Strukturované datové typy

Datový typ množina

V jazyku Delphi lze pracovat s datovými objekty, jejichž hodnotami jsou množiny. Popis typu množina má tvar **set of bázový typ**, kde bázový typ může být pouze označení ordinálního typu (identifikátor typu i popis typu) a určuje se jím typ prvků množiny.

Příklad:
```
Type Barva = (cervena, zluta, modra, zelena, seda);
     Odstin = set of Barva;
var Odstin1, Odstin2 : Odstin;
```

Hodnoty proměnné typu množina se konstruují podobně jako v matematice, tj. výčtem hodnot jednotlivých prvků množiny. Konstrukci výčtu hodnot předepisuje konstruktor množiny. Pro proměnné typu množina jsou definovány obvyklé množinové a relační operace:

+ sjednocení
* průnik
- rozdíl
= rovnost
<> nerovnost
>= obsahuje
<= je obsažena v

Operandy těchto operací však nemohou být libovolné množiny, ale pouze množiny s kompatibilními bázovými typy. Typ výsledku operací + - a * je dán typem operandů, typ výsledku relačních operátorů je Boolean.

Příklad [cervena]+Odstin1;

Relaci je prvkem množiny označuje operator in, který má stejnou prioritu jako relační operátory. Výrazy vytvořené pomocí tohoto operátoru mají tvar prvek in mnozina, kde prvek a množina jsou jednoduché výrazy, přičemž bázový typ množiny na pravé straně musí být kompaktibilní s typem prvku na levé straně. Hodnota je True, jestliže hodnota prvku patří do množiny, jinak je hodnotou false.

Příklad: cervena in Odstin1

Při psaní přiřazovacích příkazů, jimiž se definují hodnoty proměnných typu množina, je třeba dodržovat požadavek kompaktibility přiřazované hodnoty s typem proměnné.

Datový typ pole

Pole je homogenní datová struktura skládající se z položek stejného datového typu, které se vzájemně rozlišují pomocí indexu. Jako indexy slouží hodnoty určitého typu, který nazýváme typem index pole a který je spolu s typem složek pole stanoven popisem typu pole. Popis typu pole má tvar **array [typ indexu] of typ složky pole**
Počet složek pole je dán počtem hodnot indexu. Indexem bývá často typ interval, integer, v každém případě musí být ordinální. Na typ složek pole se neklade žádné omezení, může to být libovolný typ. Každý typ pole může být zaveden a deklarován v popisu typu pole například:

```
Type Vektor = array [1..3] of Integer;
     Veta   = array[1..3] of Char;
```

Proměnné těchto typů se deklarují obvyklým způsobem:

```
var A, B : Vektor;
    C : Veta;
```

Proměnná typu pole může být deklarována bez popisu typu:

```
var A:= array[1..3] of integer;
```

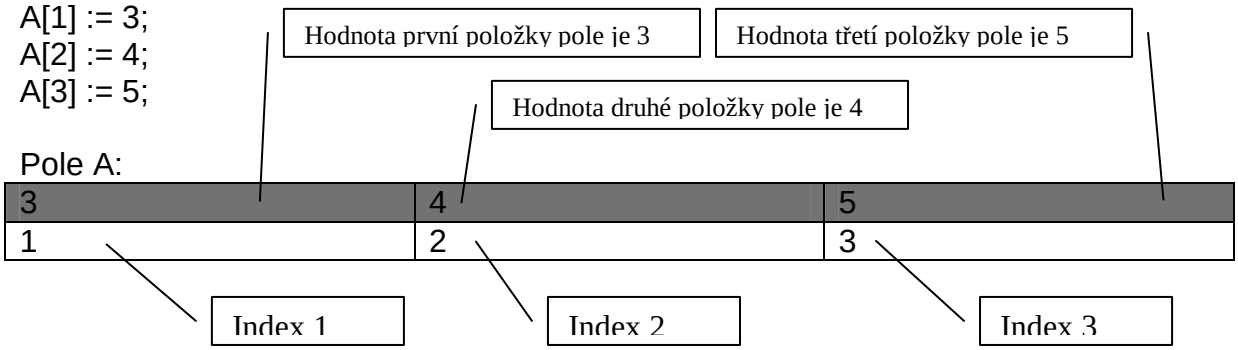
Proměnná A je typu Vektor a Vektor je typ pole o 3 složkách, do každé z těchto složek se bude ukládat celé číslo. Proměnnou typu pole si lze představit jako souvislý blok paměti, který je rozdělen na menší části, do nichž se vedle sebe ukládají jednotlivé hodnoty. Mohli bychom také říct, že je to vlastně jakási tabulka, kde každé políčko představuje jednu proměnnou jak to ukazuje obrázek.

1 hodnota	2 hodnota	3 hodnota
1. index	2. index	3. index

S každou proměnnou můžeme pracovat jako s celkem nebo můžeme označit nějakou položku pole. K jednotlivým položkám pole se přistupuje pomocí indexů.

Příklad označení položky pole proměnné A:

```
A[1] := 3;
A[2] := 4;
A[3] := 5;
```



Označování položek proměnných typu pole je jedinou operací, která je pro proměnné typu pole definována. Tuto operaci nazýváme selektorem pole a jejím výsledkem je odkaz na příslušnou složku pole. Selektor pole patří mezi zvláštní operace, které

nazýváme přístupovými operacemi. Hodnoty proměnných typu pole se obvykle definují a mění postupně tak, že se přiřazují hodnoty jednotlivým složkám pole. Přiřazovacím příkazem je však možné definovat i celou hodnotu proměnné typu pole, v tom případě musí být na pravé straně přiřazovacího příkazu uvedena proměnná téhož typu. Je povolen příkaz :

B := A;

to je ekvivalentní příkazům:

B[1] := A[1]; B[2]:= A[2]; B[3] := A [3];

Používáme – li v programu ještě proměnnou **var** W : **array** [1..3] **of** Real;, pak příkaz W := A; není povolen, protože W není typu vektor!!! Kompatibilita přiřazenín se u proměnných typu pole zužuje na požadavek totožnosti typu.

Dynamické pole

Výrazným nedostatkem jazyka Pascal a prvních verzí Delphi byla konstantní velikost polí, kterou nejenom že nebylo možné měnit během programu, ale dokonce jejich velikost musela být známa již v době překladu. Nevýhodou je malá pružnost použití. Dynamické pole nemá pevně stanovenou délku a deklaruje se **array** of základní typ ;

Příklad:

```
type
   DynamickePole = array of byte;

var DPole: DynamickePole;
```

Takto deklarované pole nemá zatím žádnou velikost a tudíž v paměti nezabírá žádný prostor. Proto před prvním použitím je nutné nastavit jeho velikost a alokovat tím volný blok paměti. K tomu slouží procedura SetLength.

Dpole[0] := 3; //Toto je hrubá chyba, v paměti dosud nebylo alokováno místo!!!!

SetLength (Dpole,10); {Nastaví velikost proměnné typu dynamické pole Dpole na 10 prvků. To znamená, že v paměti se zabere prostor pro 10 hodnot typu Byte}
Dpole[0] := 3; //Do 0 – tého prvku pole uloží hodnotu 3. Nyní je vše v pořádku.

Indexy u dynamických polí začínají vždy od nuly, takže předchozí pole má indexy v rozsahu 0..9. Pokud jde o překročení povoleného rozsahu indexů, na rozdíl od statických polí se při překladu negeneruje chyba. Je plně v kompetenci programátora, aby napravil správné indexování. Velikost pole, jeho nejvyšší a nejnižší index, je možné zjistit pomocí standardních funkcí Sizeof, Low a High. Funkce Sizeof vrací hodnotu velikosti pole. Low vrací vždy 0 a High velikost pole – 1.
Výrazným rozdílem mezi statickými a dynamickými poli je způsob přiřazování polí. Pokud se přiřadí statické pole do nové proměnné, jeho obsah se zkopíruje a v paměti se vyskytují 2 stejná pole. Jinak tomu je u dynamických polí. Tam se do nové proměnné přiřadí pomocí ukazatele pouze odkaz na existující pole. Proto se po provedení kódu:

```
var A,B: array of Byte; //deklarace proměnné typu dynamické pole
begin
  SetLength(A,10);  //Nastavení velikosti pole na 10 prvků
  for i := 0 to 9 do A[i]:=1; //Načtení do 10 prvků pole hodnotu 1
  B:=A; //Do pole B se uloží odkaz na pole A
  B[1]:=3; //Na pozici B[1] se uloží 3, ale tím se změní také i hodnota A[1]!!!!
end;
```

v proměnné A na pozici A[1] objeví 3, přestože u statických polí by tam byla stále 1. Pokud má být pole dynamické, je potřeba, aby bylo možné měnit jeho velikost i během programu. Ke zvětšení délky řetězce lze s úspěchem znovu použít proceduru SetLength.

Příklad:
SetLength(A,10); //Nastaví délku pole na 10
SetLength(A,20); //Zvětší délku pole na 20

Pokud se ukáže, že zvětšení délky pole bylo velké, je možné jeho velikost zmenšit pomocí funkce Copy.

A:=Copy (A,0,5); //V poli A zůstane pouze 5 prvků od pozice 0. Výsledná délka pole je tedy 5 a ostatní prvky se odříznou.

Dvojrozměrné pole

Je – li typem prvku pole opět pole vzniká dvojrozměrné pole.

Příklad:
var Mat: **array**[1..3,1..3] **of** String;

Chceme – li v programu definovat dvojrozměrné pole obsahující N*N prvků typu string, můžeme tak učinit deklarací:

type Matice = **array** [1..N] of **array** [1..N] **of** string;

nebo zkráceným zápisem:

type Matice = **array** [1..N,1..N] **of** string;

Proměnné tohoto typu budou mít v obou případech stejné vlastnosti. Ilustrujme přístup ke složkám vícerozměrného pole na příkladu proměnné typu matice

var Mat : Matice;

Pak přístup k položce vícerozměrného pole má tvar:

Mat [i][j]; nebo zkráceně Mat[i,j];

Kde slovy Mat[i,j] je prvek matice Mat, který leží v I – tém řádku a j – tém sloupci.

Grafický příklad:

Mat

	j=1	j=2	j=3
i=1	Mat[1,1]	Mat[1,2]	Mat[1,3]
i=2	Mat[2,1]	Mat[2,2]	Mat[2,3]
i=3	Mat[3,1]	Mat[3,2]	Mat[3,3]

Výhody polí:

- Možnost uchovávat více hodnot stejného typu v jedné proměnné.
- Snadná manipulace s daty pomocí indexů.
- Možnost zvolit rozsah indexů pole např. 1..5 nebo 132..136.

Nevýhody polí:

- Nevýhodou je statická struktura, to znamená , že velikost pole nelze měnit během programu. Proto se musí hned na začátku nadeklarovat na maximální velikost i přesto, že většina pole nebude využitá a zábírá pouze paměťový prostor.
- Velikost pole musí být známa během překladu to znamená, že za běhu programu nelze velikost pole měnit ani ji nastavit.
- Pozn.: u komponenty Stringgrid je index řádku zaměněn s indexem sloupce.

Datový typ řetězec

Datový typ řetězec se deklaruje pomocí klíčového slova String.

Příklad:
var s: String;

Řetězcům lze přiřadit řetězcové konstanty:

```
s :=  'text';
s :=  ' '; //Řetězec obsahuje prázdný znak
s :=  ''; //Řetězec neobsahuje žádný znak
```

K řetězci lze přistupovat podobně jako k poli:

```
var s : String;
      c : Char;
...
 s :=  'text';
 c:= s[1]; //V proměnné c bude uložen znak 't'
```

Řetězce lze spojovat pomocí operátoru +.

Příklad:
```
var s, s1, s2 : string;
...
s1 :=  'řetězce';
s2 :=  'lze spojovat';
s3 := s1+s2; //Řetězec s bude obsahovat text řetězce lze spojovat
```

Délku řetězce lze zjistit pomocí funkce Length.

Příklad:
```
var s : String;
    d : Byte;
...
s := 'text dlouhý 20 znaků';
d := Length (s); //V proměnné d bude uložena  hodnota 20
```

Delphi rozpoznává 3 typy řetězců:
ShortString
AnsiString
WideString

My probereme pouze Shortstring.
Jedná se o statické pole s délkou 256 znaků. Hlavní nevýhodou je, že při deklaraci se v paměti vyhradí tak velký blok, aby bylo do něj možné zapsat řetězec o maximálním počtu znaků t.j. 256.

9	d	o	b	r	ý		d	e	n	
0	1	2	3	4	5	6	7	8	9	10

Na 0 – té pozici je uložena délka řetězce – to je počet znaků řetězce.

Příklad deklarace řetězců:
```
var   s : String; //Řetězec má v paměti alokován prostor pro 256 znaků
      s1 : String [100]; //Řetězec má v paměti alokován prostor pro maximálně 100 znaků
```

Toto omezení použijeme v případě, když budeme dopředu vědět, že délka řetězce nepřekročí 100 znaků. Šetříme tím paměť.

Datový typ záznam

Záznam je nehomogenní datová struktura skládající se z určitého počtu pojmenovaných složek, které mohou být různého typu a které nazýváme položkami záznamu. Jména a typy položek se specifikují v části type. Obsahuje – li záznam více položek stejného typu oddělují se čárkou.

Příklad definice datového typu záznam:

```
type    TDatum  = record   //Deklarace datového typu záznam
                    Den : 1..31;
                    Mesic: 1..12;
                    Rok: 1..2000
                end;

var Datum : TDatum;    //Deklarace proměnné typu záznam

type TOsoba = record          //Deklarace datového typu záznam
        Jmeno    : Strring[20]; //Položka záznamu obsahuje maximalně 20znaků
        Prijmeni : Strring[20]; //Položka záznamu obsahuje maximalně 20 znaků
        Vek      : Byte; //Položka záznamu může obsahovat číslo od 1 do 255
    Rok narozeni : 1..2000;  //Položka záznamu může obsahovat číslo od 1 do 2000
                end;

var Osoba : TOsoba;  //Deklarace proměnné typu záznam
```

Přístup k jednotlivým položkám proměnných typu záznam se provádí pomocí selektoru záznamu, jehož zápis obsahuje označení proměnné typu záznam a identifikátor příslušné položky. Jestliže Osoba je proměnná typu záznam pak Osoba.jmeno identifikuje položku typu záznam.

Zanořený záznam

```
type Data = record
                jmeno    : String;
                prijmeni : String;
        end;

type TOsoba = record
            student : Data; //Položka student je typu Data a jedná se o zanořený
záznam
             vek      : byte;
          end;

var a : array [0..4] of TOsoba;
      i : integer;
     K: integer;
```

Jeli položka záznamu opět typu záznam jedná se o zanořený záznam.

Příklad:
Položka student je typu Data a typ data představuje zanořený záznam. Deklarace položek zanořeného záznamu musí předcházet deklaraci položek typu zanořený záznam. Přístup k položce zanořeného záznamu je definován selektorem záznamu a to tak, že zanořená položka záznamu je oddělena další tečkou.

Příklad:
a[0].student.jmeno identifikuje první položku zanořeného záznanamu v 0 – tém indexu pole
a[0].student.prijmeni identifikuje druhou položku zanořeného záznanamu v 0 – tém indexu pole
a[1].student.jmeno identifikuje první položku zanořeného záznanamu v 1 indexu pole
a[1].student.prijmeni identifikuje druhou položku zanořeného záznanamu v 1 indexu pole

Povolené operace se zanořeným záznamem:
1. Zápis hodnoty do proměnné typu záznam po jednotlivých položkách
2. Výpis hodnoty proměnné typu záznam po jednotlivých položkách

Příkaz Width

Při práci s proměnnými typu záznam se často stává, že v poměrně malém úseku programu použijeme nějakou položku jednoho záznamu několikrát nebo použijeme několik položek téhož záznamu. Příkladem takového opakovaného přístupu k jednomu záznamu je trojice příkazů, kterou se postupně definují hodnoty položek proměnné Dnes. Tato 3 přiřazení lze zkráceně zapsat pomocí příkazu **width**:

```
type TDnes = record
                Den : 1..31;
              Mesic : 1..12;
                Rok : 1..2050
             end;

var Dnes : TDnes;
...
width Dnes do
             begin
                  Den  :=  21;
                  Mesic :=  3;
                  Rok := 2006
             end;
```

Ekvivalentní zápis bez použití příkazu **width** má tvar:
Dnes.den := 21;
Dnes.mesic := 3;
Dnes.Rok := 2006;

Uvnitř příkazu, který je vnořen do příkazu **width**, je možné označovat položky proměnné pouze s uvedením příslušné položky identifikátoru položky. Významnou vlastností příkazu width je, že přístup k proměnné typu záznam se v něm provádí pouze jednou. Tímto příkazem lze proto urychlit zpracování záznamů, které jsou složkami polí.

Datový typ soubor

Souborem dat rozumíme uspořádanou množinu dat uloženou mimo operační paměť počítače. Soubory v Delphi jsou abstrakcemi skutečných souborů a program proto neobsahuje žádné informace o fyzických vlastnostech souborů. V jazyku Delphi se soubor chápe jako posloupnost složek stejného typu, pro niž jsou definovány určité operace. Tyto operace umožňují sekvenční zpracování souborů. To znamená postupné vytváření souboru nebo postupné čtení jeho složek. V každém okamžiku zpracování souboru je přístupna jediná jeho složka.

Z hlediska programátora je možné soubory rozdělit na 3 základní skupiny:
- Binární soubory s udaným typem
- Netypové binární soubory
- Textové soubory

Binární soubory s udaným typem:
var Fin : file of datový typ složky souboru;

Typ složky souboru může být libovolný typ s výjimkou typu soubor. Logické vlastnosti souboru jsou určeny jeho typem.

Příklad deklarací:
var Fin : **file of** Integer;

Fin je proměnná typu soubor, jejíž položku tvoří celé číslo. Vstupní soubory, do kterých se zapisuje se standardně pojmenovávají Fin. Výstupní soubory, ze kterých se čte se standardně pojmenovávají Fout.

Rozlišujeme dva způsoby zpracování souboru:
- Vytváření souboru, neboli zápis do souboru po jednotlivých položkách.
- Prohlížení souboru, neboli výpis obsahu souboru po jednotlivých položkách.

Klíčové procedury pro operace s proměnnou typu soubor:
Procedura **AssignFile**(F, 'C:\data.dat ');
Dříve než začneme se souborem pracovat, musíme nějakým způsobem svázat proměnnou typu soubor s nějakým existujícím souborem, který je uložen na disku. K tomu je určena standardní procedura AssignFile.Tato procedura má 2 parametry. Prvním je libovolná proměnná typu soubor, druhým parametr je typu string a udává skutečné jméno souboru. Druhý parametr je vlastně přístupovou cestou k souboru na disku.

AssignFile (F, 'C:\data.dat ');
Procedura AssignFile přiřadí k proměnné typu soubor externí soubor na disku C:\.
Všechny operace, které provedeme s proměnnou F se budou provádět se souborem data.dat na disku C. Nyní již můžeme přistoupit ke čtení nebo zápisu do souboru.

Procedura **Rewrite**(F);
Zápis do souboru a vytvoření souboru procedurou Rewrite (F);
Procedura Rewrite (F) vytvoří soubor zadaný procedurou AssignFile a odpovídající proměnnou F, která reprezentuje soubor, pozice v souboru je nastavena na první

prázdnou položku. Soubor je připraven pro zápis první položky. Po zápisu první položky se pozice automaticky přesune na druhou prázdnou položku atd. Procedura Rewrite (F) je velice nebezpečná. Pokud ji totiž použijete na existující soubor, nejprve jej smaže, potom vytvoří nový soubor stejného jména a bez sebemenšího varování Vás připraví o uložená data. Pokud je soubor otevřený můžeme do souboru zapisovat po jednotlivých položkách tedy sekvenčně.

Procedura **Write**(F,Hodnota);
Pro zápis do souboru je určena procedura Write (F, hodnota1). Procedura zapíše do souboru F hodnotu proměnné hodnota1. Proměnná hodnota1 musí být stejného typu jako položka souboru!!

Procedura **Reset**(F);
Výpis obsahu souboru, tedy čtení ze souboru, se provádí po jednotlivých položkách. Procedura Reset (F) otevře soubor pro čtení. Existující soubor se otevře procedurou Reset(F) za předpokladu, že již byla použita procedura AssignFile, krerá přiřadí proměnné typu soubor fyzicky existující soubor na disku. Když použijeme příkaz Reset (F) na již otevřený soubor, je tento soubor nejdříve uzavřen a potom znovu otevřen. Pozice v souboru bude nastavena na první položku, která je připravena ke čtení.

Procedura **Read** (F,X);
Pro čtení položky souboru je určena procedura Read (F,X); Příkaz přečte položku ze souboru a uloží do proměnné X. Proměnná X musí být stejného typu jako položka souboru!!

Procedura **CloseFile**(F);
Po skončení práce se souborem nesmíme zapomenout soubor zavřít. K tomu slouží procedura CloseFile(F);

Funkce **eof** (F);
Funkce eof(F); testuje jestli se ukazatel položky souboru nachází na konci souboru.
Je – li ukazatel na konci souboru, funkce eof(F) vrátí hodnotu 1, pokud není na konci souboru, pak vždy vrátí hodnotu 0. Pro sekvenční zápis a čtení položek souboru je bezpodmínečně nutné použít jednotlivé procedury ve správném pořadí jak ukazuje příklad. Pořadí použitých procedur nelze zaměnit!!!

Příklad:

```
var   F : file of Integer;
      X : Integer;

implementation

{$R *.dfm}

//tato procedura demonstruje zápis jedné položky do souboru
procedure TForm1.Button1Click(Sender: TObject);
begin
X := StrToInt(Edit1.Text); {Přečte zadanou hodnotu z komponenty edit1  do prom. X}
AssignFile(F,'I:\data\text.txt'); {Vytvoří propojení mezi proměnnou F a skutečným
souborem na disku I:\data\text.txt}
Rewrite(F); {Vytvoří soubor text.txt a jeho reprezentaci pomocí proměnné F
a nastaví ukazatel na první prázdnou položku do které je očekáván zápis položky}
Write(F,X); {Zapíše obsah proměnné x do první položky souboru a zároveň
se vytvoří 2 prázdná položka souboru, na kterou se nastaví ukazatel
a do které můžeme zapisovat další hodnotu}
CloseFile(F); {Uzavře soubor}
end;

//Tato procedura demonstruje výpis obsahu jedné položky souboru
procedure TForm1.Button2Click(Sender: TObject);
begin
AssignFile(F,'I:\data\text.txt');
reset(F); {Otevře soubor pro čtení položek a ukazatel položky nastaví
na 1 položku}
Read(F,X); {Přečte obsah první položky souboru do proměnné X a ukazatel položky
automaticky nastaví na druhou položku}
Edit2.Text:=IntToStr(X); //Obsah proměnné X zobrazí v komponentě Edit2
CloseFile(F); //Uzavře soubor
end;
```