Obsah

1 Jednoduché datové typy

Jednoduché datové typy

- Typy, které obsahují jednu „informaci“ – jeden údaj; číslo, písmeno, ...
- Číselné typy (celá čísla, čísla v plovoucí řádové čárce)
- Znakový datový typ
- Pravdivostní datový typ – `boolean`
  - Může obsahovat jen 2 možné hodnoty – True (pravda) a False (nepravda)

Další jednoduché datové typy

- Datový typ *výčet*

```
var obdobi:(jaro,leto,podzim,zima);
```

- Datový typ interval

```
var cislice:0..9;
    mala_pismena:'a'..'z';
    tepla_obdobi:jaro..podzim;
```

- Ordinální datové typy– typy, které mají hodnoty „očíslované“
  - Celočíselné typy, pravdivostní typ, znakový typ, výčtové typy
  - Funkce Inc/Dec – zvětší/zmenší položku o 1
  - Funkce Pred/Succ – vrátí předchůdce/následníka položky

2 Strukturované datové typy

Strukturované datové typy

- Typy, které obsahují více informací než jednoduché typy
- Skládá se z jednodušších datových typů (ať už jednoduchých nebo strukturovaných)
- Pole
- Záznam
- Množina
- Soubor

2.1 Pole

Datový typ pole

- Skládá se z pevného počtu položek stejného typu
- Položky se rozlišují pomocí číselných indexů, u položek není možné najít jejich jiné pojmenování
- Mezi položkami dat a hodnotami indexů je jednoznačné přiřazení
- Definice obsahuje velikost pole a typ datové položky

```
var pole:array[1..100] of integer;
```

Zápis typů bez deklarace proměnné

```
type TCislo=integer;
     TPole=array[1..10] of char;
     TPoleCisel=array[1..10] of TCislo;
     TPolePoli=array[1..5] of TPoleCisel;

var Cislo:TCislo;
    Pole:TPolePoli;
```

Použití polí

- Jediná operace definovaná pro pole je označení položky – *selektor pole*
  - Touto operací získáme příslušnou položku pole

```
var telefon:array[1..9] of byte;
...
telefon[1]:=5;
WriteLn('4. cislice cisla je: ', telefon[4]);
```

Použití polí

Lze provádět i kopírování celých polí:

```
type TTelefon=array[1..9] of byte;
var telefon1, telefon2: TTelefon;
...
telefon2:=telefon1;
```

Tento zápis odpovídá kopírování člen po členu:

```
telefon2[1]:=telefon1[1]; ...
```

Pozor na kompatibilitu typů!

```
var telefon3:array[1..9] of byte;
...
telefon3:=telefon1;
```

Pole

- Indexy polí mohou být libovolné ordinální typy:

  ```
  var a:array[-100..100] of integer;
      b:array[1000..2000] of char;
      c:array['a'..'z'] or string;
  ```

- Vícerozměrná pole:

  ```
  var d:array[1..10,1..10,1..10] of integer;
      e:array[1..10] of array[1..10] of integer;
  ...
  d[1][2][3]:=1;
  ```

- String je speciální typ pole, index od 0 (zde je délka)
  - Standardně obsahuje 255 znaků, ale jejich počet lze snížit:

    ```
    var s:string[10];
    ```

2.2 Záznam

Datový typ záznam

- Skládá se z určitého množství položek, které mohou být různého typu
- Každá položka má své jméno – *identifikátor položky*

```
type TDatum=record
       den:1..31;
       mesic:1..12;
       rok:integer;
     end;
var datum:TDatum;
...
datum.den:=28;
datum.mesic:=3;
datum.rok:=2006;
```

2.3 Množina

Datový typ množina

- Je složen z ordinálních typů (bázový typ)
- Uchovává v sobě, zda se daná hodnota v množině vyskytuje nebo ne
- Bázový typ může obsahovat maximálnř 256 hodnot

```
type TBarva=(cervena, modra, zelena, zluta);
     TOdstin=set of TBarva;
```

Datový typ množina – přiřazení

- Konstruktor množiny (množinový literál) se zapisuje do hranatých závorek

```
var barva:TOdstin;
...
barva:=[];  Přiřazení prázdné množiny
barva:=[cervena, modra];
```

Datový typ množina – operace

- Rovnost množin – `=`

  ```
  odstin1=odstin2
  ```

- Nerovnost množin – `<>`
- „Je podmnožinou“ – `<=`
- „Je nadmnožinou“ – `>=`
- Sjednocení množin – `+`

  ```
  [cervena]+[modra] = [cervena, modra]
  ```

- Rozdíl množin – `-`
- Průnik množin – `*`
- Je prvkem – `in`

```
modra in odstin
```

3 Rekurze

Rekurze

- Patří mezi složitější programátorské konstrukce
- Odpovídá matematické indukci
- Pro výpočet volá funkce (procedura) sama sebe
- Nutnost ukončovací podmínky
- Např.: Faktorial: $n! = n\,(n-1)!, 1! = 1$ Fibonacciho číslo: $F_n = F_{n-1} + F_{n-2}, F_0 = 0, F_1 = 1$

4 Strategie návrhu programu

4.1 Návrh programu shora dolů

Návrh programu shora dolů

Problém se snažíme rozdělit na jednodušší podproblémy.

Takto iterativně pokračujeme, dokud nejsou všechny problémy jednoduše vyřešitelné.

Návrh programu shora dolů – klady a zápory

Výhody:

- Od začátku máme přehled o celku (vidíme les a hledáme stromy)

Nevýhody:

- Schopnost odhadnout správné rozčlenění
- Nemožnost paralelního návrhu

4.2 Návrh programu zdola nahoru

Návrh programu zdola nahoru

Od začátku mám (nebo jsem schopný udělat) základní bloky.

Jednotlivé bloky skládám do větších celků, dokud se nedostanu k řešení původního problému.

Návrh programu zdola nahoru – klady a zápory

Výhody:

- Od začátku máme stromy a z nich tvoříme les
- Rychlost návrhu (paralelní návrh)

Nevýhody:

- Vysoký stupeň abstrakce – potřeba mít programátorské zkušenosti
- Nepřehledný návrh ve většině fází vývoje
- Některé části návrhu se objevují postupně