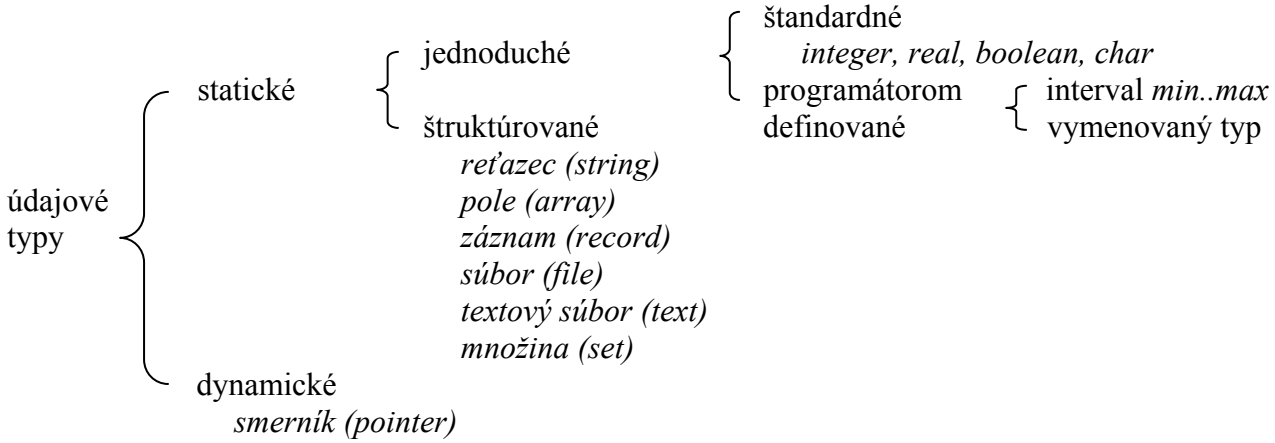
Rozdelenie údajových typov



Ordinálne sú typy integer, boolean, char a programátorom definované typy. Platia pre ne operácie succ, pred a ord.

Štandardné údajové typy:

Meno typu: **Integer** (ordinálny typ)
Množina hodnôt: celé čísla z intervalu -2 147 483 648 po 2 147 483 647 (konštanta MaxInt)
Dovolené operácie: + (sčítanie), - (odčítanie), * (násobenie), div (celočíselné delenie), mod (zvyšok po celočíselnom delení)
Funkcie: abs(x), sqr(x), odd(x), trunc(x), round(x), succ(x), pred(x), ord(x)
Relačné operátory: <, <=, =, <>, >=, >
Vysvetlivky:
Ak nám uvedený rozsah nevyhovuje, môžeme použiť ďalšie celočíselné typy, ktorých rozsahy si môžete pozrieť pomocou kontextového helpu F1 s kurzorom v slove integer.
U celočíselných typov nie je dovolené klasické delenie, lebo jeho výsledkom nemusí byť celé číslo. Dovolené je tzv. celočíselné delenie (celočíselný podiel) div a zvyšok po celočíselnom delení mod. Tieto funkcie si treba ozrejmiť, lebo sa v numerických algoritmoch a programoch často využívajú. Napríklad 15 div 6 = 2 a 15 mod 6 = 3 lebo 15:6 = 2 zvyšok 3, alebo 7 div 10 = 0 a 7 mod 10 = 7 lebo 7:10 = 0 zvyšok 7, alebo 21 div 3 = 7 a 21 mod 3 = 0 lebo 21:3 = 7 zvyšok 0. Všeobecne pre kladné celé čísla A a B platí: Nech A div B = C a A mod B = D potom B.C + D = A a 0<=D<B. Pre záporné celé čísla funkcie div a mod v Pascale pracujú ináč ako v matematike!
Výsledkom:
abs(x) je absolútna hodnota čísla x, napr. abs(8) = 8, abs(-5) = 5, abs(0) = 0
sqr(x) je druhá mocnina čísla x, napr. sqt(8) = 64, sqr(-5) = 25, sqr(0) = 0
odd(x) je true (pravda), ak x je nepárne číslo, inak false (nepravda), napr. odd(0) = false
trunc(x) je celá časť reálneho čísla x, napr. trunc(8.9) = 8, trunc(-5.6) = -5, trunc(0.5) = 0
round(x) je zaokrúhlené reálne číslo x, napr. round(8.9) = 9, round(-5.6) = -6, round(0.5) = 1
succ(x) je nasledovník x, napr. succ(5) = 6, succ(-1) = 0, succ(-5) = -4
pred(x) je predchodca x, napr. pred(5) = 4, pred(-1) = -2, pred(1) = 0
ord(x) je poradové číslo x, napr. ord(5) = 5, ord (0) = 0, ord (-12) = -12

Meno typu: **Real** (nie je ordinálny typ!)
Množina hodnôt: niektoré reálne čísla z intervalu v absolútnej hodnote približne $5{,}0.10^{-324}$ po $1{,}7.10^{308}$ s presnosťou na 15-16 platných cifier.
Dovolené operácie: +, -, *, / (delenie)
Funkcie: abs(x), sqr(x), sqrt(x), trunc(x), round(x), int(x), frac(x), sin(x), ln(x), exp(x), arctan(x)
Relačné operátory: <, <=, =, <>, >=, >

Vysvetlivky:
Keďže reálnych čísel v matematike je nekonečne veľa (aj medzi dvoma ľubovoľnými reálnymi číslami) a pamäť počítača je konečná, počítač si dokáže zapamätať len niektoré reálne čísla.
Niektoré funkcie sú popísané vyššie, u typu integer, a málo používané funkcie nepopíšeme.
Výsledkom:
sqrt(x) je druhá odmocnina z nezáporného čísla x, napr. sqrt(9) = 3, sqrt(2) = 1,4142135624
int(x) je celá časť (cifry pred desatinnou bodkou) z reálneho čísla x, napr. int(123.456) = 123
frac(x) je desatinná časť z reálneho čísla x, napr. frac(123.456) = 0.456
exp(x) je e^x
ln(x) je prirodzený logaritmus kladného čísla x[1]

Meno typu: **Boolean** (ordinálny typ)
Množina hodnôt: {False, True}
Dovolené operácie: not (negácia), and (logický súčin), or (logický súčet), xor (logický xor)
Funkcie: succ(x), pred(x), ord(x)
Relačné operátory: <, <=, =, <>, >=, >, in (je prvkom)
Vysvetlivky:
Ide o logické hodnoty False – nepravda, 0 a True – pravda, nie 0; false < true.
pred(true) = false, succ(false) = true; ord(false) = 0, ord(true) = 1

Meno typu: **Char** (ordinálny typ)
Množina hodnôt: znaky použitej kódovacej tabuľky
Funkcie: upcase(z), succ(z), pred(z), ord(z), chr(x)
Relačné operátory: <, <=, =, <>, >=, >, in
Vysvetlivky:
Prvých 32 znakov (0-31) je riadiacich. Znak s poradovým číslom 32 je medzera atď.
Výsledkom funkcie chr(x) je znak zodpovedajúci poradovému číslu x.
Napríklad succ('A') = B, pred('A') = @, ord('A') = 65 a chr(65) = A, chr(ord(z)) = z, ord(chr(x)) = x.
Všimnite si, že ak sa jedná o konkrétny znak, t.j. konštantu, musí byť v apostrofoch, podobne ako reťazcová konštanta.
Funkcia UpCase(z) zmení malé písmeno na veľké, na iné znaky nemá vplyv, napr. upcase ('b') = B.

Údajové typy, pre ktoré sú definované funkcie succ (successor - nasledovník), pred (predecessor - predchodca) a ord (ordinal - poradové číslo) nazývame **ordinálne** (každá hodnota má presne určené miesto). Sú to typy integer, boolean a char. V Pascale sú pre ne zavedené ešte dva praktické príkazy inc (increase - zväčšiť) a dec (decrease - zmenšiť). Príkaz inc(p) zväčší hodnotu premennej p o jednu pozíciu vpravo, napr. inc(5) = 6, inc('A') = B, inc(false) = true a opačne dec(p) zmenší hodnotu premennej p o jednu pozíciu vľavo, napr. dec(5) = 4, dec('B') = A, dec(true) = false. Tieto príkazy pracujú podobne ako funkcie succ a pred avšak môžu mať aj tvar inc(p,n) resp. dec(p,n), kde n je celé číslo (môže byť aj záporné) udávajúce, o koľko treba posunúť hodnotu premennej p. Napr. inc(5,10) = 15, inc('a',10) = k, dec(5,10) = -5, dec('a',5) = W.

Štruktúrované údajové typy

Meno typu: **string** alebo **string[n]** kde n je celé číslo od 1 po 255 – udáva maximálnu dĺžku reťazca
Množina hodnôt: reťazce zo znakov použitej kódovacej tabuľky
Dovolené operácie: + (spája reťazce)
Funkcie: length, copy, delete, concat, insert, pos
Relačné operátory: <, <=, =, <>, >=, >
Vysvetlivky:
Napríklad string[5] vyhradí v pamäti miesto pre päťznakový reťazec (pre najviac 5 znakov, ostatné nebudú zapamätané). Reťazec je v Object Pascale (Delphi) ukončený znakom nula (#0, ord(NulovýZnak)=0), pričom v Turbo Pascale je dĺžka reťazca zakódovaná v jeho 0. znaku.

[1] Posledné dve funkcie možno použiť na výpočet x^y pomocou výrazu exp(y*ln(x)), x kladné reálne a y reálne číslo.

Výsledkom funkcie length(r) je celé číslo - dĺžka reťazca r. Prázdny reťazec '' má dĺžku 0.
Pri relačných operáciách sa najprv porovnajú prvé znaky v porovnávaných reťazcoch, ak sú zhodné, druhé atď.

Jednorozmerné pole

Pole (Array) je štruktúrovaný údajový typ, ktorý sa skladá z prvkov rovnakého typu, pričom k prvku pristupujeme cez index – „pozíciu“ daného prvku v poli.

Statické pole

(„klasické pascalovské“)

Nevýhodou statického poľa je, že veľkosť poľa t.j. počet a typ prvkov poľa musí byť určený už pri spustení programu, aby sa mohla vyhradiť zodpovedajúca veľkosť pamäte a tá je rezervovaná, či sa plne využije alebo nie. Ak pred spustením programu nepoznáme presný počet prvkov poľa (veľkosť poľa), môžeme v Delphi použiť dynamické pole.

Definícia typu pole má tvar:

type *MenoTypuPole* = array [*RozsahIndexu*] of *TypPrvkuPola*;

pričom *RozsahIndexu* musí byť ordinálneho typu.

Pr.:
```
type
  tPole1 = array [1..10] of integer;   //Typ pole so štruktúrou 10 čísel typu integer
const
  MaxPocPrvkov = 100;
type
  tIndex = 0..MaxPocPrvkov;
  tPole2 = array [TIndex] of real;     //Typ pole so štruktúrou 101 čísel typu real
```

Deklarácia premennej typu pole má tvar:

var *IdentifikatorPremennej* : *MenoTypuPole*;

alebo

var *IdentifikatorPremennej* : array [*RozsahIndexu*] of *TypPrvkuPola*;

Pr.:
```
var A: TPole1;          //Premenná typu pole so štruktúrou 10 čísel typu integer
    B: array [ 1..MaxPocPrvkov] of Boolean;        //Premenná typu pole so štruktúrou
                                                   //100 hodnôt False alebo True
```

Sprístupnenie prvkov poľa sa realizuje zápisom:

MenoPola [*IndexovyVyraz*]

kde *IndexovyVyraz* je rovnakého typu (ordinálneho), ako je typ *RozsahIndexu*.

Pr.: A[1]:= 5; A[2*i-i]:= i; B[MaxPocPrvkov]:= True; B[i-1]:= not B[i];

Dynamické pole

Jeho definícia alebo deklarácia má tvar:

... array of *TypPrvkuPola*;

Pr.:
```
type
  tDynamickePole = array of integer;
var
  D: tDynamickePole;
  A: array of real;
```

Takto definované alebo deklarované pole nemá žiadnu veľkosť a nezaberá v pamäti priestor.
Pred prvým použitím premennej typu dynamické pole je potrebné nastaviť veľkosť poľa príkazom **SetLength**, čím sa alokuje (vyhradí) potrebná veľkosť z voľnej pamäte.

Pr.:
```
SetLength(D,100);     //V pamäti sa vyhradí miesto pre 100 čísel typu integer
SetLength(A,Pocet)    //V pamäti sa rezervuje miesto pre Pocet čísel typu real
```

Indexy dynamických polí začínajú vždy od 0 (nuly)! Po prekročení rozsahu indexov sa negeneruje chybové hlásenie! Odporúča sa používať štandardné funkcie **SizeOf**, **Low** a **High**.

Funkcia **SizeOf** vracia hodnotu: veľkosť poľa * veľkosť prvku poľa.
Napríklad v 32-bitových Delphi zaberá integer 4B => MaxInt = 2^{31} - 1 (v TP 2B), real 6B.
Funkcia **Low** vracia najnižší index poľa, pri dynamických poliach vždy 0.
Funkcia **High** vracia najvyšší index poľa, pri dynamických poliach: veľkosť poľa – 1.
Pr.: for i:= low(A) to high(A) do A[i]:= i; //analogicky: for i:= 0 to Pocet-1 do A[i]:= i;

Zväčšiť veľkosť dynamického poľa funkciou SetLength možno počas behu programu viackrát. Zmenšiť veľkosť poľa možno funkciou **Copy**, ktorá má tvar:
PremennaTypuPole := Copy (*PremennaTypuPole*, *PociatocnyIndex*, *PocetPrvkovPola*)
Pr.: A:= copy(A,0,5); //V poli A zostane 5 prvkov od pozície 0.
Dynamické pole sa zruší priradením hodnoty **nil**.
Pr.: A:= nil; //Uvoľní sa alokovaná pamäť.

Pamätaj!
Po priradení B:= A; kde A,B sú dynamické polia, každá zmena v jednom poli sa prejaví aj v poli druhom! Nevytvorí sa totiž kópia poľa A (ako pri statických poliach), ale do poľa B sa uloží len ukazovateľ na pole A.

Typizované konštanty

Ak sa hodnota objektu počas behu programu nemení, používame typizované konštanty.
Typizované konštanty definujeme:
const *IdentifikatorKonštanty* : *typ* = *hodnota*;
Pr.: const Meno: string = ‘Peter’;

Konštantné pole je typizovaná konštanta typu pole, ktorej hodnoty sú uvedené v okrúhlych zátvorkách, oddelené čiarkami.
Pr.: const A : array [1..10] of integer = (1,0,0,1,1,0,0,1,0,1);
Morse : array [‘A’..’Z’] of string[6] = (‘.-‘ , ‘-...‘ , ‘-.-.‘ ,..., ‘--..‘);
Farba : array [0..4] of TColor = (clWidth,clRed,clBlue,clYellow,clBlack);

Hodnoty premenných zadané pri ich deklarácii
Ak chceme, aby premenná mala už v úseku deklarácií priradenú počiatočnú hodnotu, môžeme tak spraviť zápisom:
var *IdentifikatorPremennej* : *typ* = *hodnota*;
Pr.: var Pole : array [1..7] of string = (‘Peter’ , ’Jana’ , ’Adam’ , ’Dalibor’ , ’Zuzana’ , ’Barbora’);
Na rozdiel od konštantného poľa možno zmeniť hodnoty poľa počas behu programu, napr. utriediť!

Otvorené pole ako parameter procedúry
Delphi umožňuje ako formálny parameter procedúry použiť pole bez udania veľkosti poľa, čo umožňuje vytvárať všeobecnejšie procedúry. V deklarácii procedúry sa uvedie len typ poľa a aktuálna veľkosť poľa, resp. najvyšší index poľa sa zistí pri skutočnom parametri poľa procedúrou Hight.

Indexy otvorených polí vždy začínajú 0 (nulou), t.j. funkcia Low vždy vracia 0 a funkcia High vracia hodnotu N-1, kde N je počet prvkov poľa. Prvok A[i] je pre Low(A) = 0 i+1. prvkom; pre Low(A) = 1 i-tym prvkom; pre Low(A) = 2 i-1. prvkom, atď.

```
Pr.: procedure NulujPole (var Pole: array of integer);
        var i: integer;
        begin
           for i:= 0 to High(Pole) do Pole[i]:= 0;
```

```
//  for i:= Low(Pole) to High(Pole) do Pole[i]:= 0;
end;
```

Pamätaj!
Parameter typu pole sa vždy snažíme nahradzovať odkazom (referenciou) a nie hodnotou, pri ktorej sa vytvára nová kópia poľa zaberajúca ďalšie miesto v pamäti.
Tak, ako nebolo v Pascale dovolené deklarovať statické pole pri písaní formálneho parametra, nie je to dovolené ani v Delphi, t.j. deklarácia napr.

```
procedure NulujPole (var A: array [1..20] of integer);
```

nie je dovolená (pri statickom poli je dovolené v deklarácii formálneho parametra použiť len meno definovaného typu pole).

Dvojrozmerné pole

V definíciách typu pole:

type mt = *array* [ti] *of* tz alebo *type* mt = *array of* tz

sme doteraz dosadzovali za typ zložky len jednoduchý typ alebo typ reťazec. V princípe typ zložky môže byť akýkoľvek, okrem typu súbor. V prípade, že typ zložky bude opäť typ pole, dostávame tzv. typ dvojrozmerné pole.
Definícia typu dvojrozmerné **statické** pole má tvar:

type mt = *array* [ti1] *of array* [ti2] *of* tz

kde mt je meno typu – identifikátor, ti1 a ti2 sú ordinálne typy indexov typu a tz je typ zložky.
Dovolený je aj skrátený zápis definície typu pole:

type mt = *array* [ti1 , ti2] *of* tz

Napríklad:

```
type   MATICA1 = array [1..10 ] of array [1..10] of integer;
       MATICA2 = array [1..10,1..10] of integer;  { definícia rovnocenná s MATICA1 }
       SACHOVNICA=array[´a´..´h´,1..8] of boolean;
       NEZMYSEL=array[boolean,boolean] of char;
```

Definícia typu dvojrozmerné **dynamické** pole má tvar:

type mt = *array of array of* tz

kde mt je meno typu – identifikátor a tz je typ zložky.
Pre prácu s dvojrozmerným dynamickým poľom platia rovnaké pravidlá a procedúry ako pre prácu s jednorozmerným dynamickým poľom, t.j. predovšetkým veľkosť poľa treba nastaviť pred jeho prvým použitím príkazom SetLength (pozri príklad nižšie).

Dvojrozmerné pole nazývame aj pole polí. Názov vznikol zrejme z poznatku, že jednorozmerné pole, resp. každý jeho prvok je znova jednorozmerným poľom. Z hľadiska štruktúry dvojrozmernému poľu zodpovedá matematický pojem matica.
Matica typu N x M („en krát em“) obsahuje N riadkov a M stĺpcov

- pre zmenu riadku, t.j. pohyb v stĺpci používame premenné Riadok, Row, I
- pre zmenu stĺpca, t.j. pohyb v riadku používame premenné Stlpec, Col, J
- schematicky (matica typu N x M):

Stlpec sa mení od 1 po M (pohyb v riadku, zmena stĺpca) →

Riadok sa mení od 1 po N (pohyb v stĺpci, zmena riadku) ↓

$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	…	$a_{1,j}$	…	$a_{1,m}$
$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	…	$a_{2,j}$	…	$a_{2,m}$
$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	…	$a_{3,j}$	…	$a_{3,m}$
…	…	…	…	…	…	…
$a_{i,1}$	$a_{i,2}$	$a_{i,3}$	…	$a_{i,j}$	…	$a_{i,m}$
…	…	…	…	…	…	…
$a_{n,1}$	$a_{n,2}$	$a_{n,3}$	…	$a_{n,j}$	…	$a_{n,m}$

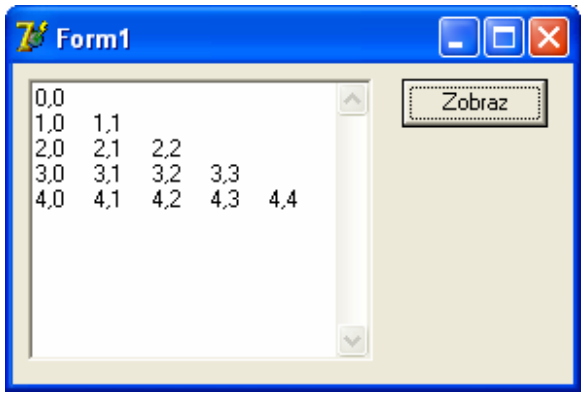
Z hľadiska algoritmických konštrukcií pre prácu s maticou väčšinou potrebujeme cyklus v cykle. Vonkajší cyklus zabezpečuje najčastejšie nastavenie príslušného riadku (for Riadok := 1 to

N do) a vnútorný cyklus pohyb v nastavenom riadku t.j. zmenu stĺpca od 1 po M (for Stlpec := 1 to M do).

Aj pri dvojrozmerných poliach pracujeme s indexovanými premennými a používame zápis, napríklad pre pole A: A [I][J] resp. skrátený zápis A[I,J] - prvok poľa A v I-tom riadku a J-tom stĺpci.

Príklad použitia dvojrozmerného **dynamického** poľa:

```
var A: array of array of string;
procedure TForm1.btZobrazClick(Sender: TObject);
var
  I, J : Integer;  Riadok: string;
begin
  SetLength(A, 5);          //počet riadkov poľa A
  for I := 0 to High(A) do
  begin
    SetLength(A[I], I+1);  //počet stĺpcov v I. riadku!
    Riadok:= '';
    for J := 0 to High(A[I]) do
    begin
      A[I,J] := IntToStr(I) + ',' + IntToStr(J) + '       ';
      Riadok:= Riadok + A[I,J];
    end;
    Memo1.Lines.Add(Riadok);
  end;
end;
```



Obrázok vznikne spustením procedúry btZobrazClic

Údajový typ záznam

Najmä pri hromadnom spracovaní údajov, pri informačných systémoch, často treba spojiť do jedného celku rôzne údajové typy (integer, string, boolean a pod.). Umožňuje to údajový typ záznam.

Záznam je nehomogénny štruktúrovaný údajový typ, ktorý sa skladá z pevného počtu položiek, vo všeobecnosti rôznych typov.

Definícia typu záznam má tvar:

```
type mt = record
            p1 : tp1;          kde mt je meno typu
            p2 : tp2;          p1 až pn sú identifikátory položiek záznamu a
            ...                tp1 až tpn sú typy jednotlivých položiek
            pn : tpn
          end;
```

Napríklad:

```
type TKarta = record
                OsCislo : integer;
                Meno,
                Priezv : string[25];
                DatNar : record
                           Den : 1..31;
                           Mes : 1..12;
                           Rok : 1900..2100
                         end;
                Adresa : record
                           Obec,
                           Ulica : string[25];
                           PSC : string[5]
                         end;
                Priemer : array [1..5] of real;
                Moze : boolean
```

```
                    end;
type TBod = record
                    x,
                    y : real;
                    farba : 0..15;
                    blik : boolean
                  end;
```

Ak p je premenná typu záznam, k jej položke s názvom poi sa dostaneme zápisom: p**.**poi. Napríklad, ak K je premenná typu TKarta, možno použiť zápisy: K.OsCislo, K.Meno, K.DatNar.Den, K.DatNar.Rok, K.Adresa.Obec, K.Priemer[1] a pod.

Príkaz with

Ak v časti programu používame opakovane tú istú položku alebo používame viacej položiek tej istej premennej typu záznam, príkaz with umožňuje zjednodušiť resp. skrátiť zápis k prístupu k týmto položkám.

Príkaz with má tvar: *with* z *do* p kde z je premenná typu záznam a p je príkaz

V príkaze p je dovolené označovať položky premennej z len identifikátormi, t.j. zápis z.polozka skrátiť na zápis polozka.

Napríklad:
```
with Bod do begin
  X := StrToFloat( InputBox('Vstup' , 'X-ová súradnica bodu: ' , '') );
  Y := StrToFloat( InputBox('Vstup' , 'Y-ová súradnica bodu: ' , '') );
end;
with Z , DatNar do begin OsCislo:=99; Den:=1; Mes:=1; Rok:=2000 end;
```

Pri zápise *with* z1,z2 *do* p možno v príkaze p skrátene označovať jednak položky záznamu z2, ale aj tie položky záznamu z1, ktoré sa nezhodujú v označení so žiadnou položkou záznamu z2.

Variantný záznam

Pri používaní údajového typu záznam môže vyvstať požiadavka na návrh takej štruktúry záznamu, v ktorej sa na daný prvok nemusí vzťahovať celý zoznam položiek uvedených v zázname, ale iba jeho určitá časť. Takto môže vzniknúť niekoľko alternatívnych typov – variant - jedného typu záznam. Takúto situáciu nám umožňuje riešiť tzv. variantný záznam, ktorý sa skladá z pevnej časti a z variantnej časti.

Variantný záznam má tvar:
```
record
   pevná časť;
   case rozlišovacia_položka : rozlišovací_typ_variantnej_časti of
      rozlišovacia_konštanta1 : ( položka1 : typ1;
                                  položka2 : typ2;
                                  ...          );
      rozlišovacia_konštanta2 : (položka1 : typ1;
                                  položka2 : typ2;
                                  ...          );
      ...
end;
```

kde rozlišovací typ variantnej časti môže byť len ordinálny typ.

Údajový typ súbor

Ak potrebujeme uchovať údaje aj po vypnutí počítača resp. po ukončení programu, musíme ich uložiť na vonkajšie pamäťové médium – disk. Zapisovať a čítať údaje z vonkajších pamäťových médií nám v Pascale umožňuje údajový typ súbor.

Súbory môžu byť:

- binárne
  - typové a
  - netypové
- textové

Typový súbor je štruktúrovaný údajový typ, ktorý sa skladá z teoreticky neobmedzeného počtu zložiek, všetky rovnakého typu. Prakticky je počet zložiek súboru obmedzený kapacitou vonkajšej pamäte.

Definícia typu súbor má tvar: *type* mt = *file of* tz

kde mt je meno typu a tz je typ zložky, ktorý nesmie byť typ súbor ani typ obsahujúci ako zložku typ súbor.

Napríklad:
```
type TCele = file of integer;                    typ súbor obsahujúci celé čísla
type TKartoteka = file of TKarta;
type TBody = file of record x,y,z:real end;      typ súbor obsahujúci trojice x,y,z
var F : file of string;                          premenná typu súbor obsahujúci reťazce
```

Základné príkazy pre prácu so súbormi umožňujú len sekvenčné spracovanie súboru, t.j. vytváranie aj čítanie súboru len zložku po zložke (od prvej k druhej atď.). V každom okamihu spracovania súboru je prístupná len jedna zložka súboru, na ktorú „ukazuje“ prístupová premenná súboru. Ak F je premenná typu súbor, k nej prislúchajúca prístupová premenná má označenie F^ a vznikne automaticky pri deklarácii premennej typu súbor.

Deklaráciou premennej typu súbor vznikne len abstraktný vonkajší súbor, sú určené len jeho logické vlastnosti. K spojeniu logického súboru s fyzickým – skutočne existujúcim na konkrétnom vonkajšom pamäťovom médiu, dôjde príkazom

AssignFile (logické_meno , fyzické_meno) kde logické meno je premenná typu súbor a fyzické meno je reťazec obsahujúci názov súboru na disku, prípadne doplnený aj o cestu.

Napríklad: assign(F,´CITATEL.DAT´); assign(SUB,´c:\tp\cele1.txt´)

Zápis údajov do súboru (napríklad F):

Zápis údajov do súboru (po stotožnení logického a fyzického mena súboru!) prebieha v dvoch fázach:

1. súbor sa pripraví na zápis príkazom *rewrite*(F). Vytvorí sa prázdny súbor (prázdna hodnota súboru), prístupová premenná F^ sa nastaví na „koniec“ súboru. Ak súbor už obsahuje nejaké zložky, budú odstránené!
2. vlastný zápis do súboru sa realizuje príkazom *write*(F,V), kde V je výraz rovnakého typu, ako je typ zložky súboru. Po vyhodnotení výrazu V, získaní konkrétnej hodnoty, dôjde k jej zápisu do súboru F. Bod 2 možno ľubovoľný počet krát opakovať.

Čítanie údajov zo súboru (napríklad F):

Aj sekvenčné čítanie údajov zo súboru prebieha v dvoch fázach:

1. súbor sa pripraví na čítanie príkazom *reset*(F). Príkaz spôsobí nastavenie prístupovej premennej F^ na začiatok súboru.
2. vlastné čítanie zo súboru sa realizuje príkazom *read*(F,X), kde X je premenná rovnakého typu ako je typ zložky súboru. Po vykonaní príkazu je v premennej X hodnota načítaná zo súboru. Čítanie zo súboru možno opakovať, až kým prístupová premenná nie je za poslednou položkou súboru.

Na zistenie pozície prístupovej premennej v súbore slúži funkcia eof (end of file), ktorá má hodnotu true, ak prístupová premenná je na konci súboru (hodnota F^ nie je definovaná), inak má hodnotu false. To možno výhodne využiť na načítanie všetkých zložiek súboru schémou:

```
reset(F);
while not eof(F) do begin        { pokiaľ nie je koniec súboru opakuj }
   read(F,X);                    { načíta hodnotu zo súboru do premennej X }
   write(X)                      { zobrazí načítanú hodnotu na obrazovke }
   end;
```

Ukončenie práce so súborom (napríklad F):

Po ukončení práce so súborom sa súbor musí zavrieť príkazom *CloseFile*(F).

Binárne súbory sú uložené po bitoch, čo síce zabezpečuje ich rýchle spracovanie, ale neumožňuje jednoduché editovanie ich obsahu. Editovanie - vytváranie, prezeranie a úpravu obsahu v jednoduchých textových editoroch umožňujú textové súbory.

Textové súbory majú vlastnosti typových súborov, naviac sú však členené na riadky. Ich typ sa označuje štandardným identifikátorom typu ***text***. V textovom súbore F okrem funkcie eof(F) je deklarovaná aj funkcia eoln(F) (end of line), slúžiaca na rozpoznanie konca riadku. Príkazy read(F,P) a write(F,V) sú rozšírené o príkazy readln(F,P) – z textového súboru F sa prečíta lexikálna jednotka (hodnota) zodpovedajúca typu premennej P a prejde sa na nový riadok, a writeln(F,V) – do textového súboru F sa zapíše hodnota výrazu V a oddeľovač riadkov eoln. Na otvorenie textového súboru na zápis na koniec textového súboru slúži príkaz append(F).

Dynamické premenné, typ ukazovateľ

Údajové typy, s ktorými sme sa zaoberali doteraz, patria medzi tzv. statické údajové typy. Premenné prislúchajúce statickým typom sú zavedené v úseku definícií a deklarácií a je im trvalo vyhradené miesto v pamäti počas behu programu. Počet a rozsah premenných sa počas práce v bloku, t.j. aj v programe, nemôže meniť.

Jazyk pascal umožňuje vytvárať premenné nie len v úseku deklarácií premenných, ale aj v príkazovej časti. Dokonca možno tieto premenné v príkazovej časti aj rušiť. Premenné s týmito vlastnosťami nazývame **dynamické premenné** a príslušné údajové typy nazývame dynamické údajové štruktúry.

Z toho, čo sme uviedli doteraz, vyplýva, že dynamické premenné nemožno zaviesť v úseku deklarácií a teda zabezpečiť prístup k ich hodnotám pomocou ich identifikátorov. Dynamické premenné vznikajú a zanikajú počas realizácie programu a sprístupnenie ich hodnôt sa robí pomocou nového údajového typu **smerník** (pointer, ukazovateľ).

Ak t je identifikátor nejakého typu, tak typ ukazovateľ definujeme zápisom:

```
type u = ^ t ;
```

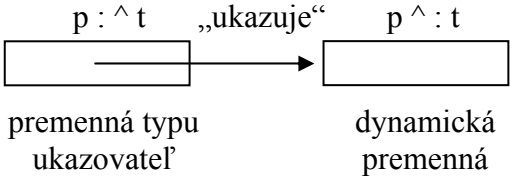
kde u je meno (identifikátor) typu ukazovateľ. Hovoríme aj, že typ t je zviazaný s typom u a množina hodnôt premennej typu ukazovateľ „ukazuje“ na prvky typu t.

Napríklad:

```
type   UKAZ1 = ^ integer;
       UKAZ2 = ^ PRVOK;
```

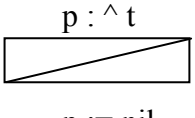
Ak p je premenná typu ukazovateľ, tak premenná typu zviazaného s typom ukazovateľ (premenná typu t) sa nazýva dynamická premenná a označuje sa p^.

Grafická interpretácia:



Pomocou hodnoty premennej typu ukazovateľ sprístupňujeme dynamickú premennú. V úseku definícií a deklarácií je zavedená len premenná typu ukazovateľ! Do množiny hodnôt premennej typu ukazovateľ patrí vždy aj hodnota *nil*, jediná konštanta typu ukazovateľ, ktorá neukazuje na nijaký prvok (na žiadnu dynamickú premennú).

Graficky:

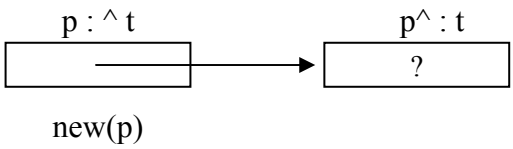


p := nil

Premenná p má síce priradenú hodnotu, ale „neukazuje“ na žiadnu dynamickú premennú.

Príkazom **new(p)**, kde p je premenná typu ukazovateľ, sa vytvorí dynamická premenná typu t s nedefinovanou hodnotou a ukazovateľ na túto premennú sa uloží do premennej p. Príkazom new(p) priraďujeme premennej p hodnotu, ktorou je zodpovedajúca dynamická premenná a ktorú označujeme p^.

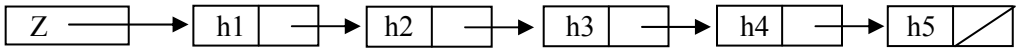
Graficky:



Príkazom **dispose(p)**, kde p je premenná typu ukazovateľ, sa zruší dynamická premenná p^, na ktorú ukazovala premenná p. Hodnota premennej p nie je po ukončení príkazu dispose(p) definovaná. Pamäťový priestor, ktorý zaberala dynamická premenná, sa dal k dispozícii na ďalšie použitie procesorom.

Lineárny jednosmernezreťazený zoznam

Z dynamických štruktúr sa najčastejšie využíva lineárny jednosmernezreťazený zoznam. Graficky ho možno znázorniť:



kde Z je premenná typu ukazovateľ (pointer), ktorá ukazuje na prvú dynamickú premennú v zozname, ktorého prvky sú typu záznam (record) s dvoma položkami: s hodnotou (h1 až h5) a s ukazovateľom na ďalší prvok zoznamu. Ukazovateľ posledného prvku zoznamu už nemá kde ukazovať a preto má hodnotu nil.
Postup v zozname je sekvenčný, od ukazovateľa na prvý prvok cez ukazovatele na nasledujúci prvok až na koniec zoznamu. V úseku definícií a deklarácií je zavedený len ukazovateľ na zoznam (premenná Z typu ^t) a typ dynamickej premennej (t). Vlastné prvky zoznamu, dynamické premenné, vznikajú podľa potreby počas práce programu príkazom new. Na pohyb v zozname používame pomocnú premennú typu ukazovateľ a premennú Z nechávame väčšinou ukazovať vždy na začiatok zoznamu (inak sa nemáme možnosť dostať na začiatok zoznamu).

Typ interval

určuje neprázdnu súvislú podmnožinu hodnôt nejakého ordinálneho typu.
Definícia typu interval má tvar: *type* mt = min..max;
kde mt je meno typu – identifikátor a min a max sú dolná a horná hranica hodnôt ordinálneho typu.
Napríklad:
type TCislice = '0'..'9'; TPocet = 1..100; TKladne = 1..MaxInt;

Vymenovaný typ

je typ definovaný vymenovaním hodnôt, preto jeho definícia má tvar:
type mt = (hodnota1, hodnota2, ..., hodnotaN); kde mt je meno typu – identifikátor
Napríklad:
type TDni = (Pondelok, Utorok, Streda, Stvrtok, Piatok, Sobota, Nedela);
TFarba = (Cervena, Oranzova, Zelena);
Každý vymenovaný typ je ordinálny a preto pre neho platia operácie succ, pred a ord.
Napríklad: ord (Pondelok) = 0, succ (Oranzova) = Zelena, Stvrtok < Piatok, succ (Nedela) nedefin.
Nevýhodou vymenovaného typu je, že jeho hodnoty nemožno priamo načítať z klávesnice ani zobraziť na monitore (výstup sa realizuje najčastejšie cez príkaz case).

Údajový typ množina

použijeme, ak potrebujeme pracovať s podmnožinami nejakého ordinálneho typu. Ak bázový typ má N rôznych hodnôt, k nemu prislúchajúci typ množina určuje 2^N podmnožín.
Definícia typu množina má tvar: *type* mt = *set of* bázový typ;
kde mt je meno typu – identifikátor a bázový typ určuje typ prvkov množiny – ordinálny typ.
Napríklad:
type TFarba = (Cervena, Oranzova, Zelena, Modra, Zlta);
TOtien = set of TFarba;
var Otien: TOtien;
hodnoty: [Cervena, Modra], [Oranzova..Zlta] alebo []

Znak in [‘0’..’9‘]
Pismena := [’A’..’Z’]
Operácie: + zjednotenie, * prienik, - rozdiel, = rovnosť, <> nerovnosť, <= „je obsiahnutá v“, >= „obsahuje“
Relácia: prvok *in* množina – „je prvkom množiny“ - nadobúda hodnoty true alebo false
Príklad použitia:

```
if Retazec[i] in [‘A’..’Z’, ‘a’..’z’] then... // písmeno
V procedúre (udalosti) Edit1KeyPress:
if not ( Key in [‘0’..’9’, #8] ) then Key := #0;          //ak nebola stlačená cifra alebo BS, nereaguje.
```

Literatúra:
Základy programovania – Turbo Pascal (www.gymparnr.edu.sk/informatika)