



Блез Паскаль  
1623-1662

# ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ ВЫСОКОГО УРОВНЯ PASCAL

*Говорите как все, но думайте по-своему.*

*Б. Паскаль*

# Содержание

2

- Историческая справка, основные свойства
- Структура программы
- Лексемы
- Константы
- Типы данных
- Выражения

# Историческая справка

3

- Разработан в 1968-71 гг. **Никлаусом Виртом**, профессором Цюрихского института информатики (Швейцария)
  - ▣ Первоначальное назначение – *обучение студентов программированию.*
  - ▣ Оказался эффективным языком высокого уровня.
- Является одним из самых распространенных языков программирования. Имеет многочисленные диалекты и расширения.



# Основные свойства Pascal

4

- Простой синтаксис
- Развитая система типов данных
- Блочная организация программы
- Строгая типизация
  - фиксация типов переменных
  - строгий контроль преобразования типов и доступа к данным в соответствии с их типом
- Отражение в управляющих структурах принципов структурного программирования

# Структура программы

5

Спецификация программы

Заголовок программы

Объявление констант

Объявление типов данных

Объявление подпрограмм

Объявление переменных

*Объявления программы*

*Головной блок программы*

*Тело программы*

- Объявления могут следовать в любом порядке, если для каждого программного объекта его определение осуществляется *перед* его использованием.

# Определение и использование программных объектов

6

```
const
  N=100;
type
  Idx=1..N;
  Arr=array [Idx] of Byte;
procedure Find(A: Arr; var N: Integer);
begin ... end;
var
  A: Arr;
```

# Спецификация программы

7

**Спецификация программы**

**Заголовок программы**

Объявление констант

Объявление типов данных

Объявление подпрограмм

Объявление переменных

*Объявления программы*

**Головной блок программы**

*Тело программы*

- Комментарий в начале программного файла, в котором указано
  - ▣ имя файла
  - ▣ назначение программы
  - ▣ автор программы
  - ▣ дата создания программы.

# Спецификация программы

8

```
{ TRIANGLE.PAS
```

```
Нахождение вида треугольника.
```

```
(с) Иванов И.И. (ММ-156)
```

```
20-сен-12 }
```

```
/*
```

```
@filename agentmgr.c
```

```
Менеджер параллельных агентов.
```

```
@author (с) Л.Б. Соколинский, М.Л. Цымблер
```

```
@version 1.0
```

```
@seealso exec.c agentmgr.h
```

```
*/
```

# Заголовок программы

9

Спецификация программы

Заголовок программы

Объявление констант

Объявление типов данных

Объявление подпрограмм

Объявление переменных

*Объявления программы*

*Головной блок программы*

*Тело программы*

- Ключевое слово `Program`, за которым следует идентификатор.
  - ▣ `Program Triangle;`
  - ▣ `Program MyProg;`
  - ▣ `Program Example;`

# Объявление констант

10

Спецификация программы

Заголовок программы

Объявление констант

Объявление типов данных

Объявление подпрограмм

Объявление переменных

*Объявления программы*

*Головной блок программы*

*Тело программы*

□ Ключевое слово `const`, за которым следуют определения КОНСТАНТ.

- `const`  
`MaxN=100;`
- `const`  
`Eps=0.001;`  
`N=1000;`

# Объявление типов данных

11

Спецификация программы

Заголовок программы

Объявление констант

Объявление типов данных

Объявление подпрограмм

Объявление переменных

*Объявления программы*

*Головной блок программы*

*Тело программы*

- Ключевое слово `type`, за которым следуют определения типов.

```
□ const
    N=100;
type
    Idx=1..N;
    Arr=array [Idx]
    of Byte;
```

# Объявление переменных

12

Спецификация программы

Заголовок программы

Объявление констант

Объявление типов данных

Объявление подпрограмм

Объявление переменных

*Объявления программы*

*Головной блок программы*

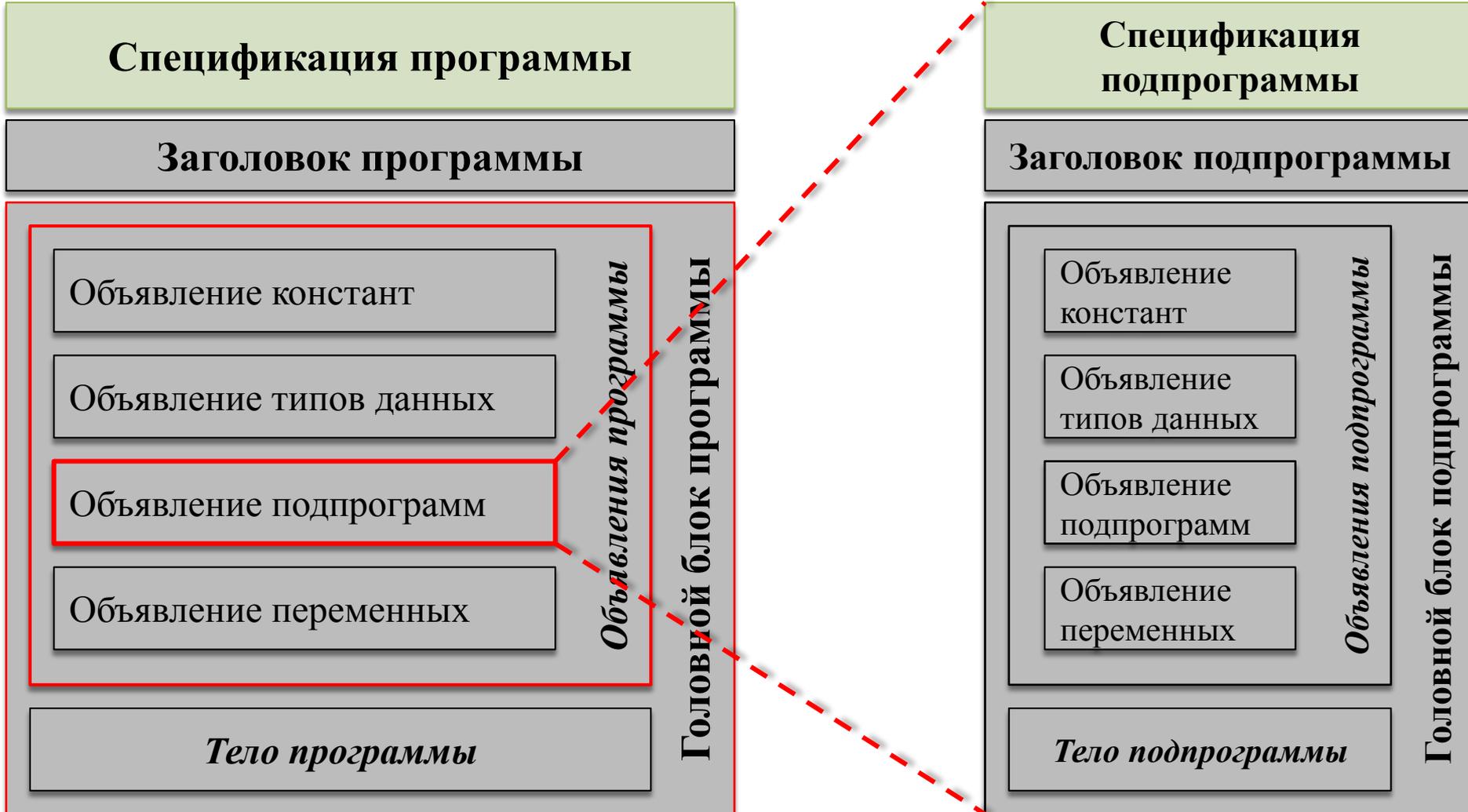
*Тело программы*

- Ключевое слово `var`, за которым следуют определения переменных.

- `const`  
    `N=100;`  
    `type`  
        `Idx=1..N;`  
        `Arr=array [Idx]`  
            `of Byte;`  
    `var A: Arr;`

# Объявление подпрограмм

13



# Объявление подпрограмм

14

**Спецификация подпрограммы**

**Заголовок подпрограммы**

Объявление констант

Объявление типов данных

Объявление подпрограмм

Объявление переменных

*Объявления подпрограммы*

*Головной блок подпрограммы*

*Тело подпрограммы*

```
{ Нахождение корней КВУР  
a*(x^2)+b*x+c=0.  
x1,x2 - корни КВУР  
N - количество корней }  
procedure Eqtn(a,b,c: Real;  
var x1,x2: Real; var N: Integer);  
var  
    D: Real;  
begin  
    D:=b*b-4*a*c;  
    if D<0 then  
        N:=0  
    else  
        if D>0 then begin  
            N:=2;  
            x1:=(-b+sqrt(D))/(2*a);  
            ...
```

# Лексемы

15

- Спецсимволы
- Резервированные слова
- Идентификаторы
- Числа
- Метки
- Символьные строки
- Комментарии
- Строки программы

# Спецсимволы

16

- **Одиночные символы – спецсимволы**  
+ - \* / = < > [ ] . , ( ) : ; ' ^ @ { } \$ #
- **Пары символов – спецсимволы**  
<= >= := .. (\* \*) (. .)
- **Эквивалентные лексемы-спецсимволы**

{	и	(*
}	и	*)
[	и	(.
]	и	.)

# Зарезервированные слова

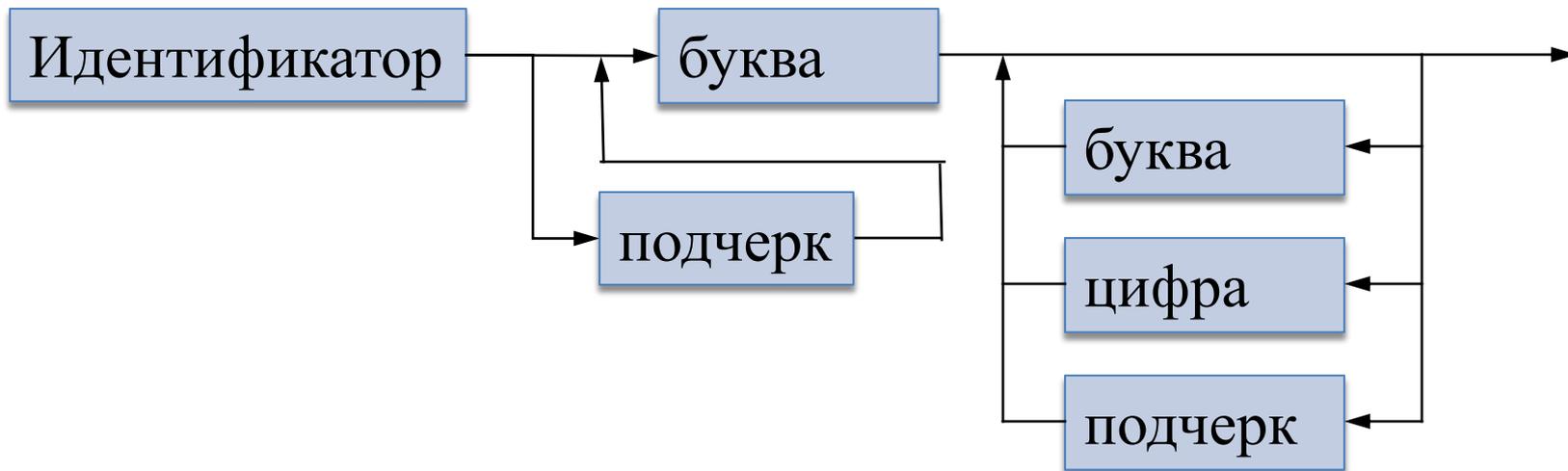
17

- *Зарезервированные (ключевые) слова* – это фиксированный набор английских слов (около 60), смысл и способ использования которых строго определен в описании языка.
  - нельзя использовать для именования программных объектов
  - При написании БОЛЬШИЕ и маленькие буквы не различаются.
- Примеры:  
begin end Program Integer Real function procedure  
in set of case Char

# Идентификаторы

18

- *Идентификаторы* используются для именования программных объектов.
- Идентификатор может быть любой длины, однако только первые 63 символа являются значимыми.



# Примеры идентификаторов

19

i  
Message  
B52  
Done  
\_stop  
\_\_NEVER\_\_  
UnitA.Size  
UnitB.Size  
Very\_long\_identifier

# Правила подбора идентификаторов

20

1. Ясность и удобочитаемость идентификатора.
2. Из слов английского языка.
3. Оптимальная длина идентификатора (зависит от 1.)

# Примеры подбора идентификаторов

21

Семантика	Хорошо	Плохо
Флаг окончания	Done, Stop, Finish	A, Konec
Точность	Eps	E, T, Tochnost
Номер	Num, Number, N	Nomer
Номер записи	RecNum, RecNumber	NumRec, { запись номера} NumRecord
Счетчик цикла	i, j, k, l, m, n	Counter, Number
Сумма	Sum	S, Summa
Количество слов	WordQty	QuantityOfWords, Quantity_of_Words

# Целые числа

22

- *Число* – это целая (типа **Integer**) или вещественная (типа **Real**) константа.
- *Целые числа* записываются с помощью *десятичной* или *шестнадцатеричной* нотации. Перед шестнадцатеричной записью числа ставится символ **\$**.
- Примеры целых чисел  
25  
-3  
\$20      { = 32 }  
\$7D1     { = 2001 }

# Вещественные числа

23

- *Вещественные числа* записываются с помощью десятичной или экспоненциальной нотации.
- В десятичной нотации целая и дробная часть разделяются точкой.
- В экспоненциальной нотации число записывается в виде <Мантисса>E<Порядок>.
- Примеры вещественных чисел:  
3.14159                      -0.005  
1.2e-3 {  $1.2 \times 10^{-3}$  }    E-3 {  $10^{-3}$  }

# Метки

24

- *Метка* – это идентификатор или целое без знака.
- Метки используются с оператором безусловного перехода **goto**. Оператор **goto**, как правило, снижает ясность и читаемость программы. Использование меток и оператора **goto** без крайней необходимости не рекомендуется.

# Символьные строки

25

- *Символьная строка* – это (возможно пустая) последовательность символов, заключенных в апострофы.
- Два смежных апострофа внутри символьной строки трактуются как один апостроф. Символьная строка может включать в себя ASCII-код символа, предваряемый символом #.
- Примеры символьных строк:  
'Pascal'  
'Скарлет О'Хара'  
'1-я строка'#10#13'2-я строка'

# Комментарии

26

- *Комментарий* – любой текст, заключенный в лексемы { и } или (\* и \*). Комментарии игнорируются компилятором и не увеличивают время работы программы.
- *Обязательные комментарии*
  - ▣ спецификация программного файла
  - ▣ спецификация подпрограммы.
- *Рекомендуемые комментарии*
  - ▣ пояснения об используемых алгоритмах (в теле программы или подпрограммы).
- *Не рекомендуемые комментарии* (примеры)
  - ▣ `var i: Integer; { Счетчик цикла }`
  - ▣ `i := i + 1; { Увеличить счетчик на единицу }`

# Препроцессор

27

- *Препроцессор* – компонент компилятора, выполняющий различные преобразования исходного текста перед проверкой его синтаксической правильности (условная компиляция, включение указанного файла и др.).
- Комментарий вида **`{<Команда>}`** является *командой препроцессора*.

Program BigTask;

**`{<I small.pas>}`**

begin

...

end.

**`{<DEFINE DemoVersion>}`**

const

**`{<IFDEF DemoVersion>}`**

MaxUsers=2;

**`{<ELSE>}`**

MaxUsers=200;

**`{<END>}`**

# Строки программы

- Строка программы (в системе Turbo Pascal) не может превышать длину в 126 символов. При записи строк программы принято использовать *лесенку*.
- *Лесенка* – это отступы от начала строки, отражающие структурную вложенность операторов. Лесенка повышает наглядность текста программы и не оказывает влияния на логику программы.

# Правила использования лесенки

29

1. Зарезервированные слова одного и того же оператора, располагающиеся в разных строках, должны начинаться с одной и той же позиции.
2. Перед вложенным оператором должен быть отступ.
3. Перед `begin` и `end` отступы не делаются.
4. Количество пробелов в отступах должно быть одинаковым в пределах программы (рекомендуемые значения: 2-4).

# Примеры лесенки

30

```
if x>y then
  if x>z then
    Max:=x
  else
    Max:=z
else
  if y>z then
    Max:=y
  else
    Max:=z;
```

```
if X>1 then
begin
  ...
end;

if X>1 then begin
  ...
end;

if X>1 then
  ...
else begin
  ...
end;

while S>Eps do begin
  ...
end;
```

# Константы

31

- Определение константы
- Использование констант

# Константа

32

- *Константа* – это идентификатор, связанный с выражением, значение которого вычисляется компилятором до выполнения программы, причем это значение не может измениться во время выполнения программы.
- *Объявление константы* определяет константу внутри блока, содержащего данное объявление.

Объявление  
константы

const

идентификатор

=

константное  
выражение

;

# Константное выражение

33

- *Константное выражение* строится по тем же правилам, что и обычное выражение.
  - В константное выражение не могут входить:
    - ссылки на переменные
    - оператор получения адреса @
    - вызовы пользовательских функций
  - Константное выражение не может ссылаться на определяемую константу.

# Примеры объявления констант

34

```
const
```

```
Min = 5;
```

```
Max = 100;
```

```
Center = (Max-Min) div 2;
```

```
Beta = Chr(225);
```

```
NumChars = Ord('z') - Ord('a') + 1;
```

```
OutOfMemMsg = 'Не хватает памяти';
```

```
ErrStr = 'Ошибка: ' + OutOfMemMsg + '.';
```

```
Numeric = ['0'..'9'];
```

```
Alpha = ['A'..'Z', 'a'..'z'];
```

```
AlphaNum = Numeric + Alpha;
```

# Переменные с начальным значением (типизированные константы)

35

- При описании типизированной константы указывается ее тип (любой, кроме файлового) и начальное значение, которое может быть впоследствии изменено.

type

```
TPerson = record
```

```
  Name: String;
```

```
  Age: Byte;
```

```
  Gender: Char;
```

```
end;
```

```
TArray=array[1..MaxN] of Real;
```

const

```
Eps : Real = E-10;
```

```
Stop: Boolean = False;
```

```
Ivanov: TPerson =
```

```
  ('Иванов И.И.', 20, 'М');
```

```
A: TArray=(1,2,3,4,5,6,7,8,9,0);
```

# Использование констант

- Константы – избыточное, но часто используемое средство языка программирования.
- Если в тексте программы (подпрограммы, модуля unit) предполагается частое использование одного и того же числа, символьной строки и др., то полезно сделать в программе (подпрограмме, модуле unit) объявление соответствующей константы.
- Использование констант с «говорящими» идентификаторами повышает наглядность и читаемость программ.

# Типы данных

37

- Понятие типа данных
- Классификация типов
- Эквивалентность и совместимость типов
- Преобразование типов

# Тип данных

38

- *Тип данных* определяет множество возможных значений и набор допустимых операций для переменных этого типа.
- Тип данных определяет
  - ▣ представление переменных в оперативной памяти
  - ▣ выбор машинных команд для выполнения операций над переменными.

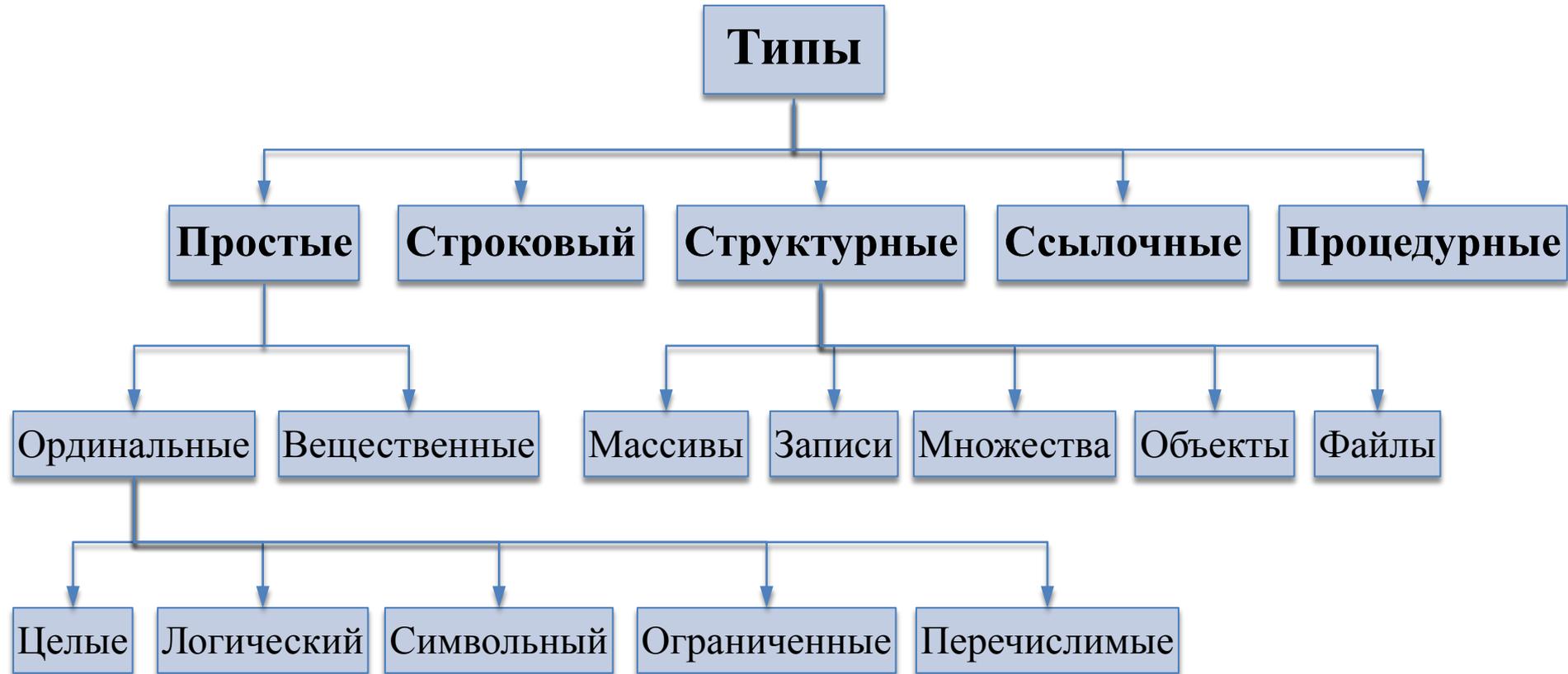
# Язык строгой типизации данных

39

- Каждый программный объект имеет один и только один тип данных.
- Тип данных каждого программного объекта определяется во время компиляции и не меняется во время выполнения программы.
- Для каждого типа данных определяется ограниченный набор операций, что позволяет проверить правильность использования типа во время компиляции.

# Классификация типов данных

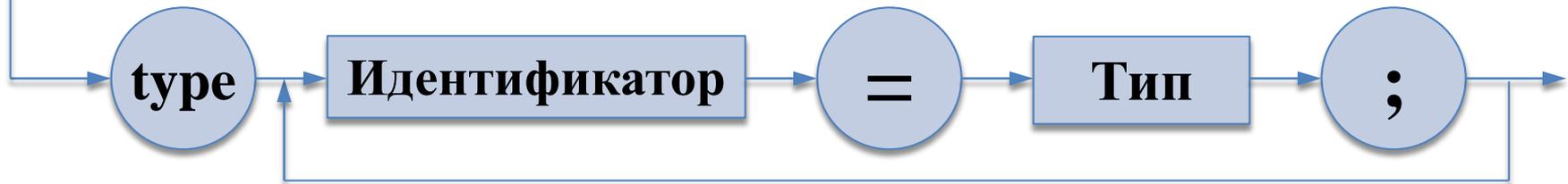
40



# Объявление типов

41

Объявление типа



# Простые типы

42

- *Простые типы* задают упорядоченные множества значений.



# Ординальные типы

43

- Значения ординального (упорядоченного) типа представляют собой *упорядоченное множество*, с каждым элементом которого связан его порядковый номер.
- Нумерация с 0. Любое значение, кроме первого, имеет предшественника. Любое значение, кроме последнего, имеет последователя.

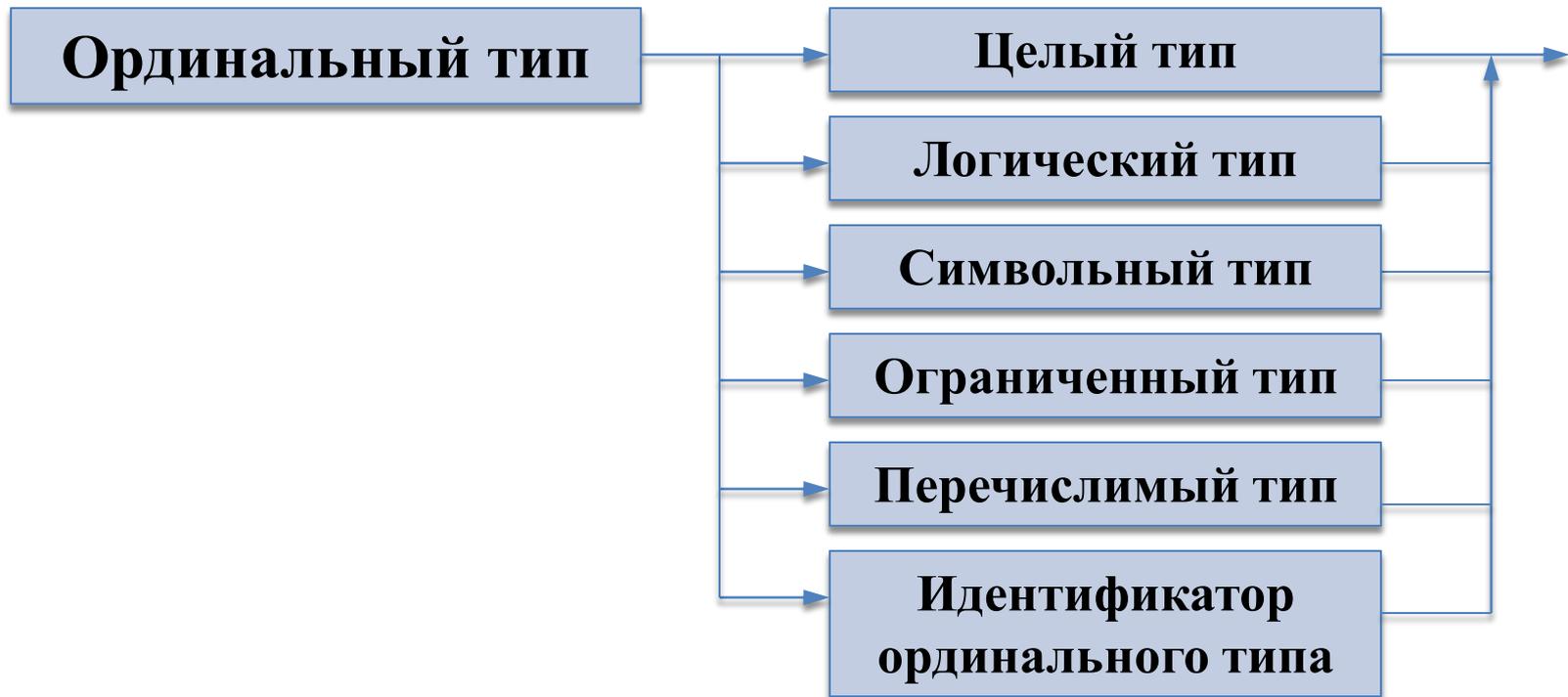
# Функции для ординальных типов

44

- **Ord**  
По значению ординального типа возвращает *порядковый номер значения*.
- **Pred**  
По значению ординального типа возвращает *предшествующее значение*.
- **Succ**  
По значению ординального типа возвращает *последующее значение*.
- **Low**  
По ординальному типу или переменной ординального типа возвращает *наименьшее значение* данного типа.
- **High**  
По ординальному типу или переменной ординального типа возвращает *наибольшее значение* данного типа.

# Классификация ординальных типов

45



# Целые типы

46

Тип	Семантика	Диапазон	Формат
<b>ShortInt</b>	Короткое целое	-128..127	8-битное со знаком
<b>Integer</b>	Целое	-32 768..32 767	16-битное со знаком
<b>LongInt</b>	Длинное целое	-2 147 483 648..2 147 483 647	32-битное со знаком
<b>Byte</b>	Байт	0..255	8-битное без знака
<b>Word</b>	Слово	0..65 535	16-битное без знака

# Арифметические операции над целыми типами

47

- Тип целой константы – это стандартный целый тип с наименьшим диапазоном, охватывающим значение данной константы.
- Для бинарных операций оба операнда приводятся к общему типу перед выполнением операции. Общий тип есть стандартный целый тип с наименьшим диапазоном, охватывающим значения обоих операндов. Например:  
 $\text{Byte} \otimes \text{Byte} = \text{Word}$ ,  $\text{Integer} \otimes \text{Word} = \text{LongInt}$ .
- Выражение справа от оператора присваивания вычисляется независимо от типа переменной слева.

# Логический тип

48

- *Логический тип* предоставляет значения **True** (истина) и **False** (ложь).
- Соотношения между значениями логического типа:  
**False < True**  
**Ord(False)=0**  
**Ord(True)=1**  
**Succ(False)=True**  
**Pred(True)=False**

# СИМВОЛЬНЫЙ ТИП (тип Char)

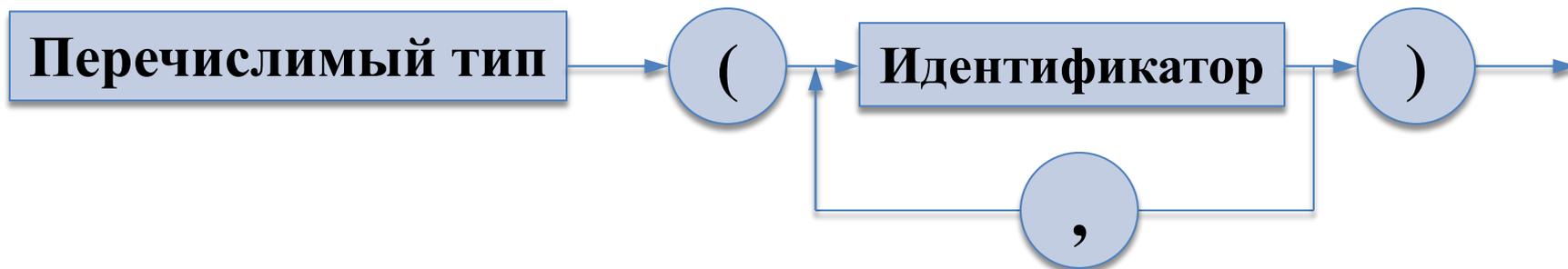
49

- Множество значений *символьного типа* есть множество символов, упорядоченных в соответствии с их ASCII-кодами.
- Любое значение символьного типа может быть получено с помощью стандартной функции *Chr* из его кода ASCII.
- Пример:  
var ch: Char;  
...  
    ch:= 'A';  
    ch:= Chr(32); { ch:= ' '; }

# Перечислимый тип

50

- *Перечислимый тип* определяет упорядоченное множество значений путем перечисления идентификаторов, обозначающих эти значения. Упорядочение значений определяется порядком следования идентификаторов, их определяющих.
- Пример:  
type Suit = (Spades, Clubs, Diamonds, Hearts);  
{ Масть = ♠, ♣, ♦, ♥ . Ord(Spades)=0, Ord(Clubs)=1, ... }



# Ограниченный тип

51

- *Ограниченный тип* представляет собой поддиапазон значений из некоторого ординального типа, называемого *базовым*.

- Примеры:

type

TeenAge=13..19;

RedSuit=Diamonds..Hearts;



# Вещественные типы

52

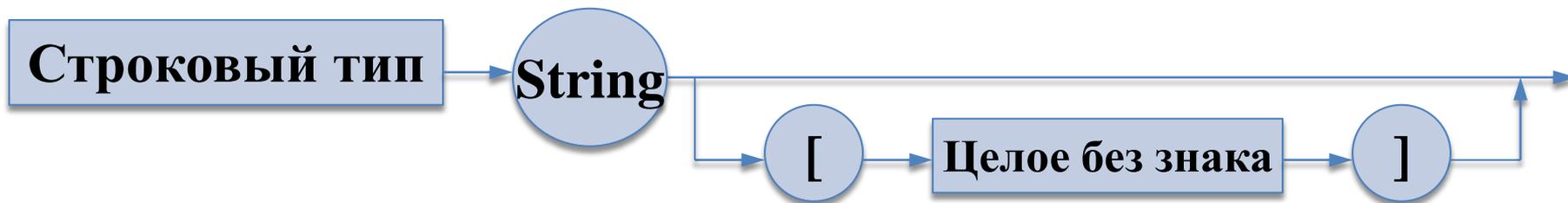
- *Вещественный тип* имеет множество значений, являющееся подмножеством множества действительных чисел, и которые могут быть представлены в формате с плавающей точкой.

Тип	Диапазон	Значащих цифр	Размер в байтах
<b>Real</b>	$2.9 \times 10^{-39} \dots 1.7 \times 10^{38}$	12	6
<b>Single</b>	$1.5 \times 10^{-45} \dots 3.4 \times 10^{38}$	8	4
<b>Double</b>	$5.0 \times 10^{-324} \dots 1.7 \times 10^{308}$	16	8
<b>Extended</b>	$3.4 \times 10^{-4932} \dots 1.1 \times 10^{4932}$	20	10
<b>Comp</b>	$-2^{63} + 1 \dots 2^{63} - 1$	20	8

# Строковый тип

53

- Значение *строкового типа* – это последовательность символов с атрибутами "динамическая длина" и "размер".
  - ▣ Динамическая длина зависит от фактического количества символов во время выполнения программы
  - ▣ Размер: от 1 до 255. По умолчанию (без указания размера) размер 255 символов.
  - ▣ Текущее значение длины: стандартная функция **Length**.
- Лексикографический порядок сортировки: 'abc' < 'ac', 'ab' < 'aba'.
- Символы строки доступны как элементы массива.



# Примеры работы со строковым типом

54

var

```
S: String[20];
```

```
Ch: Char;
```

...

```
S:='Pascal/Delphi';
```

```
WriteLn('Длина S = ', Length(S)); { 13 }
```

```
Ch:=S[1]; { 'P' }
```

```
Ch:=S[0]; { #13 }
```

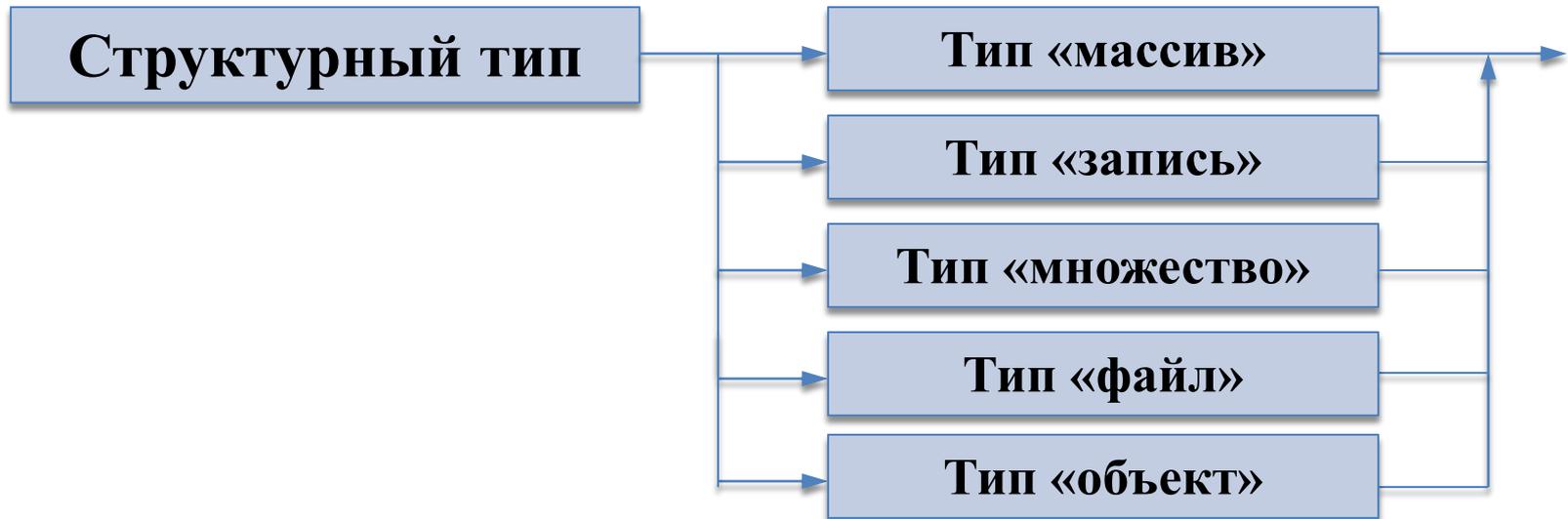
# Структурные типы

55

- *Структурный тип данных* представляет объекты, содержащие сразу несколько значений, называемых элементами. Структурный тип характеризуется типом (или типами) элементов, составляющих объект, и способом доступа к элементам.
- Элементы объекта структурного типа сами могут иметь структурный тип (многоуровневая структуризация). Количество уровней структуризации не ограничено.

# Классификация структурных типов

56



# Тип «массив»

57

- *Массив* содержит фиксированное число элементов одного типа.
- В качестве индексного типа допустим любой ординальный тип, кроме `LongInt` и ограниченных типов, основанных на `LongInt`.
- Стандартные функции **Low** и **High**, примененные к массиву, выдают нижнюю и верхнюю границы (первого) индекса соответственно.



# Примеры работы с массивами

58

```
type
  TVector=array [1..N] of Real;
  TMatrix=array [1..N,1..N] of Real;
  { или   =array [1..N] of
    array [1..N] of Real }
  TCube=array[1..N,1..N,1..N] of Real;
  Month=(Jan,Feb,Mar,Apr,May,Jun,
  Jul,Aug,Sep,Oct,Nov,Dec);
  DaysInMonth=array [Month] of Byte;
var
  V: TVector;
  M: TMatrix;
  C: TCube;
  DM: DaysInMonth;
```

```
for i:=1 to N do
  V[i]:=0;
```

```
for i:=1 to N do
  for j:=1 to N do
    M[i,j]:=1;
```

```
C[N div 2, 1, 1]:=3;
```

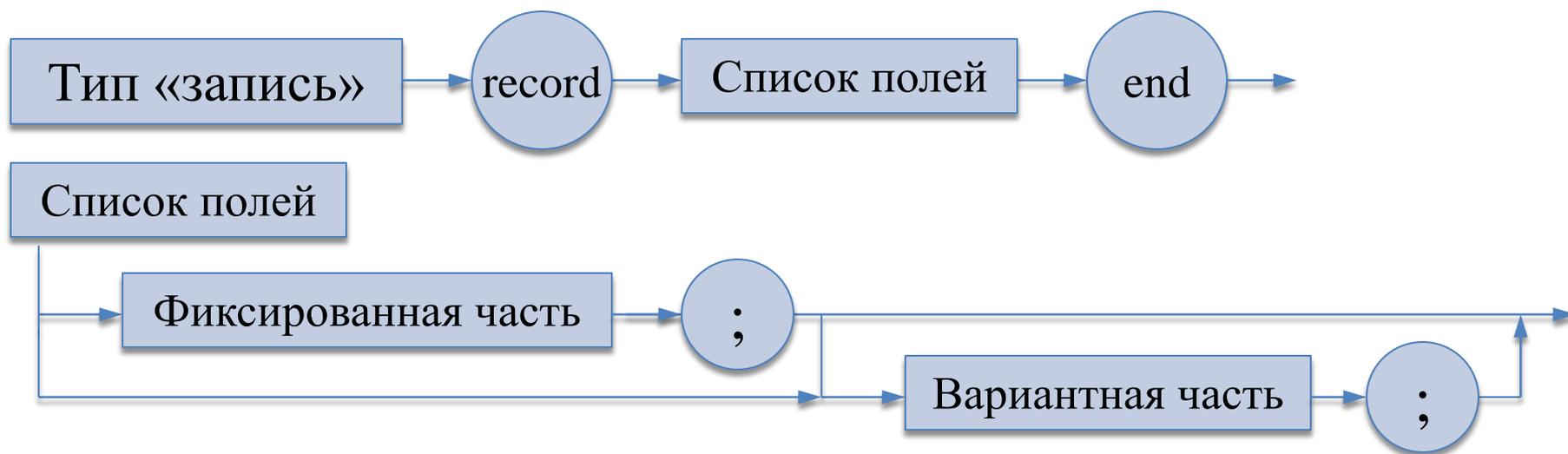
```
DM[Jan]:=31;
```

```
DM[Feb]:=28;
```

# Тип «запись»

59

- *Запись* содержит несколько элементов разных типов. Элемент записи называется *полем*.



<Фиксированная часть> ::= <Список идентификаторов> : <Тип>  
[ ; <Фиксированная часть>]

<Вариантная часть> ::= case [< Идентификатор>: ] <Тип поля-тега> of  
<Список вариантов> end;

<Тип поля-тега> ::= <Идентификатор ordinalного типа>

< Вариант > ::= <Список констант> : (<Список полей>)

# Пример: запись без вариантной части

60

```
type
  TDate=record
    Day: 1..31;
    Month: 1..12;
    Year: Word;
end;
Gender=(Male, Female);
TPerson=record
  FirstName,
  LastName: String[50];
  BithDate: TDate;
  Sex: Gender;
end;
```

```
var
  P: TPerson;
  P.FirstName:='Владимир';
  P.LastName:='Маяковский';
  P.BirthDate.Day:=19;
  P.BirthDate.Month:=7;
  P.BirthDate.Year:=1893;
  P.Sex:=Male;
```

# Пример: запись с вариантной частью

61

```
type
  TShape=(Point, Circle, Rectangle);
  TFigure=record
    X,Y: Real;
    case Shape: TShape of
      Point: ();
      Circle: (Radius: Real);
      Rectangle: (A, B: Real);
    end;
  end;
...
case F.Shape of
  Point: WriteLn('Точка не имеет площади!');
  Circle: WriteLn('Площадь круга ', Pi*F.Radius*F.Radius);
  Rectangle : WriteLn('Площадь прямоугольника ', F.A*F.B);
end;
```

# Тип «множество»

62

- *Тип «множество»* представляет всевозможные подмножества значений некоторого ординального типа, называемого базовым. Базовый тип не может иметь более 256 возможных значений.

- Примеры:

type

CharSet=set of Char;

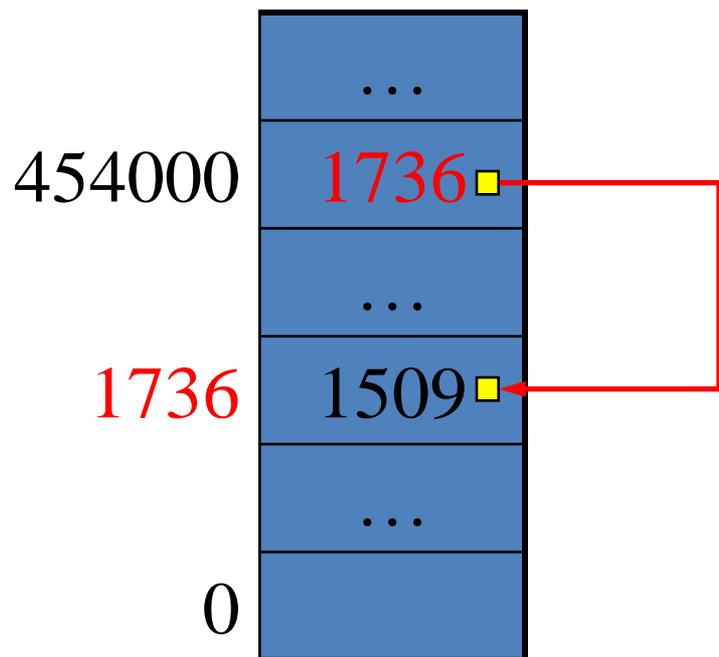
WordSet=set of Word; { Синтаксическая ошибка! }



# Ссылочный тип

63

- *Ссылочные типы* используются для описания указателей.
- *Указатель* – значение, задающее адрес другого значения в памяти.

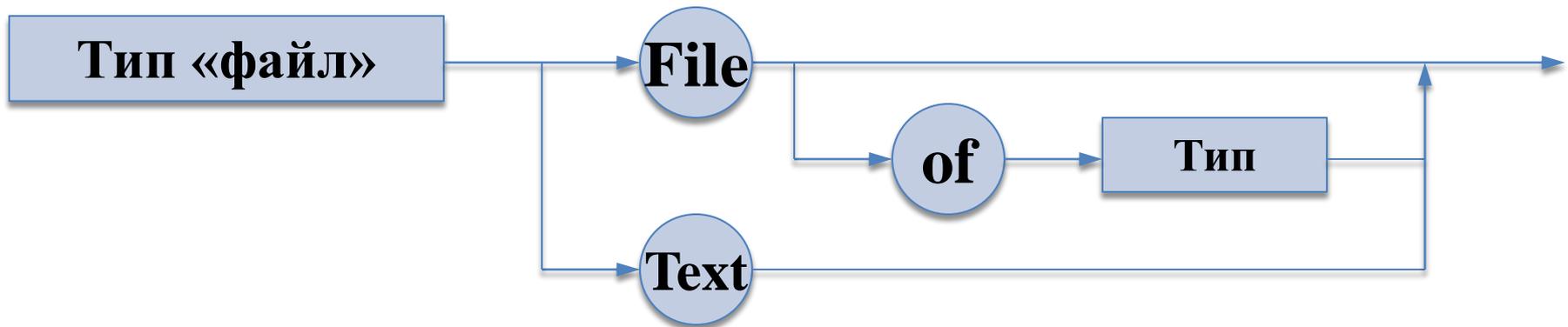


```
type
  PInteger = ^Integer;
  PReal = ^Real;
  PNode = ^TNode;
  TNode = record
    Info: Real;
    Next: PNode;
  end;
```

# Тип «файл»

64

- *Тип «файл»* состоит из линейной последовательности компонент некоторого типа.
- Тип компонент файла не может быть файловым типом, структурным типом, содержащим элементы файлового типа, и объектным типом. Количество компонент не фиксируется при определении файлового типа.



# Процедурный тип

65

- *Процедурный тип* предоставляет возможность использования переменных-подпрограмм (процедур и функций).

```
type
  TFunc = function(X: Real): Real;

procedure PrintFunc(a, b, S: Real; f:
TFunc);
{ Печатает таблицу значений функции
f на отрезке [a;b] с шагом S }
var x: Real;
begin
  x:=a;
  while x<=b do begin
    WriteLn('f(',x:5:5,')=', f(x):5:5);
    x:=x+S;
  end;
end;
```

```
function tg(X: Real): Real;
begin
  tg := sin(X)/cos(X);
end;
```

```
function MyF(X: Real): Real;
begin
  MyF := sin(X*X)+cos(X*X);
end;
```

```
begin
  PrintFunc(1, 2, 0.01, tg);
  PrintFunc(0, 1, 0.001, MyF);
end.
```

# Объектный тип

66

- *Объектный тип* обеспечивает синтаксическую поддержку концепций объектно-ориентированного программирования (ООП).

```
type
  TMan = class
    Name: String;
    Sex: Char;
    Age: Integer;
    procedure Init(aName: String; aSex: Char; aAge: Integer);
  end;
```

# Эквивалентность типов

67

- Для контроля соответствия типов переменных (например, формальных и фактических параметров подпрограмм) в языке программирования должно быть определено понятие *эквивалентности типов данных*.
- Подходы к определению эквивалентности типов:
  - *структурная эквивалентность* – типы эквивалентны, если эквивалентны их структуры (число составляющих компонентов и их тип);
  - *именная эквивалентность* – типы эквивалентны, если эквивалентны идентификаторы их имен.

# Эквивалентность типов в языке Паскаль

68

- В языках строгой типизации, как правило, используется *именная эквивалентность*.
- *Типы T1 и T2 эквивалентны*, если выполняется одно из следующих условий:
  - ▣ T1 и T2 есть один и тот же идентификатор типа;
  - ▣ T1 объявлен как type T1=T2;
  - ▣ T2 объявлен как type T2=T1.

# Примеры эквивалентности типов

69

type

```
T1 = Integer;  
T2 = T1;  
T3 = Integer;  
T4 = T2;  
T5 = array [1..3] of Integer;  
T6 = array [1..3] of Integer;
```

var

```
V1, V2: array [1..3] of  
        Integer;  
V3: array [1..3] of Integer;  
V4: array [1..3] of Integer;  
V5: Integer;  
V6: Integer;
```

**Эквивалентные типы**

**{ структурная и именная эквивалентность }**

T1, T2, T3, T4 и Integer  
переменных V1 и V2  
переменных V5 и V6

**НЕ эквивалентные типы**

**{ структурная, но не именная эквивалентность }**

T5 и T6  
переменных V3 и V4

# Использование анонимных типов

70

```
const N=10;
type
  TArray = array [1..N] of Integer;
  _TNode = record
    Info: Integer;
    Next: ^TNode;
end;
PNode = ^TNode;
TNode = record
    Info: Integer;
    Next: PNode;
end;
var
  V1: array [1..3] of Integer;
  V2: TArray;
  P1: ^Node;
  P2: PNode;
```

## НЕ эквивалентные типы

- TArray и переменной V1
- поля Next в типе \_TNode и переменной P1

## Эквивалентные типы

- TArray и переменной V2
- поля Next в типе TNode и переменной P2
- Анонимные типы использовать не рекомендуется.

# Совместимость типов

71

- При проверке соответствия типов формальных и фактических параметров подпрограмм (именная) эквивалентность снижает универсальности подпрограмм. Требуется менее строгое понятие – *совместимость типов*.
- *Типы T1 и T2 совместимы*, если выполняется одно из следующих условий:
  - T1 и T2 эквивалентны;
  - тип T1 – диапазон типа T2, либо T1 и T2 – диапазоны одного и того же типа;
  - T1 и T2 – типы-множества с совместимыми базовыми типами;
  - T1 и T2 – типы-строки одной и той же длины;
  - и т.д.

# Примеры совместимости типов

72

type

```
TNumber=Integer;
TTeenAge=13..19;
TIndex=TNumber;
CharSet= set of Char;
YesNo=Boolean;
ZeroOrOne=0..1;
Str10=String[10];
Ch10Str=array[1..10] of Char;
Suit=(Spades, Clubs,
Diamonds, Hearts);
RedSuit=Diamonds..Hearts;
TArr=array[1..N] of Byte;
THalfArr=array[1..N div 2] of
Byte;
TextDoc=Text;
StatFile=file of Real;
```

{ Совместимые типы }

```
Byte, LongInt
Real, Extended
Byte, TNumber
TIndex, TNumber
YesNo, Boolean
CharSet, set of Char
Word, TTeenAge
Suit, RedSuit
String, Str10
TextDoc, Text
```

{ НЕСОВМЕСТИМЫЕ типы }

```
TArr, THalfArr
TNumber, Real
YesNo, ZeroOrOne
Str10, Ch10Str
ZeroOrOne, RedSuit
TextDoc, StatFile
```

# Совместимость по присваиванию

73

- Для проверки синтаксической правильности операторов присваивания в языке определяется понятие *совместимости по присваиванию*.
- *Значение типа T2 совместимо по присваиванию с типом T1* (`var V1: T1; V2:T2;` и допустим оператор `V1:=V2;`), если выполняется одно из следующих условий:
  - T1 и T2 эквивалентны и ни один из них не является файловым типом;
  - T1 и T2 – совместимые ординальные типы, и все значения типа T2 из диапазона значений T1;
  - T1 и T2 – совместимые типы-множества, и все значения типа T2 из диапазона значений базового типа для типа T1;
  - T1 и T2 – совместимые строковые, ссылочные или процедурные типы;
  - тип T1 – вещественный, а тип T2 – целый;
  - тип T1 – строка, а тип T2 – символьный;
  - тип T1 – объектный, а тип T2 – объектный тип-потомок T1.

# Примеры совместимости по присваиванию

74

type

```
TNumber=Integer;  
TTeenAge=13..19;  
TIndex=TNumber;  
TArr=array [1..N] of Real
```

var

```
S: String;  
str10: String[10];  
n: TNumber;  
i: Integer;  
Ch: Char;  
R: Real;  
F1, F2: Text;  
Young: TTeenAge;  
Idx: TIndex;  
B: Byte;  
X, Y: TArr;
```

{ Синтаксически верно }

```
S:=str10;  
n:=i;  
Ch:=S[1];  
R:=i;  
R:=Young; n:=Young;  
Idx:=B;  
X:=Y;
```

{ Синтаксически НЕВЕРНО }

```
F2:=F1;  
n:=R;  
i:=R;  
str10:=S;  
Ch:=S;  
Ch:=str10;  
Young:=n; Young:=Idx;  
B:=i;
```

# Преобразование типов

75

- *Явное преобразование*
  - *с помощью встроенных функций, аргументы которых принадлежат одному типу, а значение другому типу;*
  - *с помощью применения имени типа к выражению преобразуемого типа.*
- *Неявное преобразование*
  - *преобразование операндов целочисленного типа к вещественному типу в выражениях с операндами целочисленного и вещественного типов.*

# Преобразование типов

76

```
var                                     type
    R: Real;                            Enumerated=(One, Two, Three, Four);
    I: Integer;                          var
    C: Char;                             B: Boolean;
    ...                                  I: Integer;
    I:=Round(R);                         C: Char;
    I:=Ord(C);                           E: Enumerated;
    R:=I+0.5;                             ...
                                           I:=Integer(' '); { I:=32; Chr(32)=' ' }
                                           C:=Char(32); { C:=' '; Ord(' ')=32 }
                                           B:=Boolean(0); { B:=False; }
                                           B:=Boolean(2); { B:=True; }
                                           W:=Word(-1); { W:=65535; }
                                           E:=Enumerated(3); { E:=Four; }
                                           I:=Integer(Two); { I:=1; }
```

# Выражения

77

- Построение выражений
- Вычисление значения выражения
- Операции в выражениях

# Выражения

78

- *Выражение* – правило для вычисления значения. Значение выражения имеет один и только один тип.
- Выражение состоит из *операндов* (переменные, константы и др.) и *операций* (отрицание, умножение и др.).
- Если все объекты, входящие в выражение, определены в момент их использования, то выражение считается *определенным*.  
Иначе значение выражения *не определено* и возникает логическая ошибка или ошибка времени выполнения.

# Построение выражений

79

<выражение> ::= <простое выражение>

[ <операция отношения> <простое выражение> ]

<простое выражение> ::= [ <знак> ] <терм>

{ <аддитивная операция> <терм> }

<терм> ::= <множитель>

{ <мультипликативная операция> <терм> }

<множитель> ::= <идентификатор> | <вызов функции> | ( <выражение>

| **NOT** <множитель> | <константа> | <конструктор множества>

<вызов функции> ::= <идентификатор> ( <выражение> [ { , <выражение> } ] )

<константа> ::= <число без знака> | <строка> | <идентификатор> | **NIL**

<конструктор множества> ::= [ [ <элемент множества>

{ , <элемент множества> } ] ]

<элемент множества> ::= <выражение> [ .. <выражение> ]

# Построение выражений

80

## Множители

Eps

125

$\text{Sin}(\text{Sqr}(x)) + \text{Sin}(\text{Sqr}(Y))$

[1..5, 10..20]

[Sun, Sat]

not Stop

$(X * A + Y / B)$

## Термы

A/B

$X * (A + B)$

$(A = B) \text{ and } (X > Y)$

## Простые выражения

$X + Y$

$-A$

A or B

## Выражения

$A = 100$

$A \geq B$

$(A \bmod B) \geq 3$

Card in RedSuit

Day in [Monday..Friday]

$A + B * C \text{ div } D - E(3 + F, 1) = 3$

# Приоритеты операций

81

Пр	Категория операций	Операции
1	Унарные	@, not
2	Бинарные	Мультипликативные *, /, div, mod, and, shl, shr
3		Аддитивные +, -, or, xor
4		Отношения =, <, >, <=, >=, in



# Правила приоритета

82

1. Выражения в скобках вычисляются в первую очередь.
2. Из операций с различными приоритетами в первую очередь вычисляются операции с более высоким приоритетом.
3. Операции с одинаковыми приоритетами вычисляются в порядке их следования, слева направо.

# Правила приоритета

83

Выражение	Порядок выполнения операций
$A+B*C$	$A+(B*C)$
$A+B*C \text{ div } D-E/F$	$(A+((B*C) \text{ div } D))-(E/F)$
$A \text{ or not } B \text{ and } C \langle \rangle \text{True}$	$(A \text{ or } ((\text{not } B) \text{ and } C)) \langle \rangle \text{True}$
$\text{not } A \text{ and } B \text{ or not } C \text{ and } D$	$((\text{not } A) \text{ and } B) \text{ or } ((\text{not } C) \text{ and } D)$
$A \text{ div } C - D * E \text{ mod } F + G > H$	$((A \text{ div } C) - ((D * E) \text{ mod } F) + G) > H$

# Вычисление операндов бинарной операции

84

- Порядок вычисления операндов бинарной операции может зависеть от реализации языка:
  - ▣ в порядке следования операций
  - ▣ в обратном порядке
  - ▣ параллельно
  - ▣ не полностью
- Неполное вычисление логических выражений в языке Турбо Паскаль  
A:=(X=Y) and (Y=Z);  
B:=FuncA(X) or FuncB(Y);

# Бинарные арифметические операции

85

Знак	Операция	Типы операндов	Тип результата	Пример
+	Сложение	Целый Вещественный	Целый Вещественный	$4+1=5$
-	Вычитание	Целый Вещественный	Целый Вещественный	$4-1=3$
*	Умножение	Целый Вещественный	Целый Вещественный	$4*1=4$
/	Деление	Целый Вещественный	Вещественный Вещественный	$4/1=4$
<b>div</b> <b>mod</b>	Деление нацело Взятие остатка	Целый Целый	Целый Целый	$5 \text{ div } 2 = 2$ $5 \text{ div } -2 = -2$ $5 \text{ mod } 2 = 1$ $5 \text{ mod } -2 = 1$

# Унарные арифметические операции

86

Знак	Операция	Тип операнда	Тип результата
+	Сохранение знака	Целый Вещественный	Целый Вещественный
-	Смена знака	Целый Вещественный	Целый Вещественный

# Логические операции

87

Знак	Операция	Типы операндов	Тип результата
<b>not</b>	Отрицание	Логический	Boolean
<b>and</b>	Логическое И	Логический	Boolean
<b>or</b>	Логическое ИЛИ	Логический	Boolean
<b>xor</b>	Логическое исключающее ИЛИ	Логический	Boolean

# Таблицы истинности

88

<b>and</b>	True	False
True	True	False
False	False	False

<b>or</b>	True	False
True	True	True
False	True	False

<b>not</b>	True	False
	False	True

<b>xor</b>	True	False
True	False	True
False	True	False

# Логические операции над целыми числами

89

Знак	Операция	Типы операндов	Тип результата
<b>not</b>	Поразрядное отрицание	Целый	Целый
<b>and</b>	Поразрядное И	Целый	Целый
<b>or</b>	Поразрядное ИЛИ	Целый	Целый
<b>xor</b>	Поразрядное исключающее ИЛИ	Целый	Целый
<b>shl</b>	Поразрядный сдвиг влево	Целый	Целый
<b>shr</b>	Поразрядный сдвиг вправо	Целый	Целый

# Примеры логических операций (1)

90

Пример	Результат	Пояснение
<b>not 5</b>	2	$5=101_2 \rightarrow 010_2=2$
<b>5 and 2</b>	0	AND $\begin{array}{r} 101_2 \\ \underline{010_2} \\ 000_2 \end{array}$
<b>5 or 2</b>	7	OR $\begin{array}{r} 101_2 \\ \underline{010_2} \\ 111_2 \end{array}$
<b>5 xor 3</b>	6	XOR $\begin{array}{r} 101_2 \\ \underline{011_2} \\ 110_2 \end{array}$

# Примеры логических операций (2)

91

Пример	Результат	Пояснение
12 shr 2	3	$12=1100_2 \rightarrow 0011_2=3$
13 shl 1	26	$13=1101_2 \rightarrow 11010_2=26$

# Операции над строками

92

Знак	Операция	Тип операнда	Тип результата
+	Конкатенация	String Char	String

# Операции над множествами

93

Знак	Операция	Пример
+	Объединение	$['0'..'9'] + ['a'..'z'] = ['0'..'9', 'a'..'z']$
-	Разность	$['0'..'9', 'a'..'z'] - ['0'] = ['1'..'9', 'a'..'z']$
*	Пересечение	$['0'..'9'] * ['a'..'z'] = []$

# Операции отношения

94

Знак	Операция	Типы операндов	Тип результата
=	Равно	Совместимые простые, указатели, множества, строки	Boolean
<>	Не равно	Совместимые простые, указатели, множества, строки	Boolean
>	Больше	Совместимые простые, строки	Boolean
<	Меньше	Совместимые простые, строки	Boolean
>=	Больше либо равно	Совместимые простые, строки	Boolean
<=	Меньше либо равно	Совместимые простые, строки	Boolean
<b>in</b>	Содержится в	Левый операнд ординального типа, правый операнд типа множество	Boolean

# Пример: операция **in**

95

```
function IsLetter(Ch: Char) : Boolean;
begin
    IsLetter := Ch in ['a'..'z', 'A'..'Z'];
    { ((Ch>='a') and (Ch<='z')) or ((Ch>='A') and
    (Ch<='Z')) }
end;

if N in [4..5] then
    WriteLn('Зачет')
else
    WriteLn('НЕЗАЧЕТ!');
```

# Заключение

- Язык Pascal
  - ▣ разработан Н. Виртом для обучения студентов программированию
  - ▣ один из самых распространенных языков высокого уровня
- Лексемы языка Pascal
  - ▣ Спецсимволы, ключевые слова, идентификаторы и др.
  - ▣ Комментарии (спецификации и команды препроцессора)
  - ▣ Строки программы: "лесенка"
- Константы
- Типы данных
  - ▣ Язык Pascal – язык строгой типизации данных с большой коллекцией типов
  - ▣ Эквивалентность типов
  - ▣ Совместимость типов
- Выражения
  - ▣ построение выражений
  - ▣ приоритет операций