второй
Операции, эквивалентные сложению	**+**, **-**, ***or***, ***xor***	Третий
Операции сравнения	**=**, **<>**, **>**, **<**, **<=**, **>=**, ***in***	Четвертый

Примеры выражений для целых чисел с указанием последовательности вычислений

```
a + b * c / d
```

3 1 2

результат принадлежит к вещественному типу данных

```
a * not b or c * d = 0
```

2 1 4 3 5

результат принадлежит к логическому типу данных

```
-min(a + b, 0) * (a + 1)
```

3 2 1 5 4

результат принадлежит к целочисленному типу данных

Совместимость типов данных

В общем случае при выполнении арифметических (и любых других) операций компилятору требуется, чтобы типы операндов совпадали:

- нельзя, например, сложить массив и множество,
- нельзя передать вещественное число функции, ожидающей целый аргумент, и т.п.

В то же время, любая процедура или функция, написанная в расчете на вещественные значения, сможет работать и с целыми числами.

Правила, по которым различные типы данных считаются взаимозаменяемыми, приведены ниже.

Эквивалентность

Эквивалентность - это наиболее высокий уровень соответствия типов. Она требуется при действиях с указателями (см. Тему далее), а также при вызовах подпрограмм.

"А как же тогда быть с оговоркой, сделанной двумя абзацами выше?" - спросите вы.

Мы не станем сейчас описывать механизм передачи аргументов процедурам и функциям, поясним лишь, что эквивалентность типов требуется только для параметров-переменных (см. Тему далее).

Эквивалентность

Итак, два типа - **T1** и **T2** - будут эквивалентными, если верен хотя бы один вариант из перечисленных ниже:

- **T1** и **T2** совпадают;
- **T1** и **T2** определены в одном объявлении типа;
- **T1** эквивалентен некоторому типу **T3**, который эквивалентен типу **T2**.

Эквивалентность

Пример.

```
type  T2 = T1;
      T3 = T1;
      T4, T5 = T2;
```

Здесь эквивалентными будут

- ✔ T1 и T2;
- ✔ T1 и T3;
- ✔ T1 и T4;
- ✔ T1 и T5;
- ✔ T4 и T5.

А вот T2 и T3 - не эквивалентны!

Итак, два типа - **T1** и **T2** - будут ***эквивалентными***, если верен хотя бы один вариант из перечисленных ниже:

- **T1** и **T2** совпадают;
- **T1** и **T2** определены в одном объявлении типа;
- **T1** эквивалентен некоторому типу **T3**, который эквивалентен типу **T2**.

Совместимость

Совместимость типов требуется при конструировании выражений, а также при вызовах подпрограмм (для параметров-значений).

Совместимость означает, что для переменных этих типов возможна операция присваивания - хотя во время этой операции присваиваемое значение может измениться:

- произойдет ***неявное приведение типов данных*** (см. п. "Приведение типов данных" ниже).

Совместимость

Два типа **T1** и **T2** будут ***совместимыми***, если верен хотя бы один вариант из перечисленных ниже:

- **T1** и **T2** эквивалентны (в том числе совпадают);
- **T1** и **T2** - оба целочисленные или оба вещественные;
- **T1** и **T2** являются подмножествами одного типа;
- **T1** является некоторым подмножеством **T2**;
- **T1** - строка, а **T2** - символ (см. Тему далее);
- **T1** - это тип ***pointer***, а **T2** - типизированный указатель (см. Тему далее);
- **T1** и **T2** - оба процедурные, с одинаковым количеством попарно эквивалентных параметров, а для функций - с эквивалентными типами результатов (см. Тему далее).

Совместимость по присваиванию

В отличие от простой совместимости, совместимость по присваиванию гарантирует, что в тех случаях, когда производится какое-либо присваивание (используется запись вида ***a:=b;*** или происходит передача значений в подпрограмму или из нее и т.п.), не произойдет никаких изменений присваиваемого значения.

Два типа данных **T1** и **T2** называются ***совместимыми по присваиванию***, если выполняется хотя бы один вариант из перечисленных ниже:

- **T1** и **T2** эквивалентны, но не файлы;
- **T1** и **T2** совместимы, причем **T2** – некоторое подмножество в **T1**;
- **T1** – вещественный тип, а **T2** – целый.

Неявное приведение типов данных

Как мы упомянули в п. "Арифметические операции" выше, тип результата арифметических операций (а следовательно, и выражений) может отличаться от типов исходных операндов.

Например, при "дробном" делении (*/*) одного целого числа на другое целое в ответе все равно получается вещественное.

Такое изменение типа данных называется ***неявным приведением типов***.

Если в некоторой операции присваивания участвуют два типа данных совместимых, но не совместимых по присваиванию, то тип присваиваемого выражения автоматически заменяется на подходящий. Это тоже ***неявное приведение***. Причем в этих случаях могут возникать изменения значений.

Неявное приведение типов данных

Т.е., если выполнить такую последовательность операторов

```
a:= 10;    {a: byte}
a:= -a;
writeln(a);
```

то на экране мы увидим не -10, а 246 (246 = 256 - 10).

Неявным образом осуществляется и приведение при несоответствии типов переменной-счетчика и границ в циклах ***for*** (см. след. Тему).

Неявное приведение типов данных

Неявное приведение типов данных можно отключить, если указать директиву компилятора ***{$R+}***, которая принуждает компилятор всегда проверять границы и диапазоны.

Если эта директива включена, то во всех ситуациях, в которых по умолчанию достаточно совместимости типов данных, будет необходима их эквивалентность.

По умолчанию такая проверка отключена, поэтому во всем дальнейшем изложении (если, конечно, явно не оговорено противное) будем считать, что эта директива находится в выключенном состоянии ***{$R-}***.

Явное приведение типов данных

Тип значения можно изменить и ***явным*** способом: просто указав новый тип выражения, например:

a:= byte(b).

В этом случае переменной ***a*** будет присвоено значение, полученное новой интерпретацией значения переменной ***b***.

Т.е., если ***b*** имеет тип ***shortint*** и значение **-23**, то в a запишется **233** (= 256 - 23).

Явное приведение типов данных

Приводить явным образом можно и типы, различающиеся по длине. Тогда значение может измениться в соответствии с новым типом.

Т.е., если преобразовать тип ***longint*** в тип ***integer***, то возможны потери из-за отсечения первых двух байтов исходного числа.

Например, результатом попытки преобразовать число **100 000** к типу ***integer*** станет число **31 072**, а к типу ***word*** - число **34 464**.

Функции, изменяющие тип данных

В заключение приведем список стандартных функций, аргумент и результат которых принадлежат к совершенно различным типам данных:

trunc: ***real*** → ***integer;***

round: ***real*** → ***integer;***

val: ***string*** → ***byte / integer / real;***

chr: ***byte*** → ***char;***

ord: **<*порядковый_тип*>** → ***longint;***

Функции, изменяющие тип данных

Пример. Присвоить целой переменной ***d*** первую цифру из дробной части положительного вещественного числа ***x***. Например, если ***x***=32,597, то ***d***=5.

Решение. **1-й способ.** Решение этой задачи можно получить, если умножить заданное число на 10. В этом случае задача сведется к выделению последней цифры полученного произведения.

Нужно еще учесть, что операция ***mod*** применима к целому числу, поэтому произведение нужно преобразовать к целому виду, например, отбросив дробную часть.

Получаем оператор, решающий задачу:

```
d:= trunc(x*10) mod 10;
```

Функции, изменяющие тип данных

2-й способ. Возможно решение с помощью преобразования числа в строку символов (будет рассмотрено в Теме 5). В этом случае в строке нужно найти позицию точки и выделить следующую за ней цифру.

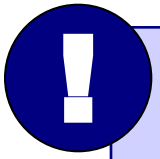
Получаем такой фрагмент:

```
var   x: real;{ заданное число }
      d, cod: integer;  { первая цифра дробной части
                          заданного числа }
   s: string;  { число, представленное
                 в виде строки символов }
begin
   str(x, s);  { преобразовали число в строку
                 32.597 → '32.597' }
Val(copy(s, pos('.', s) + 1, 1), d, cod)
      { выделили нужную цифру }
end.
```

6. 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***.

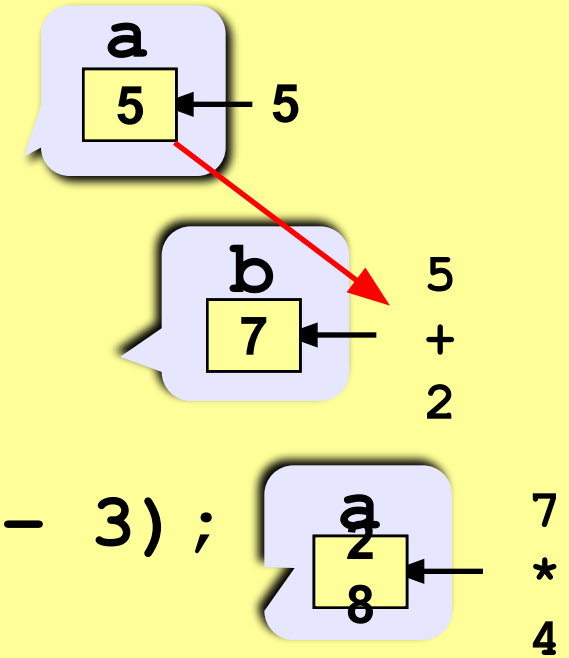
- Условные операторы (***If***, ***Case***).
- Операторы цикла (***Repeat***, ***While***, ***For***).
- Оператор присоединения (***With***).

Как изменить значение переменной?

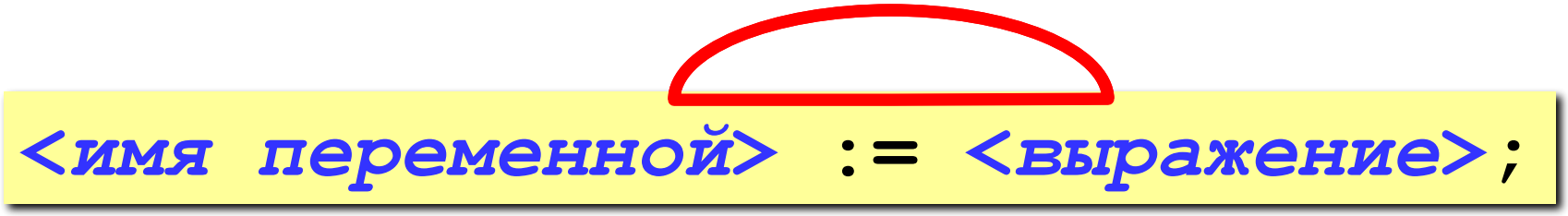
Оператор присваивания служит для изменения значения переменной.

Пример:

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



Оператор присваивания



Арифметическое выражение может включать

- константы
- имена переменных
- знаки арифметических операций:

```
+ -   *   /     div   mod
```

умножение

деление

деление нацело

остаток от деления

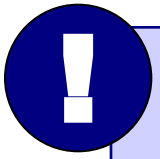
- вызовы функций
- круглые скобки ( )

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;
Letter := Succ(Succ(Letter));
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

Ручная прокрутка программы

```
program qq;
var  a, b: integer;
begin
  a := 5;
  b := a + 2;
  a := (a + 2)*(b - 3);
  b := a div 5;
  a := a mod b;
  a := a + 1;
  b := (a + 14) mod 7;
end.
```

a	b
?	?
5	
	7
28	
	5
3	
4	
	4

Порядок выполнения операций

- вычисление выражений в скобках
- умножение, деление, **div**, **mod** слева направо
- сложение и вычитание слева направо

```
          2  3  5  4   1    7  8   6
z := (5*a*c+3*(c-d))/a*(b-c)/ b;
```

$$z = \frac{5ac + 3(c-d)}{ab}(b-c)$$

$$x = \frac{a^2 + 5c^2 - d(a+b)}{(c+d)(d-2a)}$$

```
          2  6  3  4  7  5   1    12   8  11  10  9
x:=(a*a+5*c*c-d*(a+b))/((c+d)*(d-2*a));
```

7. Ввод и вывод данных

Ввод и вывод: консоль

Как мы уже говорили, любой алгоритм должен быть результативным. В общем случае это означает, что он должен сообщать результат своей работы потребителю: пользователю-человеку или другой программе (например, программе управления принтером).

Мы не будем описывать здесь внутренние автоматические процессы, использующие сигналы непрерывно функционирующих программ, а сосредоточим внимание на взаимодействии программы и человека, т.е. на процессах ввода информации с клавиатуры и вывода ее на экран.

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:byte; 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).

Ввод с консоли

При вводе исходных данных происходит преобразование из внешней формы представления во внутреннюю, определяемую типом переменных.

Типы вводимых значений должны совпадать с типами указанных переменных, иначе возникает ошибка. Поэтому нужно внимательно следить за правильностью вводимых данных.

Вообще, вводить с клавиатуры можно только данные базовых типов: целого, вещественного, символьного (за исключением логического).

Ввод с консоли

Если же программе все-таки необходимо получить с консоли значение для *boolean*-величины, придется действовать более хитро:

- вводить оговоренный символ, а уже на его основе присваивать логической переменной соответствующее значение.

Например:

```
repeat
  writeln('Согласны ли Вы с этим утверждением?');
  writeln('y - да, n - нет');
  readln(c);{c:char}
  case c of
   'y':  b:= true;
   'n':  b:= false;
   else: writeln('Ошибка!');
  end;
until (c='n') or (c='y');
```

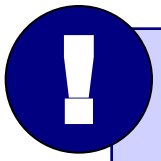
Ввод с консоли

Второе исключение: строки, хотя они и не являются базовым типом, вводить тоже разрешается.

Признаком окончания ввода строки является нажатие клавиши Enter, поэтому все следующие за нею переменные необходимо вводить с новой строчки.

Вывод на консоль

Сделаем одно 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: "лысый" 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 должны иметь тип либо базовый, либо строчный.

Например:

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

Переменные, составляющие список вывода, могут относиться к целому, вещественному, символьному или булевскому типам.

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:20:5);
```

количество позиций, выделяемых под всю переменную

количество позиций, выделяемых под дробную часть числа

Форматированный вывод

Если число длиннее, чем отведенное под него пространство, количество позиций будет автоматически увеличено.

Если же выводимое число короче заданного формата, то спереди к нему припишутся несколько пробелов.

Таким образом можно производить вывод красивыми ровными столбиками, а также следить за тем, чтобы переменные не сливались.

Форматированный вывод

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

```
writeln(a:5, ' ',b, c:20:5);
```

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.

Минимальной длиной для вывода вещественных чисел является 10 (при формате **_x.xE+yyyy**). Первая позиция зарезервирована под знак "**-**".

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 плавающей точкой в крайние правые позиции поля заданной шириной.

Оператор вывода	Значение переменной	Результат
`Write(R:15);`	**511.04**	**5.11040000E+02**
`Write(R:15);`	**-511.04**	**-5.11040000E+02**
`Write(-R:12);`	**46.78**	**-4.67800E+01**

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**

Примеры форматированного вывода

Для выражения логического типа

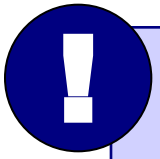
Вывод результата выражения **B** (***True*** или ***False***), начиная с позиции расположения курсора.

Оператор вывода	Значение переменной	Результат
`Write(B);`	**True**	**True**
`Write(B);`	**False**	**False**
`Write(B, not B);`	**True**	**TrueFalse**

Вывод результата выражения **B** (***True*** или ***False***) в крайние правые позицию поля заданной ширины.

Оператор вывода	Значение переменной	Результат
`Write(B:6);`	**True**	**__True**
`Write(B:10);`	**False**	**_____False**
`Write(B:5, not B:8);`	**True**	**_True___False**

Форматированный вывод



В случае недостаточной длины вывода число будет автоматически округлено.

Например (подчеркивание служит для визуализации пробела):

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

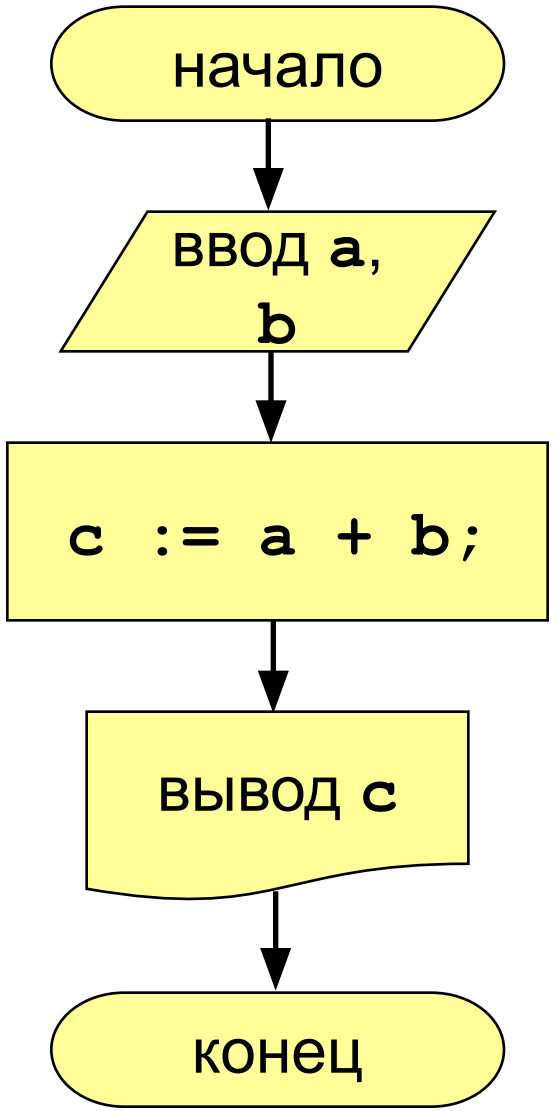
Пример 1. Сложение двух чисел

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}
```

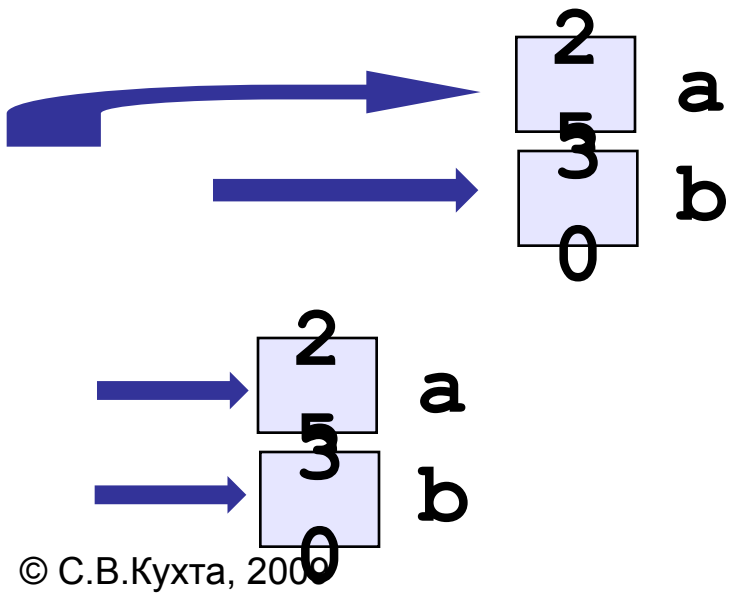
Как вводить два числа?

через пробел:

```
25 30
```

через *Enter*:

```
25
30
```



Оператор вывода

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

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

writeln ( 'Привет!' );  { вывод текста}

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

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

Форматы вывода

```
program qq;
var i: integer;
    x: real;
begin
  i := 15;
  writeln ( '>', i, '<' );
  writeln ( '>', i:5, '<' );
  x := 12.345678;
  writeln ( '>', x, '<' );
  writeln ( '>', x:10, '<' );
  writeln ( '>', x:7:2, '<' );
end.
```

всего символов

```
>15<
>   15<

>1.234568E+001<
> 1.23E+001<
>  12.35<
```

всего символов

в дробной части

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

Пример 2.

Задача. Программа вычисления площади круга, вписанного в треугольник и площади круга, описанного около треугольника.

```
Program Triangle;
Const  Pi = 3.1415926;
   Line ='____________';
Var  a, b, c : Real;
   Sint, Sout : Real;
   S, p : Real;
Begin
{ ввод данных }
Write ('введите стороны
          треугольника a b c
        ');
Readln (a, b, c);
{ вычисления }
p:=(a+b+c)/2;
S:=Sqrt(p*p-a)*(p-b)*(p-c));
Sout:=Pi*sqrt(a*b*c/4/S);
Sint:=Pi*sqrt(2*S/p);

{ печать ответа }
Writeln;
Writeln (Line);
Writeln('Sвпис = ', Sout);
Writeln(Line);
Writeln('Sопис = ', Sint);
Writeln(Line)
End.
```

Пример 3.

Задача. Программа возведения положительного числа “x” в степень”n”, т.е. $y=x^n$. Расчет производится по формуле: $y=e^{n*\ln(x)}$.

```
Program   N_2;
   {возведение положительного числа в степень}
Var   n, x, y: real;
BEGIN
   Writeln ('Введите значение аргумента x>0');
   Readln (x);
   Writeln ('Введите значение показателя степени n=');
   Readln (n);
   y:= exp( n * ln(x));
   Writeln (' Результат y=', y:10:6);
   Writeln (' Нажмите Enter');
   readln;     {задержка экрана до нажатия Enter}
END.
```

8. 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.

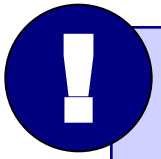
Исключением являются только процедуры и функции (см. соответствующую Тему):

- внутрь них и наружу безусловные переходы невозможны.

Оператор безусловного перехода

Оператор ***Goto*** следует применять как можно реже, так как его частое употребление резко усложняет понимание логики программы. В крайнем случае он может использоваться для преждевременного выхода из операторов повтора.

Вообще же использование безусловных переходов в структурном и надежном программировании считается "дурным тоном". Поэтому настоятельно советуем:



Воздерживаться от употребления *операторов* ***goto***.

Язык Паскаль обладает достаточным количеством структурных конструкций и возможностей, позволяющих достичь хороших результатов надежными средствами.

Пример

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