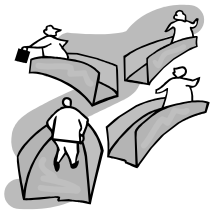
Типы данных



В Англии преобладают два типа женщин: одни не могут рассказать анекдот, другие не могут его понять.

Дж. Грир

Содержание

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

Типы данных

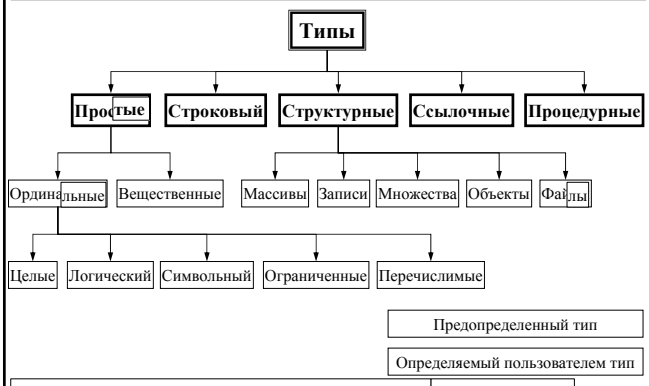
- При объявлении переменной должен быть указан ее *тип данных*.
- *Тип данных* определяет *множество возможных значений* и *набор допустимых операций* для переменных этого типа.
- Во время компиляции информация о типах данных используется для представления переменных в оперативной памяти компьютера и выбора соответствующих машинных команд для выполнения операций над переменными.

Язык строгой типизации данных

- 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.

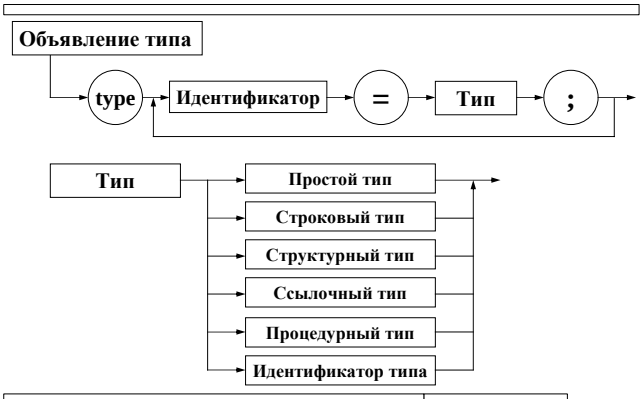
Типы данных | © М.Л. Цымблер | 4

Классификация типов данных



Типы данных | © М.Л. Цымблер | 5

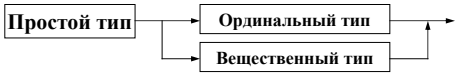
Объявление типов



Типы данных | © М.Л. Цымблер | 6

Простые типы

- *Простые типы* задают упорядоченные множества значений.



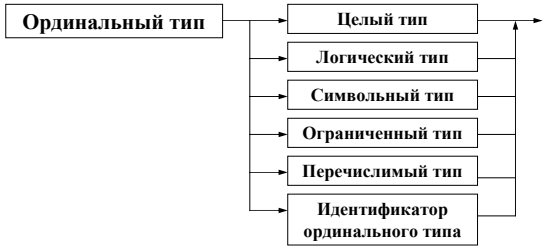
Ординальные типы

- Значения ординального (упорядоченного) типа представляют собой упорядоченное множество, с каждым элементом которого связан его порядковый номер.
- Первое по порядку значение любого ординального типа имеет номер 0, следующее – номер 1 и т.д. Любое значение, кроме первого, имеет предшественника, и любое значение, кроме последнего, имеет последователя, определяемых отношением порядка данного типа.

Функции для ординальных типов

- **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



Целые типы

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

Арифметические операции над целыми типами

- 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: **Byte⊗Byte=Word**, **Integer⊗Word=LongInt**.
- Выражение справа от оператора присваивания вычисляется независимо от типа переменной слева.

Логический тип

- *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

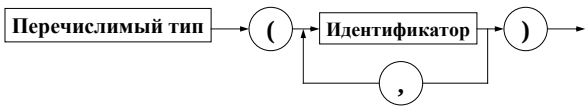
Символьный тип (тип Char)

- 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

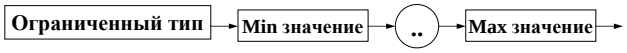
- *Перечислимый тип* определяет упорядоченное множество значений путем перечисления идентификаторов, обозначающих эти значения. Упорядочение значений определяется порядком следования идентификаторов, их определяющих.
- Пример:
  type Suit = (Spades, Clubs, Diamonds, Hearts);
  { Масть = ♠, ♣, ♦, ♥ .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



Ограниченный тип

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

```
type
    TeenAge=13..19;
    RedSuit=Diamonds..Hearts;
```



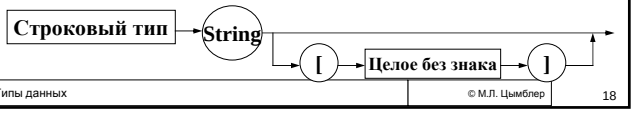
Вещественные типы

- *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
Real	$2.9\times10^{-39}..1.7\times10^{38}$	12	6
Single	$1.5\times10^{-45}..3.4\times10^{38}$	8	4
Double	$5.0\times10^{-324}..1.7\times10^{308}$	16	8
Extended	$3.4\times10^{-4932}..1.1\times10^{4932}$	20	10
Comp	$-2^{63}+1..2^{63}-1$	20	8

Строковый тип

- Значение *строкового типа* - это последовательность символов с атрибутом «динамическая длина» (зависящим от фактического количества символов во время выполнения программы) и с атрибутом «размер» от 1 до 255.
- Строковому типу, объявленному без указания размера, по умолчанию дается размер в 255 символов.
- Текущее значение атрибута «длина» можно получить с помощью стандартной функции **Length**.
- Для значений строкового типа определен лексикографический порядок: 'abc'<'ac' , 'ab'<'aba'.
- Символы строки доступны как элементы массива.



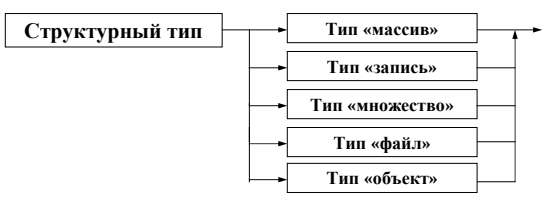
Примеры работы со строковым типом

```
var
        S: String[20];
        Ch: Char;
…
        S:='Turbo Pascal';
        WriteLn('Длина S = ', Length(S)); { 12 }
        Ch:=S[1]; { 'T' }
        Ch:=S[0]; { #12 }
```


Структурные типы

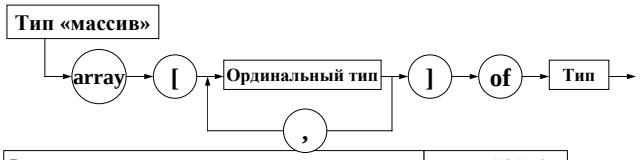
- *Структурный тип данных* представляет объекты, содержащие сразу несколько значений, называемых элементами. Структурный тип характеризуется типом (или типами) элементов, составляющих объект, и способом доступа к элементам.
- Элементы объекта структурного типа сами могут иметь структурный тип (многоуровневая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

Классификация структурных типов



Тип «массив»

- *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.



Типы данных © М.Л. Цымблер 22

Примеры работы с массивами

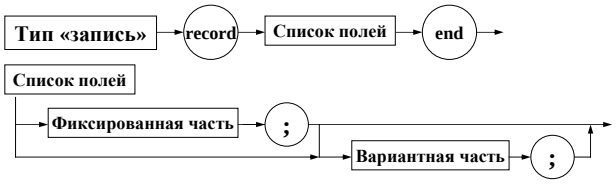
```
type
        TVector=array [1..N] of Real;
        TMatrix=array [1..N,1..N] of Real;
        { или   =array [1..N] of
                array [1..N] of Real }
        TCube=array[1..N,1..N,1..N] of Real;
        Month=(Jan,Feb,Mar,Apr,May,Jun,
                Jul,Aug,Sep,Oct,Nov,Dec);
        DaysInMonth=array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

Типы данных © М.Л. Цымблер 23

Тип «запись»

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




```
<Фиксированная часть> ::= <Список идентификаторов> : <Тип>
                                [ ; <Фиксированная часть>]
<Вариантная часть> ::= case [< Идентификатор>: ] <Тип тэгового поля> of
                        <Список вариантов> end;
<Тип тэгового поля> ::= <Идентификатор ординального типа>
< Вариант > ::= <Список констант> : (<Список полей>)
```

Типы данных © М.Л. Цымблер 24

Пример: запись без вариантной части

```
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;
```

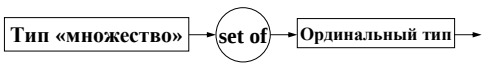

Пример: запись с вариантной частью

```
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»

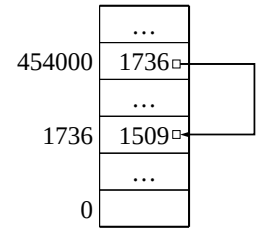
- *Тип «множество»* представляет всевозможные подмножества значений некоторого ординального типа, называемого базовым. Базовый тип не может иметь более 256 возможных значений.
- Примеры:

```
type
    CharSet=set of Char;
    WordSet=set of Word; { Синтаксическая ошибка! }
```



Ссылочный тип

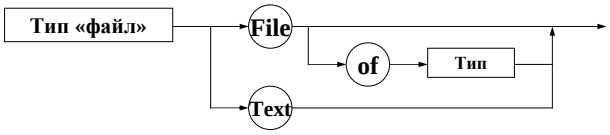
- *Ссылочные типы* используются для описания *указателей*.
- *Указатель* – значение, задающее адрес другого значения в памяти.




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


Тип «файл»

- *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

- *Процедурный тип* предоставляет возможность использования переменных-подпрограмм

```
type
   TFunc = function(X: Real): Real;
function tg(X: Real): Real;
begin
   tg := sin(X)/cos(X);
end;
procedure PrintFunc(Start, Stop, Step: Real; f: TFunc);
{ Печатает таблицу значений функции f
на отрезке [Start;Stop] с шагом Step }
…
PrintFunc(1, 2, 0.01, tg(x));
```

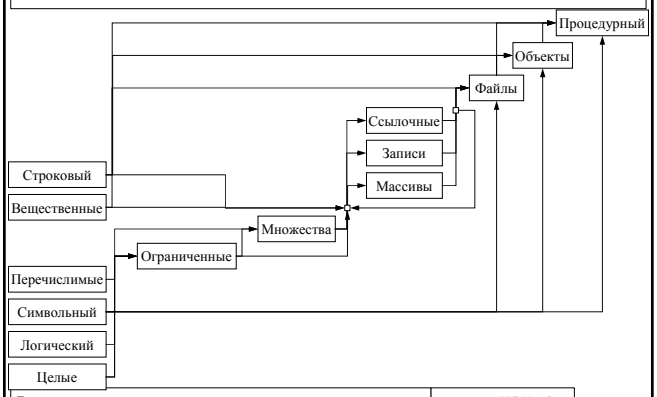
Объектный тип

- *Объектный тип* введен в Turbo Pascal, начиная с версии 5.5, для синтаксической поддержки концепций объектно-ориентированного программирования (ООП).

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

Типы данных © М.Л. Цымблер 31

Построение типов данных



Типы данных © М.Л. Цымблер 32

Эквивалентность типов

- 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.

Типы данных © М.Л. Цымблер 33

Эквивалентность типов в языке Паскаль

- 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.

Типы данных © М.Л. Цымблер 34

Примеры эквивалентности типов

```
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

Типы данных © М.Л. Цымблер 35

Использование анонимных типов

```
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.

Типы данных © М.Л. Цымблер 36

Совместимость типов

- При проверке соответствия типов формальных и фактических параметров подпрограмм (именная) эквивалентность является тормозом для 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;
  - T1 и T2 – процедурные типы с одинаковыми типом результата (для функций), количеством формальных параметров и типами соответствующих параметров;
  - тип T1 – ссылочный, а T2 – нетипизированный ссылочный;
  - тип T1 ссылается на объектный тип, а T2 ссылается на родственный объектный тип.

Типы данных © М.Л. Цымблер 37

Примеры совместимости типов

```
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
```

Типы данных © М.Л. Цымблер 38

Совместимость по присваиванию

- 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 попадают в диапазон возможных значений T1;
  - T1 и T2 – совместимые типы-множества, и все значения типа T2 попадают в диапазон возможных значений базового 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.

Типы данных © М.Л. Цымблер 39

Примеры совместимости по присваиванию

```
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;
```

Типы данных © М.Л. Цымблер 40

Преобразование типов

- *Явное* преобразование
  - *с помощью встроенных функций*, аргументы которых принадлежат одному типу, а значение другому типу;
  - *с помощью применения имени типа* к выражению преобразуемого типа (только Турбо Паскаль).
- *Неявное* преобразование
  - *преобразование операндов целочисленного типа* к вещественному типу в выражениях с операндами целочисленного и вещественного типов;
  - *совмещение в памяти данных* разных типов (только Турбо Паскаль).

Типы данных © М.Л. Цымблер 41

Преобразование типов

```
var
   R: Real;
   I: Integer;
   C: Char;
…
I:=Round(R);
I:=Ord(C);
R:=I+0.5;
```

```
type
   Enumerated=(One, Two, Three, Four);
var
   B: Boolean;
   I: Integer;
   C: Char;
   E: Enumerated;
…
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; }
```

Типы данных © М.Л. Цымблер 42