# Delphi – lekce 6

## Minimum z Object Pascalu

Vrátíme se ještě k základům Object Pascalu.

## Struktura programu

Object Pascal je přísně **typový procedurální jazyk**, který umožňuje jak strukturované, tak objektově orientované programování.

Příkazy jsou oddělovány středníkem, bílé znaky (mezery, tabulátory, nové řádky jsou ekvivalentem jedné mezery – pokud nejsou součástí řetězce), skupiny příkazů začínají **begin** a končí **end**.

Program se člení na příkazy seskupené do procedur nebo funkcí, které se případně seskupují do tříd. Procedury a funkce, nebo třídy jsou uloženy v jednotce (unit).

Celý program se skládá ze souboru s klíčovým slovem **program** a případných jednotek. Nejmenší program má jen jeden soubor s uvedeným klíčovým slovem **program** – v delphi dle konvence soubor **dpr**. Jednotky jsou uloženy v souborech **pas**.

Jednotka má *obvykle* dvě části: rozhraní (interface) a implementaci (implementation).

```
unit Jednotka;

interface
uses
  Jednotka2;
procedure Test;

implementation
uses
  Jednotka3;
procedure Test;
begin
  ShowMessage('test');
end;

end.
```

V části interface se specifikuje všechno, co má být dostupné z jiných jednotek. V části implementation rozhraní implementujeme.

Části s uses jsou nepovinné. Ohledně umístění uses (interface x implementation): pokud je to možné tak používáme uses v implementation, jelikož tím usnadníme kompilátoru kontrolu závislostí (a tím zrychlíme kompilaci) a vyhneme se kruhové referenci jednotek.

## Typy proměnných

### Ordinální typ

Jedná se o datový typ, kdy každé hodnotě daného typu se dá přiřadit unikátní hodnota pořadí (ordinální hodnota). Patří sem všechny celočíselné typy, char, boolean a výčtový typ.

Pro získání ordinální hodnoty existuje funkce **ord**, předchůdce je **pred** a následník je **Succ**. Dále lze použít **dec** a **inc**, kdy tyto procedury snižují nebo zvyšují hodnotu o 1 (případně o hodnotu specifikovanou druhým parametrem).

### Numerické typy

Rozeznáváme celočíselné (zároveň jsou ordinální) a realné. Kromě toho ještě zvláštní typ Currency, což je kříženec, jelikož je to reálný typ, ale s fixní přesností a je uložen interně jako celé číslo (při operacích mezi ním a reálným typem kompilátor interně násobí nebo dělí 10000).

Celočíselné typy (byte, word, integer, cardinal, longint, int64…) se rozlišují dle velikosti (uložení v paměti na byte) a zda jsou znaménkové. Některé mají definovanou přesnou velikost, jiné jsou generické, a tudíž mohou mít na jiných platformách nebo CPU jinou velikost (Integer). Obecně by se měl používat Integer (pokud není zvláštní důvod pro jiný typ). Pro zadání v 16soustavě se používá prefix $ např. $0A.

Reálné typy se rozlišují dle velikosti v paměti (single, double, extended) a tím dle přesnosti. Původní typ real z pascalu je nyní mapován na double, pokud někdo potřebuje původní real (který není přímo podporován matematickým koprocesorem) musí použít real48. Obecně by se měl používat double.

Celočíselné typy podporují **mod** a **div**, tj. celočíselné dělení a dále bitové operace **and or xor not** a bitové posuny **shr** a **shl**. Kromě toho lze použít funkci **Odd**, která vrací zda je argument lichý.

Konverze mezi reálnými a celočíselnými typy: **round** vrací zaokrouhlené číslo, **trunc** vrací celou část reálného čísla, **frac** desetinnou.

Aritmetické funkce: Abs, ArcTan, Cos, Sin (a další geometrické funkce), Ln, Pi, Sqr, Sqrt, Exp. Dále v jednotce Math jsou další optimalizované funkce.

### Boolean

Typ představující logický stav (True nebo False). Uložen jako jeden byte. Podporuje logické operace and, or, xor a not.

Obecně kompilátor podporuje zkrácené vyhodnocování (lze nastavit), tj. pokud je složitější výraz a v některém okamžiku je jasný výsledek – další části výrazu se neprovádí. Obecně tedy je nutno na to pamatovat a to:

- nejrychlejší část výrazu dávat na začátek
- pokud je ve výrazu volání funkce, nemusí být provedeno!

**Char**

Reprezentuje znak. Do Delphi 2007 včetně je velikosti 1 byte (hodnoty 0-255), v následujících unicode verzích 2 byte. Pozor na to v případě ukládání atd. Kompilátor vypisuje v nových Delphi varování. AnsiChar = 8bit char vždy.

Konstanta se definuje jako např. 'z', případně #66, kde 66 je ASCII kód.

Převod na typ Char je možné realizovat pomocí **Chr** a zpět přes **Ord**.

**String**

jedná se o posloupnost Char. Obecně je několik typů stringů, ale zůstaneme u obecného string.

Pro práci s řetězci je mnoho funkcí, např. StringReplace (nahrazení), Pos (výskyt), Copy (substring), Delete (mazání), Length (délka), QuotedStr (vrátí řetězec v apostrofech), Trim (apod. odstraní bílé znaky).

String je velice inteligentně implementován a kompilátor provádí spoustu magie, aby byla práce s nimi rychlá a příjemná (např. kopírování řetezce po přiřazení, až se změní atd).

**Výčtové typy**

Jedná se o ordinální typy, kdy hodnota může být jen s přísně definované množiny dat. Tento typ se ve VCL často používá. Např.

TFormStyle = (fsNormal, fsMDIChild, fsMDIForm, fsStayOnTop);

TPismeno = 'A'.. 'Z';

Je konvencí, že hodnota výčtového typu začíná prvními písmeny názvu typu, viz. **fs**Normal. V paměti je proměnná uložena jako celé číslo.

**Množiny (set)**

Množina je skupina hodnot jednoho ordinálního typu, přičemž základní ordinální typ nesmí mít více než 255 hodnot.

např.
TSetPismeno = set of TPismeno;

```
var
  Znak: TSetPismeno;
begin
  Znak := ['A','B','C'];
  if ‘A’ in Znak then …
```

Pro přidání do množiny se dají používat operátory + a – nebo funkce Include a Exclude.

**Záznamy**

jedná se o strukturovaný datový typ s deklarací

```
type TZaznam = record
  iPolozka1:integer;
  sPolozka2:string;
end;
```

V novějších verzích Delphi byla deklarace rozšířena o možnost volání procedur a funkcí, např.:

```
type
  TMyRecord = record
    var
      Red: Integer;
    procedure printRed();
end;
var zaznam: TZaznam;
zaznam.iPolozka1 := 10;
```

Často se používá s klíčovým slovem **with**, kdy jakoby vytkneme proměnnou a pak jen pracujeme s položkami:

```
with zaznam do
begin
 iPolozka1 := 10;
end;
```

**Procedurální typy**

Tento (nejtěžší) typ se používá pro kontrolované volání a předávání funkcí. Jedná se o základní prvek VCL knihovny, kdy všechny obsluhy (např. ButtonClick) jsou deklarovány jako procedurální typ (ale IDE delphi to skrývá). Mimo to používáme např. pro volání funkcí v DLL knihovnách.

property OnClick: TNotifyEvent
kde deklarace
TNotifyEvent = procedure(Sender: TObject) of object;
tj. metoda objektu s jedním parametrem (Sender)
tj. pak
  procedure Button1Click(Sender: TObject);

Výhodou tohoto přístupu je, že můžeme deklarovat proměnnou typu TNotifyEvent, případně parametr a pak s ním pracovat, předávat atd. a až je třeba tak funkci zavolat.

Např.: mTest(Button1Click); kde

```
procedure TForm1.mTest(pClick: TNotifyEvent);
begin
   pClick(Self); // zavolame predanou obsluhu
end;
```

**Pole**

pole je strukturovaný datový typ skládající se z některého z předchozích typů (včetně např. záznamu nebo stringu). Jednotlivé položky jsou uloženy za sebou. Pole na rozdíl od jiných jazyků nemusí začínat 0, jediné omezení je, že interval musí být ordinálního typu (tedy i výčet nebo znak)

```
type  TPole = array [1..10] of integer;
var pole: TPole;
pole[1] := 123;
```

Ještě existuje možnost nespecifikovat limit, pak je takové pole nazýváno otevřené. Dá se do něj předat pole libovolné velikost.
Velikost pole se dá zjistit pomocí volání Length, meze Low a High.

```
function Sum( var X: array of Double): Double;
var
  I: Word;
  S: Double;
begin
  S := 0;
  for I := 0 to High(X) do S := S + X[I];
  Result := S;
end;
```

## Příklad:

Vytvořte jednotku (unit) pro práci s polem komplexních čísel.

TComplex = record im, re:double; end;

Jednotka by měla mít nejméně dvě funkce, Add a Sub (sečte nebo odečte všechny čísla v poli), parametrem je pole TComplex a výsledkem je jedno číslo TComplex.

Otestovat voláním z hlavního programu.