

Další skupinu strukturovaných příkazů tvoří příkazy opakovací neboli příkazy cyklu. Příkazy cyklu předepisují opakované provádění příkazu nebo posloupnosti příkazů. Kromě příkazů, které mají být opakovaně prováděny, je součástí cyklu specifikace řídící toto opakování. Cyklus může být **nepodmíněný** nebo **podmíněný**. **Nepodmíněný** cyklus (**for – to – do**) použijeme v případě, je-li předem znám počet opakování, **podmíněný** cyklus (**while – do** resp **repeat – until**) tehdy, je-li opakování vázáno na splnění nějaké podmínky). Velmi efektivním (ovšem někdy dosti nebezpečným) příkazem cyklu je **rekurzivní volání funkce či procedury**.

Nepodmíněný cyklus – příkaz for: Příkaz **for** zajistí opakované provádění jednoduchého nebo složeného příkazu:

for <řídící proměnná>:=<počáteční hodnota> **to** <koncová hodnota> **do** <příkaz>
event.

for <řídící proměnná>:=<počáteční hodnota> **downto** <koncová hodnota> **do** <příkaz>

Řídící proměnná musí být ordinálního typu a musí být deklarována lokálně, a to v bloku, který obsahuje příkaz **for**. Počáteční a koncová hodnota musí být kompatibilní s přiřazením ordinálnímu typu. Příkaz se provádí pro každou jednotlivou hodnotu intervaku definovaného počáteční a koncovou hodnotou. Řídící proměnná vstupuje do opakování vždy s počáteční hodnotou. Použitím **to** každým průchodem hodnotu řídící proměnné o jedničku zvyšujeme, použitím **downto** naopak snižujeme. Je-li počáteční hodnota vyšší než konečná, příkaz se neprovede. Změna hodnoty řídící proměnné vnořeným příkazem se považuje za chybu.

Příklad 2 – výpočet faktoriálu nepodmíněným cyklem: naprogramujeme jako funkci

```
function Faktorial(n:Integer):LongInt;
var   i:Integer;
      PomPromenna:LongInt;
begin
  PomPromenna:=1;
  for i:=1 to n do PomPromenna:=PomPromenna*i;
  Faktorial:=PomPromenna;
end;
```

Příkaz While: Také příkaz **While** obsahuje výraz, který řídí opakované provádění příkazu:

while <booleovský výraz> **do** <příkaz>

Výraz musí být standardního booleovského typu a vyhodnocuje se předtím, než se provede vnořený příkaz. Ten může být jednoduchý nebo složený. Vnořený **příkaz se provádí** tak dlouho, **dokud je hodnota** výrazu **True**. Je-li hodnota výrazu **False**, příkaz se neprovede a cyklus končí.

Příklad 3 – výpočet faktoriálu podmíněným cyklem **While**: naprogramujeme opět jako funkci

```
function Faktorial(n:Integer):LongInt;
var i:      Integer                                {index - počítadlo cyklů}
    PomProm:LongInt;
begin
  PomProm:=1;i:=1;
```

```

While i<n do                                {dokud je splněna podmínka, dělej}
  begin
    inc(i);                                    {zvýšení hodnoty počítadla o jedničku}
    PomProm:=PomProm*i;
  end;
  Faktorial:=PomProm;
end;

```

Na rozdíl od předchozího cyklu, kdy **nesmíme** měnit hodnotu řídicí proměnné, v tomto cyklu **musíme** měnit hodnotu počítadla, které má zde analogickou úlohu. Pokud bychom hodnotu neměnili, program uvízne v nekonečném cyklu.

Příkaz Repeat obsahuje podobně jako **While** výraz, který řídí opakované provádění vnořeného příkazu:

```

repeat <příkaz> until <booleovský výraz>

```

Výraz musí být standardního booleovského typu a vyhodnocuje se až po provedení vnořeného příkazu. Vnořený příkaz se tedy provede vždy alespoň jednou. Provádění vnořeného příkazu se opakuje tak dlouho, **dokud je hodnota booleovského výrazu False**.

Příklad 4 – výpočet faktoriálu podmíněným cyklem **Repeat**: naprogramujeme opět jako funkci

```

function Faktorial(n:Integer):LongInt;
var   i:Integer (index)
      PomProm:LongInt;
begin
  PomPromenna:=1;i:=1;
  Repeat                                {opakuj}
    inc(i);
    PomProm:=PomProm*i;
  Until i>n;                             {dokud není splněna podmínka}
  Faktorial:=PomProm;
end;

```

Dobře si všimněme některých rozdílů mezi cyklem **while-do** a **repeat-until**. První je čistě syntaktický: v obou případech provádíme v cyklu dva příkazy – příkaz procedury inc(i) a přiřazovací příkaz PomProm:=PomProm*i. V cyklu je proto musíme uzavřít do „programových závorek“, což jsou v Pascalu ve většině případů klíčová slova **begin – end**. (viz cyklus while – do). Cyklus repeat – until je jednou z mála výjimek – zde funkci závorek mohou převzít přímo klíčová slova repeat – until. Dále - opakování v cyklu **while – do** je vázáno na **splnění podmínky**, opakování v **repeat – until** na její **nesplnění**. Použijeme-li nicméně podmínku místo její negace či naopak, není to chyba nijak záluďná a snadno ji odhalíme (V tomto případě by např. pro každé n>1 vracela funkce jedničku). Záludnější je však poslední a často podceňovaný rozdíl: Cyklus while – do testuje podmínku na **začátku cyklu**, repeat – until až na jeho **konci**. Je-li v cyklu **while – do** podle b) zadáno n=0 nebo n=1, podmínka i<n není splněna nikdy a tento cyklus **vůbec neproběhne**. V proměnné PomProm zůstane jednička, kterou pak dle posledního přiřazovacího příkazu vrátí funkce Faktoriál, tj dle b) je 0! =1!=1, což je správně. Cyklus **Repeat – until** však **bez ohledu na jakoukoli podmínku proběhne alespoň jednou**. Dle c) je tak v každém případě proměnná i nastavena

na dvojku a do PomProm přiřazen součin $1*2=2$ a teprve potom je proměnná i porovnávána s parametrem n. Jeli tedy zadáno $n=0$ nebo $n=1$, pak funkce vrací špatný výsledek, neboť „tvrdí“, že $0! = 1! = 2$. Tuto chybu lze sice v tomto případě velmi jednoduše napravit (ponecháváme čtenáři), nicméně je dobře na tuto záludnost vždy pamatovat (v následujícím textu uvedeme některé další příklady).

Příkaz With je posledním strukturovaným příkazem. Umožňuje zkrácený přístup k položce záznamu či vlastnosti objektu.

With <Proměnná typu záznam nebo objekt> **do** <příkaz>

V argumentu příkazu je třeba použít identifikátor proměnné typu záznam nebo objekt, vnořený příkaz bývá většinou složený. Tento příkaz bývá užitečný až u složitějších příkladů. Zde ilustrujeme jeho použití na jednoduchém modelovém příkladu.

Příklad 5 – jednoduchá reakce na zmáčknuté tlačítko. Sestavme jednoduché uživatelské



rozhraní se dvěma tlačítky a jedním objektem typu TStaticText dle připojeného obrázku. Na zmáčknutí některého tlačítka má program reagovat změnou několika vlastností nápisu na formuláři (změna textu, barvy, fontu apod.). Tyto změny bude provádět procedura spojená onClick s příslušným tlačítkem. Pro levé tlačítko tedy např.



```
procedure TForm1.Button1Click(Sender: TObject);
begin
    StaticText1.Caption:='Ted' jsi zmáčkl levé';
    StaticText1.Color:=clYellow;
    StaticText1.Font.Name:='Courier';
    StaticText1.Font.Color:=clBlue;
end;
```

S použitím příkazu **Width**:

```
procedure TForm1.Button1Click
    (Sender: TObject);
begin
    With StaticText1 do
    begin
        Caption:='Ted' jsi zmáčkl levé';
        Color:=clYellow;
        Font.Name:='Courier';
        Font.Color:=clBlue;
    end;
end;
```

```
procedure TForm1.Button1Click
    (Sender: TObject);
begin
    With StaticText1 do
    begin
        Caption:='Ted' jsi zmáčkl levé';
        Color:=clYellow;
        With Font do
        begin
            Name:='Courier';Color:=clBlue;
        end;
    end;
end;
```

Efektivita tohoto přístupu nicméně vynikne až při větším počtu položek.

6. Uživatelské funkce

Jazyk Pascal je založen, ostatně jako všechny vyšší programovací jazyky, na tzv. podprogramech. Podprogramy realizují myšlenku rozdělit program na několik menších problémů. Hlavní program pak ve vhodném pořadí volá jednotlivé podprogramy. To má značné výhody. Pokud se nějaká činnost v programu často opakuje, stačí ji naprogramovat pouze jednou v podprogramu a pak tento podprogram opakovaně volat. V Pascalu známe dva druhy podprogramů – procedury a funkce. Procedury a funkce tvoří soubory instrukcí, které potřebujeme v programu provádět na různých souborech dat nebo na různých místech programu. Řada procedur a funkcí je standardních – jsou k dispozici okamžitě po instalaci Delphi. Další (uživatelské) může psát programátor. Uživatelské procedury a funkce následují za deklaracemi typů, konstant a proměnných. Procedura nebo funkce může být po deklaraci použita kdekoliv v následujícím textu bloku programu. Rozdíl mezi procedurou s funkcí je v tom, že funkce vrací hodnotu a lze ji použít přímo ve výrazech. Procedura se volá k vykonání jedné nebo více operací. Je schopna nastavovat hodnoty jedné nebo i několika proměnných, nelze ji však použít ve výrazech.

Procedury a funkce mohou být vnořeny do kteréhokoli bloku programu. Každá deklarace procedury nebo funkce musí mít hlavičku, po které následuje blok příkazů. Hlavička uživatelské funkce má obecně tvar

Function <jméno funkce> (<seznam argumentů a jejich typů>):<typ výstupní proměnné>;

Struktura procedury a funkce může být obecně téměř shodná se strukturou celého programu. Mezi hlavičkou funkce a seznamem příkazů lze lokálně deklarovat typy, konstanty, proměnné, další procedury a funkce. Na rozdíl od programu zde nesmí být seznam používaných knihoven (klíčové slovo `uses`).

Příklad 1: Program na výpočet kombinačních čísel: V tomto programu budeme opakovaně počítat faktoriál, a to z různých argumentů. Pro výpočet faktoriálu proto využijeme uživatelsky definované funkce.



Výpočet spustíme na stisk tlačítka. Zjištěné počty permutací, variací a kombinací zobrazíme v objektech Edit. Kromě toho jsou na formuláři umístěny objekty Radio Button, pomocí nichž může uživatel vybírat jednu z několika možností:

```
procedure TForm1.KombinacniCisla(Sender: TObject);  
var n,k :Integer;  
    Permutace,Variace,Kombinace :LongInt;  
    PrevodRetezec :String;  
Function Faktorial(n:Integer):LongInt; {uživatelská funkce pro výpočet faktoriálu}  
var i:Integer;  
    PomPromenna:LongInt;
```

```

begin
    PomPromenna:=1;
    for i:=1 to n do PomPromenna:=PomPromenna*i;
    Faktorial:=PomPromenna;
end;

begin
    {načítání vstupních hodnot}
    n:=SpinEditN.Value           {Okna se zadávanými parametry jsou objekty typu TSpinEdit
                                   v liště Samples}
    k:=SpinEditK.Value;          {Jejich původní jména SpinEdit1 resp. SpinEdit2
                                   jsou změněna z Object Inspectoru.}
    {výpočet a tisk permutací. Jejich obsahem je proměnná Value typu integer,
                                   rozsahu MinValue - MaxValue}

    Permutace:=Faktorial(n);
    Str(Permutace,PrevodRetezec); {Editovací okna pro výpis výsledků
                                   jsou objekty typu TEdit,}

    EditPerm.Text:=PrevodRetezec; {přejmenované naEditPerm, EditVar a EditKomb.}
    {výpočet a tisk variací}      {vzorečky u těchto oken jsou sestaveny z objektů typu TLabel}
    Variace:=Trunc(Faktorial(n)/Faktorial(n-k));
    Str(Variace,PrevodRetezec);
    EditVar.Text:=PrevodRetezec;
    {výpočet a tisk kombinací}
    Kombinace:=Trunc(Faktorial(n)/Faktorial(k)/Faktorial(n-k));
    Str(Kombinace,PrevodRetezec);
    EditKomb.Text:=PrevodRetezec;
    {naplnění výsledkového okna dle vybrané možnosti}
    if RadioPerm.Checked then EditVysl.Text:=EditPerm.Text;    {Knoflíky pro výběr
                                                                    jedné z možností jsou}
    if RadioVar.Checked then EditVysl.Text:=EditVar.Text;      {objekty typu TRadioButton
                                                                    z lišty Standard}
    if RadioKomb.Checked then EditVysl.Text:=EditKomb.Text;    {přejmenované
                                                                    na RadioPerm,-Var, -Komb}
end;

```

Tlačítko s titulkem '**Konec**' je událostí onClick spojeno s procedurou, která jediným příkazem ukončí program:

```

procedure TForm1.Konec(Sender: TObject);
begin
    Application.Terminate;
end;

```

Kombinační čísla:

[Zdrojový kód](#)

[Spustitelný kód](#)