Pascal

Katedra aplikované kybernetiky

Ing. Miroslav Vavroušek

Verze 7

Proměnné

Proměnná uchovává nějakou informaci potřebnou pro práci programu. Má ve svém oboru platnosti unikátní jméno. (Připadne, musí byt dodrženy pravidla překrývaní jmen) Každá proměnná je nějakého konkrétního datového typu.

Datové typy a struktury

Typ určuje jaké data je možné do proměnné uložit a jaké operace lze s proměnnou vykonávat.

Jednoduché datové typy

Identifikátor	Charakteristika	Rozsah	Ukázka deklarace a iniciace
Boolean	Booleovská hodnota	Dva stavy. True a False. (Pravda a nepravda, 1 a 0)	Deklarace: Ok: Boolean; Iniciace: Ok:= True;
Integer	Celé číslo	-32768 až 32767	Deklarace: X: Integer; Iniciace: X:= 5;
Real	Reálné číslo	$2.9*10^{-39}$ až $1.7*10^{38}$	Deklarace: X: Real; Iniciace: X:= 2.92;
String	Text	ASCII znaky (nemůžete používat české znaky)	Deklarace: Text: String; Iniciace: Text:= ‘Ahoj’;
Char	Jeden znak textu	ASCII znak	Deklarace: Znak: Char; Iniciace: Znak:= ‘a’;

Složené datové typy

Často nazývané také záznamy. Umožňují definovat komplexnější datové typy. Záznamy jsou složeny z proměnných jednoduchých i složených datových (dříve nadefinovaných) typů. Zapisují se do deklarační části programu. K jednotlivým vnořeným atributům se přistupuje pomocí operátoru „**.**“

Složený datový typ

```
type
  Jmeno = record
    JmenoPromenne: DatovyTyp;
    //adt.
  end;
```

```
type
  TBod = record
    X: Integer;
    Y: Integer;
  end;
```

Jméno: TBod	Vlastnosti:
X: Integer;	
Y: Integer;	

Přistup k jednotlivým vlastnostem

```
Jmeno.JmenoAtributu
```

```
A:= Bod.X; {Vlozeni hodnoty X
            souřadnice do proměnné A}
```

Využití dříve definovaných záznamů při definici nových záznamů

```
type
  TBod = record  //Základní datový typ
    X: Integer;
    Y: Integer;
  end;

type
  TCara = record  {Při definici typu je použit
    Bod1: TBod;    dříve definovaný datový typ}
    Bod2: TBod;
  end;
```

Pole

Datová pole představují množiny prvků stejného datového typu. K jednotlivým prvkům pole přistupujeme pomocí indexu. Každé pole je nějakého konkrétního rozměru. Jednorozměrná pole si můžeme představit, jako sloupec tabulky kde každá buňka je zvoleného datového typu. Dvourozměrné pole lze reprezentovat jako tabulku a třírozměrné jako kvádr z krychliček jednotného typu. Pole můžeme rozdělit na statická a dynamická

Statická pole

Velikost statických polí je známa již při kompilaci. Rozsah je zadán pomocí dvou ordinálních proměnných stejného typu, které určují rozsah

Statická pole - deklarace	
`JmenoPole: array [OD..DO] of JmenoDatovehoTypu;`	`Pole1: array [1..15] of Integer;` `Pole2: array ['A'..'Z'] of String;`
Statická pole - použití	
`JmenoPole[index]`	`Pole1[5]:= 3; {Do prvku s indexem 5 je uložena hodnota 3}` `A:= Pole2['C']; {Do proměnné A je uložena hodnota prvku s indexem 'C'}`
Statická pole – projití celého pole cyklem for-to-do	

`for A:= Low(Pole1) to High(Pole1) do`
`begin`
`  //Tělo cyklu`
`end;`	

Operátory

Aritmetické operátory

Identifikátor	Operace	Ukázka použití

+	Sčítaní. Datový typ výsledku je určen datovými typy operandů. Pokud je alespoň jeden s operandu typu Real výsledek je také typu Real.	`X:= 5 + 6;  //X = 11`
		`X:= 5.2 + 6; //X = 11.2`
		`C:= A + B;`

-	Odčítání. Datový typ viz „+“.	`X:= 5 - 6;  //X = -1`
		`C:= A - B;`

*	Násobení. Datový typ viz „+“.	`X: = 5 * 6;  //X = 30`
		`C:= A * B;`

/	Děleni. Výsledek je typu Real.	`X: = 5 / 6;  //X = 0.83`
		`C:= A / B;`

div	Celočíselné dělení. Výsledkem je cela část čísla po dělení. Výsledek je typu Integer.	`X: = 47 div 6; //X = 7`
		`X: = -9 div 4; //X = -2`
		`C:= A div B;`

mod	Zbytek po celočíselném dělení. Často nazýván také Modulo.	`X: = 47 mod 6; //X = 5`
		`X: = -9 mod 4; //X = -1`
		`C:= A mod B;`

Booleovské operátory

Identifikátor	Operace	Ukázka použití

not	Negace	`X: = not True;  //X = False`
		`B:= not A;`

and	Logicky součin (oba musí platit)	`X: = True and False; //X = False`
		`C:= A and B;`

or	Logicky součet (alespoň jeden musí platit)	`X: = True or False;  //X = True`
		`C:= A or B;`

>	Větší	`X: = 5 > 6;  //X = False`
		`C:= A > B;`

<	Menší	`X: = 5 < 6;  //X = True`
		`C:= A < B;`

>=	Větší a rovno	`X: = 5 >= 6;  //X = False`
		`C:= A >= B;`

<=	Menší a rovno	`X: = 5 <= 6;  //X = True`
		`C:= A <= B;`

=	Rovná se	`X: = 5 = 6;  //X = False`
		`C:= A = B;`

<>	Nerovná se	`X: = 5 <> 6;  //X = True`
		`C:= A <> B;`

Řízení toku kódů

Konstrukce pro řízení toku kódu slouží buď k větvení (Podmínky) anebo k opakování (Cykly) částí programu. Pomocí těchto konstrukci je dosaženo toho, že program vykoná pouze určité operace a v určitém počtu, podle vstupních dat.

Podmínky

Podmínky slouží k větvení programu. Zda se má daná část kódu vykonat. Při pohledu na tvorbu programu jako na dialog programátora a počítače umožňují podmínky programátorovi pokládat dotazy počítači. Podmínka musí být výraz, který je možné ohodnotit booleovskou hodnotou (‚Ano/Ne').

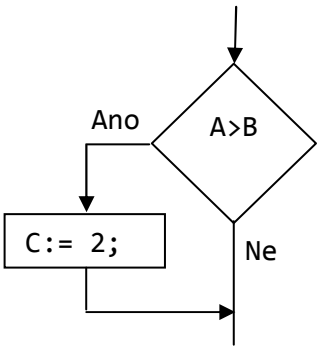
If-then

Podmínka If-then umožňuje rozhodnout, zda vykonat část kódu. Volitelně muže obsahovat část vykonanou při nesplnění podmínky

Podmínka if

```
if Podminka then
begin
  // Pokud je podmínka splněna
end;
```

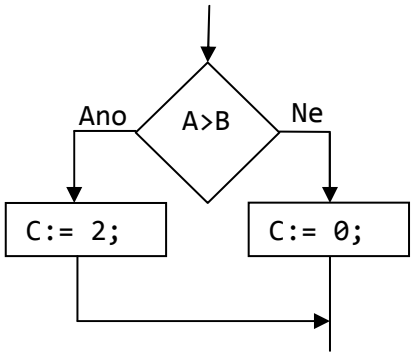
```
if A>B then
begin
  C:= 2;{Pokud je A větší než B do C vlož 2}
end;
```



Úplná podmínka if-else

```
if Podminka then
begin
  // Pokud je podmínka splněna
end else
begin
  // Pokud je podmínka nesplněna
end;
```

```
if A>B then
begin
  C:= 2;{Pokud je A větší než B do C vlož 2}
end else
begin
  C:= 0;{Pokud A není větší než B do C vlož 0}
end;
```

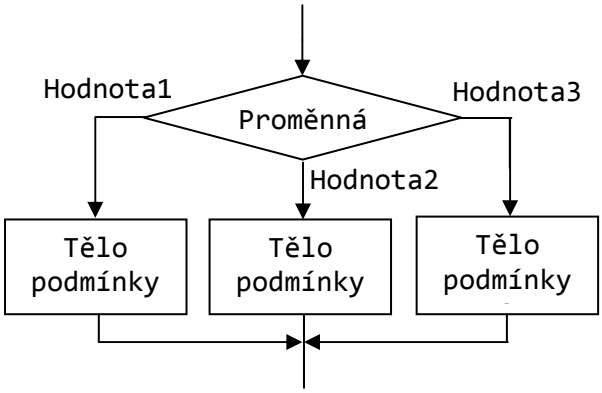


Case-of

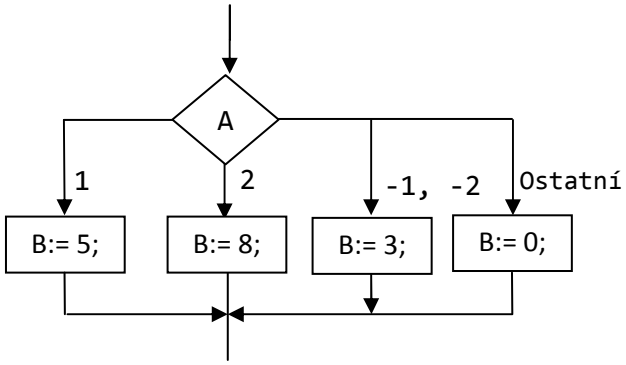
Podmínka case-of umožňuje větvení do více než dvou cest. Vhodná cesta je vybrána podle hodnot v návěstí. Také je možné vytvořit větev, která je vykonána pokud hodnota proměnné neodpovídá žádnému navěstí. Proměnná požita v podmínce case-of musí být ordinálního typu.

Podmínka case-of

```
case Proměnná of
  Hodnota1: begin
       //Tělo podmínky 1
     end;
  Hodnota2: begin
       //Tělo podmínky 2
     end;
  Hodnota3: begin
       //Tělo podmínky 3
     end;
end;
```




```
case A of
  1: begin
       B:= 5; {Tělo podmínky
               pro A = 1}
     end;
  2: begin
       B:= 8; {Tělo podmínky
                 pro A = 2}
     end;
  -1, -2: begin
       B:= 3; {Tělo podmínky
               pro A = -1, -2}
     end;
  else begin
       B:= 0; {Tělo podmínky
               pro A = ostatní}
     end;
end;
```



Cykly

Cykly se požívají k určitému počtu opakovaní zvolené části kódů. V jazyku Object Pascalu existují tři typy cyklů. Vhodný typ cyklu je zvolen na základě vlastností vyžadovaných v konkrétní situaci.

For-to-do

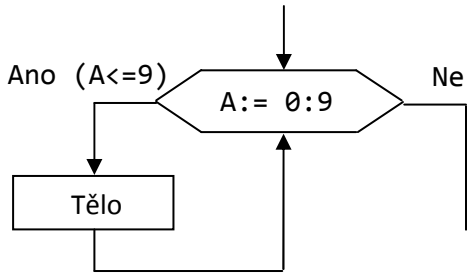
Vlastnosti cyklu for:

- Nejčastěji používaný cyklus
- Používá se všude tam kde je znám počet opakovaní před započetím cyklu
- Vždy používá některou z proměnných (tzv. řídící proměnná cyklu), která musí být deklarována jako proměnná ordinálního typu
- Vždy má definovánu počáteční a koncovou hodnotu
- Cyklus provádí změnu řídicí proměnné automaticky sám
- Hodnotu řídící proměnné cyklu je zakázáno v těle cyklu měnit. Je jí možné pouze číst.
- Nikdy nemůže nastat nekonečný cyklus
- Krok je vždy 1
- Tělo cyklu nemusí proběhnou ani jednou

Cyklus for-to-do vzestupný

```
for Proměnná:=A to B do
begin
  {Hodnota v proměnná stoupá
  v jednotlivých opakováních od A
  do B}
end;
```

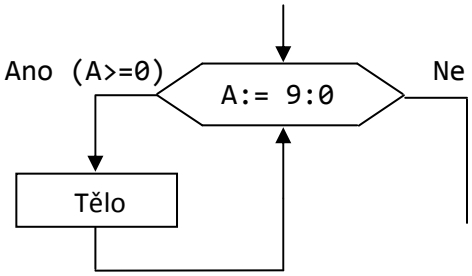
```
for A:=0 to 9 do
begin
  {Hodnota v A stoupá v jednotlivých
   opakováních od 0 do 9}
end;
```



Cyklus for-downto-do sestupný

```
for Proměnná:=A downto B do
begin
  {Hodnota v proměnná klesá
  v jednotlivých opakováních od A
  do B}
end;
```

```
for A:=9 downto 0 do
begin
  {Hodnota v A klesá v jednotlivých
   opakováních od 9 do 0}
end;
```



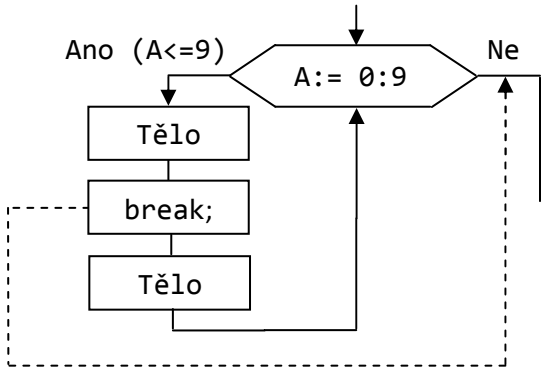
Příkazy break a continue

Tyto příkazy umožňují přerušení vykonávání těla cyklu. Příkaz break přerušuje cely cyklus for-to-do. Vztahují se k cyklu, ve kterém jsou zapsány (k nejbližší úrovni vnoření).

Cyklus for-to-do příkaz break

```
for Proměnná:=A to B do
begin
  //Tělo cyklu
  break;
  //Tělo cyklu
end;
```

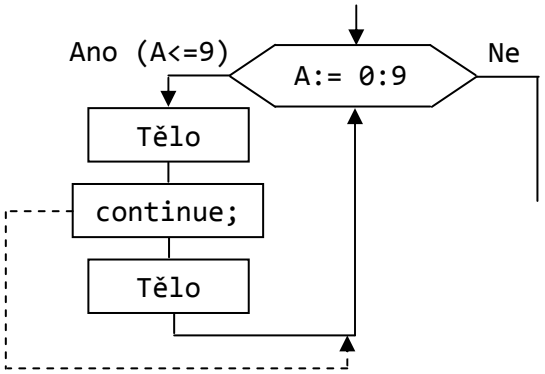
```
for A:=0 to 9 do
begin
  //Tělo cyklu
  if A=5 then
  begin
    break; {Při A = 5 ukončí cyklus.
            Nedojde tedy nikdy na 6-9}
  end;
  {Tělo cyklu (neproběhne pro A = 5)}
end;
```



Cyklus for-to-do příkaz continue

```
for Proměnná:=A to B do
begin
  //Tělo cyklu
  continue;
  //Tělo cyklu
end;
```

```
for A:=0 to 9 do
begin
  //Tělo cyklu
  if A=5 then
  begin
    continue; {Při A = 5 ukončí kolo
               cyklu a pokračuje při
               A = 6}
  end;
  {Tělo cyklu (neproběhne pro A = 5)}
end;
```



Repeat-until

Vlastnosti cyklu repeat-until:

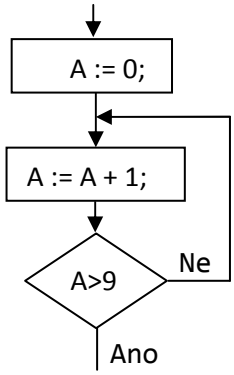
- Používá se všude, kde je před započetím cyklu, není znám počet opakovaní a je vhodné projít alespoň jednou tělo cyklu
- Podmínka je na konci cyklu. Podmínkou je myšlen jakýkoli výraz o kterém lze rozhodnout „Ano/NE“
- Cyklus probíhá při nesplnění podmínky
- Při nevhodném zadání může vest na nekonečný cyklus
- Hodnotu proměnné vyžité v podmínce je třeba v cyklu měnit
- Tělo cyklu proběhne alespoň jednou

Cyklus repeat-until

```
repeat
begin
  //Tělo cyklu
end until Podmínka;
```

```
A:= 0; //Iniciace proměnné

repeat
begin
  A:= A + 1; {Změna hodnoty proměnné
              použité v podmínce.
              Inkrementace A}
end until A>9; //Podmínka
```



While-do

Vlastnosti while-do:

- Používá se všude, kde je před započetím cyklu, není znám počet opakovaní a je možné že tělo cyklu nebude potřebovat vůbec vykonat
- Podmínka je na začátku cyklu. Podmínkou je myšlen jakýkoli výraz, o kterém lze rozhodnout „Ano/NE“
- Cyklus probíhá při splnění podmínky
- Při nevhodném zadání může vést na nekonečný cyklus
- Hodnotu proměnné vyžité v podmínce je třeba v cyklu měnit
- Tělo cyklu nemusí proběhnou ani jednou

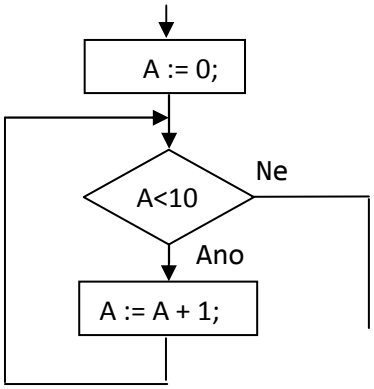
Cyklus while-do

```
while Podmínka do //Podmínka
begin
  //Tělo cyklu
end;
```

```
A:= 0; //Iniciace proměnné

while A<10 do //Podmínka
begin
  //Tělo cyklu

  A:= A + 1; {Změna hodnoty proměnné
              použité v podmínce.
              Inkrementace A}
end;
```



Procedury

Vlastnosti procedur:

- Používá se k zapouzdření dílčích částí algoritmu
- Využití procedur pomáhá k rozdělení a strukturovaní kódu
- Pomáhají také ke zvýšení přehlednosti a znuvupoužitelnosti vytvořeného kódu
- Nemají návratovou hodnotu (nevrací volajícímu žádnou informaci)
- Jedná se o předpis toho, co se má udělat

Procedury

```
//Implementace procedury 1
procedure Nazev( Atributy);
begin
  //Telo procedury
end;

//Implementace procedury 2
procedure Nazev2; //Bez atributu
begin
  //Telo procedury
end;

//Volani procedur
begin
  Nazev( Atributy);
  Nazev2;
end.
```

```
//Implementace procedury Draw
procedure Draw( colored: Boolean);
begin
  //Telo procedury
end;

//Volani procedur
begin
  Draw( True);
end.
```

Funkce

Vlastnosti funkcí:

- Mají pro kód obdobný přínos jako procedury
- Mají návratovou hodnotu (mají nějaký výsledek)
- Jedná se o operaci

Funkce

```
//Implementace funkce 1
function Nazev( Atributy): NavratovyTyp;
begin
  Nazev:= //Telo funkce
end;

//Implementace funkce 2
function Nazev2: NavratovyTyp2;
begin
  Nazev2:= //Telo funkce
end;

//Volani funkci
var
  A: NavratovyTyp;
  B: NavratovyTyp2;
begin
  A:= Nazev( Atributy);
  B:= Nazev2;
end.
```

```
//Implementace funkce 1
function NaDruhou( A: Integer): Integer;
begin
  NaDruhou:= A*A;  //Telo funkce
end;

//Volani funkci
var
  A: Integer;
  A2: Integer;
begin
  A2:= NaDruhou ( A);
end.
```